Designing and Optimizing Deep Learning Methods for Genomic Sequencing Data

Hüseyin Anil Gündüz

Designing and Optimizing Deep Learning Methods for Genomic Sequencing Data

Hüseyin Anil Gündüz

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität München

eingereicht von
Hüseyin Anil Gündüz
am 30.09.2024

Erster Berichterstatter: Prof. Dr. Bernd Bischl
Zweiter Berichterstatter: Prof. Dr. Alice Carolyn McHardy
Dritter Berichterstatter: Prof. Dr. Christoph Lippert

Tag der Disputation: 09.04.2025

Acknowledgments

I am deeply grateful to

... Prof. Dr. Bernd Bischl, Dr. Mina Rezaei, Dr. Philipp Münch, and Martin Binder

... for their continuous supervision, guidance, and inspiration.

Their efforts, support, and supervision were crucial to making this Ph.D. thesis possible.

I am thankful to Prof. Alice C. McHardy and Prof. Dr. Christoph Lippert

... for agreeing to review my thesis,

and to Prof. Dr. Christian Heumann and PD Dr. Fabian Scheipl

... for their kindness in agreeing to be on the examination panel.

Special thanks goes to my colleagues and collaborators:

... René, Özgür, Emilio, Yin, Julia, Lisa, Sheetal, Amadeu, and Amirhossein.

I fully appreciate my wife Hande for her eternal support throughout this Ph.D. journey,

... in addition to simply making my life much more amazing.

I also appreciate our cat Findik for her emotional support in the form of soothing purrs

... and happily sitting on my lap while I endeavored to advance science.

I am also grateful to my parents and brother for their support,

... which has been essential in getting me to where I am today.

Summary

While modern deep learning techniques have significantly impacted fields such as natural language processing and computer vision, their application to biology still needs to be expanded. To bridge the gap between these fields, several deep learning approaches are proposed and tailored to genomics, based on recent advances in machine learning research and the characteristics of sequential genomic data. The main contributions of the dissertation aim at advancing several aspects of deep learning for sequential genomic data: self-supervised learning, uncertainty quantification, and automated model design or, more generally, optimization of architectures and hyperparameters.

A challenge that this thesis aims to address is the effective use of unlabeled genomic data to improve model performance. In this context, self-supervised approaches for sequential genomic data are investigated. These methods improve performance especially when the amount of labeled data is limited and the acquisition of large amounts of annotated data is not feasible due to factors such as increased cost. A major contribution of the thesis, Self-GenomeNet, is a self-supervised learning method tailored for genomic data, using reverse-complement sequences within self-supervised learning.

Another aspect explored in this thesis is the design and optimization of deep learning architectures and hyperparameters for genomics. While models such as ResNets or Visual Transformers are the standard architectures in computer vision and various transformer models such as BERT or GPT in natural language processing, there is still no consensus on a standard architecture in computational biology, a field with many different tasks and subfields. As a result, it can be difficult for researchers to train successful machine learning models in genomics using out-of-the-box architectures and hyperparameters. To address this problem, the first part of the thesis investigates automated model design methods. These methods optimize model architectures for the specific dataset and task. An important contribution of this work is a model-based optimization approach called GenomeNet-Architect, which simultaneously optimizes both the model architecture through proposed hyperparameters and the optimization hyperparameters. In another contribution of this thesis, various neural architecture search methods are optimized using our proposed search space and benchmarked against expert-designed architectures. Both papers suggest that automated architecture design methods find better models than those designed by experts.

Other contributions of this thesis deal with uncertainty quantification methods applied to genomic data. Applications in health and biology are often safety-critical, so the reliability of deep learning models should be investigated and improved. To this end, various uncertainty quantification methods for predicting regulatory activity are investigated, and a novel deep learning method for improving the calibration of predictions is studied in the context of sequential genomic data.

Finally, contributions are made to the development of user-friendly software that can handle different formats of genomic data, including the integration of key parts of several of the methods outlined in this thesis into this software.

Zusammenfassung

Obwohl moderne Deep-Learning-Techniken bereits einen großen Einfluss auf Bereiche wie die Verarbeitung natürlicher Sprache und maschinelle Bildverarbeitung haben, muss ihre Anwendung in der Biologie erst noch entwickelt werden. Um die Lücke zwischen diesen beiden Bereichen zu schließen, werden verschiedene Deep-Learning-Ansätze vorgeschlagen und auf die Genomik zugeschnitten, die auf den jüngsten Fortschritten in der maschinellen Lernforschung und den Eigenschaften sequenzieller Genomdaten basieren. Die Hauptbeiträge der Dissertation zielen darauf ab, verschiedene Aspekte des Deep Learning für sequenzielle Genomdaten voranzutreiben: selbstüberwachtes Lernen, Quantifizierung von Unsicherheiten und automatisiertes Modelldesign oder, allgemeiner, Optimierung von Architekturen und Hyperparametern.
Eine Herausforderung, die in dieser Arbeit angegangen werden soll, ist die effektive Nutzung unmarkierter Genomdaten zur Verbesserung der Modellleistung. In diesem Zusammenhang werden selbstüberwachende Ansätze für sequenzielle Genomdaten untersucht. Diese Methoden verbessern die Leistung insbesondere dann, wenn die Menge an markierten Daten begrenzt ist und die Beschaffung großer Mengen annotierter Daten aus Kostengründen nicht möglich ist. Ein wichtiger Beitrag der Arbeit, Self-GenomeNet, ist eine selbstüberwachte Lernmethode, die auf genomische Daten zugeschnitten ist und revers-komplementäre Sequenzen innerhalb des selbstüberwachten Lernens verwendet.
Ein weiterer Aspekt, der in dieser Arbeit untersucht wird, ist der Entwurf und die Optimierung von Deep-Learning-Architekturen und Hyperparametern für die Genomik. Während Modelle wie ResNets oder Visual Transformers Standardarchitekturen in der Computer Vision und verschiedene Transformatormodelle wie BERT oder GPT in der natürlichen Sprachverarbeitung sind, gibt es noch keinen Konsens über eine Standardarchitektur in der Computerbiologie, einem Gebiet mit vielen verschiedenen Aufgaben und Teilgebieten. Infolgedessen kann es für Forscher schwierig sein, erfolgreiche maschinelle Lernmodelle in der Genomik mit Standardarchitekturen und Hyperparametern zu trainieren. Um dieses Problem zu lösen, werden im ersten Teil der Arbeit Methoden zur automatischen Modellentwicklung untersucht. Diese Methoden optimieren Modellarchitekturen für den jeweiligen Datensatz und die jeweilige Aufgabenstellung. Ein wichtiger Beitrag dieser Arbeit ist ein modellbasierter Optimierungsansatz namens GenomeNet-Architect, der sowohl die Modellarchitektur durch vorgeschlagene Hyperparameter als auch die Optimierungshyperparameter gleichzeitig optimiert. In einem weiteren Beitrag dieser Arbeit werden verschiedene Suchmethoden für neuronale Architekturen, die den von uns vorgeschlagenen Suchraum nutzen, optimiert und mit von Experten entworfenen Architekturen verglichen. Beide Arbeiten deuten darauf hin, dass automatische Architekturentwurfsmethoden bessere Modelle finden als von Experten entworfene Modelle.
Weitere Beiträge dieser Arbeit befassen sich mit Methoden zur Quantifizierung von Unsicherheiten, die auf genomische Daten angewendet werden. Anwendungen im Bereich Gesundheit und Biologie sind oft sicherheitskritisch, weshalb die Zuverlässigkeit von Deep-Learning-Modellen untersucht und verbessert werden sollte. Zu diesem Zweck werden verschiedene Methoden zur Quantifizierung von Unsicherheiten bei der Vorhersage regulatorischer Aktivitäten untersucht und eine neue Deep-Learning-Methode zur Verbesserung der Kalibrierung von Vorhersagen im Kontext sequenzieller Genomdaten erforscht.
Schließlich wird ein Beitrag zur Entwicklung einer benutzerfreundlichen Software geleistet, die verschiedene Formate genomischer Daten verarbeiten kann, einschließlich der Integration wichtiger Teile mehrerer in dieser Arbeit vorgestellter Methoden in diese Software.

Contents

Contributing Articles

Chapter 4.1 Gündüz HA, Binder M, To XY, Mreches R, Bischl B, McHardy AC, Münch PC, Rezaei M (2023). "A self-supervised deep learning method for data-efficient training in genomics." *Communications Biology*, **6**(1), 928. doi:10.1038/s42003-023-05310-2

Chapter 5.1 Gündüz HA, Mreches R, Moosbauer J, Robertson G, To XY, Franzosa EA, Huttenhower C, Rezaei M, McHardy AC, Bischl B, Münch PC, Binder M (2024). "Optimized model architectures for deep learning on genomic data." *Communications Biology*, **7**(1), 516. doi:10.1038/s42003-024-06161-1

Chapter 5.2 Scheppach A, Gündüz HA, Dorigatti E, Münch PC, McHardy AC, Bischl B, Rezaei M, Binder M (2023). "Neural Architecture Search for Genomic Sequence Data." In *2023 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, pp. 1–10. doi:10.1109/CIBCB56990.2023.10264875

Chapter 6.1 Gündüz HA, Giri S, Binder M, Bischl B, Rezaei M (2023). "Uncertainty Quantification for Deep Learning Models Predicting the Regulatory Activity of DNA Sequences." In *2023 International Conference on Machine Learning and Applications (ICMLA)*, pp. 566–573. doi:10.1109/ICMLA58977.2023.00084

Chapter 6.2 Turkoglu MO, Becker A, Gündüz HA, Rezaei M, Bischl B, Daudt RC, D' Aronco S, Wegner J, Schindler K (2022). "FiLM-Ensemble: Probabilistic Deep Learning via Feature-wise Linear Modulation." In S Koyejo, S Mohamed, A Agarwal, D Belgrave, K Cho, A Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 22229–22242. Curran Associates, Inc. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/8bd31288ad8e9a31d519fdeede7ee47d-Paper-Conference.pdf

Chapter 6.3 Vahidi A, Wimmer L, Gündüz HA, Bischl B, Hüllermeier E, Rezaei M (2024). "Diversified Ensemble of Independent Sub-networks for Robust Self-supervised Representation Learning." In A Bifet, J Davis, T Krilavičius, M Kull, E Ntoutsi, I Žliobaitė (eds.), *Machine Learning and Knowledge Discovery in Databases. Research Track. ECML PKDD 2024*, pp. 38–55. Springer Nature Switzerland, Cham. doi:10.1007/978-3-031-70341-6_3

Chapter 7.1 Mreches R, To XY, Gündüz HA, Moosbauer J, Klawitter S, Deng ZL, Robertson G, Rezaei M, Asgari E, Franzosa EA, Huttenhower C, Bischl B, McHardy AC, Binder M, Münch PC (2024). "GenomeNet: A platform for deep learning on (meta)genomic sequences."

List of Abbreviations

A adenine

ACC accuracy

AMBER automated modeling for biological evidence-based research

AUC area under curve

AUROC area under receiver operating characteristic

AutoML automated machine learning

BANANAS bayesian optimization with neural architectures for neural architecture search

BERT bidirectional encoder representations from transformers

BONAS bayesian optimization neural architecture search

C cytosine

CNN convolutional neural networks

CPC contrastive predictive coding

CV computer vision

CWP-DARTS continuous weight sharing progressive differentiable architecture search

DAG directed acyclic graph

DARTS differentiable architecture search

DeepSEA deep learning–based sequence analyzer

DL deep learning

DNA deoxyribonucleic acid

ECE expected calibration error

EDP-DARTS edge discarding progressive differentiable architecture search

ENAS efficient neural architecture search

FiLM feature-wise linear modulation

FN false negative

FP false positive

G guanine

GAP global average pooling

GMP global max pooling

GPU graphics processing unit

GRU gated recurrent unit

HP hyperparameter

HPO hyperparameter optimization

ID in-distribution

LM language model

LLM large language model

LSTM long short-term memory

M million

MAE mean absolute error

MBO model-based optimization

MC-Dropout monte carlo dropout

MCMC markov chain monte carlo

MIMO multi input multi output

ML machine learning

MSE mean square error

NAS neural architecture search

NGS next generation sequencing

NLL negative log-likelihood

NLP natural language processing

OOD out-of-distribution

ORF open reading frames

P-DARTS progressive differentiable architecture search

PPR-Meta phage and plasmid recognizer for metagenomes

PR-AUC area under precision recall curve

RC reverse-complement

ReLU rectified linear unit

RMSE root mean square error

RNA ribonucleic acid

RNN recurrent neural networks

ROC receiver operating characteristic

RS random search

SCE static calibration error

SGD stochastic gradient descent

SH successive halving

SMBO surrogate model-based optimization

SNP single nucleotide polymorphism

SSL self-supervised learning

T thymine

TACE threshold adaptive calibration error

TN true negative

TP true positive

U uracil

UPC upper confidence bound

UQ uncertainty quantification

Part I

Introduction, Background, and Conclusion

1 Introduction

1.1 Introduction

As an introductory section, the motivation for the thesis is presented, along with the type of data that the contributions of this thesis will deal with, namely nucleic acids, and the historical developments that lead to the abundance of this type of data. Advances in sequencing technology, particularly the reduction in the cost of the technology, have paved the way for more effective use of deep learning algorithms, which can require large amounts of data to learn from.

Nucleic acids are the primary carriers of genetic information in all living organisms, including viruses. They regulate vital functions of organisms such as development, function, and reproduction. Two major classes of nucleic acids are DNA (deoxyribonucleic acid) sequences and RNA (ribonucleic acid) sequences. The building blocks of nucleic acids are called nucleotides. A nucleotide contains a five-carbon sugar, a phosphate group, and a nitrogen-containing nucleobase. The sugar present is deoxyribose in DNA and ribose in RNA, which is also part of the names of these nucleic acids. There are four types of nucleobases in DNA and RNA. The four nucleobases in DNA are adenine (A), cytosine (C), guanine (G), and thymine (T). In RNA, there is uracil (U) instead of thymine. By providing diversity, nucleobases (bases) are the source of information in nucleic acids. Since nucleobases are the main source of information in nucleic acids, DNA or RNA sequencing refers to determining the order of these bases, which can then be stored and processed.

Watson and Crick (1953) proposed the double helix structure of DNA. Holley *et al.* (1965) determined the first whole nucleic acid, which was alanine transfer RNA and was only 77 nucleotides long. The first breakthrough in DNA sequencing was proposed by Sanger *et al.* (1977b). The method, also known as Sanger sequencing, was used to sequence the first DNA, a bacteriophage with 5,386 base pairs (Sanger *et al.*, 1977a). Sanger sequencing was the first widely used sequencing technique and has been the most widely used sequencing technique for several decades. Heather and Chain (2016) describe the proposal of the method as the birth of $first - generationsequencing$. A major milestone in DNA sequencing was the Human Genome Project between 1990 and 2003, a large and well-organized project aimed at sequencing the entire human genome. By the end of the project, 92% of the human genome had been sequenced. Another DNA sequencing technique, called pyrosequencing, was proposed by Ronaghi *et al.* (1996) and formed the basis of the first so-called Next Generation Sequencing (NGS). NGS is a high-throughput, high-speed sequencing technique that allows hundreds of millions of DNA or RNA fragments to be sequenced simultaneously. In addition to being much faster, sequencing technology has also become much more affordable over the past few decades.

Large amounts of genomic data needed by machine learning algorithms are already available thanks to NGS. Machine learning methods have also advanced significantly in many different fields such as computer vision and natural language processing, as discussed in more detail in chapter 2.

Despite these advances in machine learning (ML) and the large amount of genomic data from which ML algorithms can learn, the application of ML to genomic data has been relatively slow. This thesis aims to bridge the gap between these fields by contributing to several areas of machine learning applied to genomics data, such as self-supervised learning, automated model design, and uncertainty quantification. These areas of machine learning are particularly relevant to genomics. First, self-supervised learning aims to reduce the need for human-annotated labels, which can be expensive and error-prone in genomics. Second, automated model design and neural architecture discovery aim to find successful models in an automated way for a given dataset and task, which is a need in genomics since the models designed for genomics have not yet been successful enough to become standard models for many different tasks in the field. Third, uncertainty quantification and calibration are important in genomics because many applications in this field can be safety-critical, and in such cases, uncertain model decisions must be avoided and human experts should make decisions. Finally, contributions to software for genomics are made so that bioinformaticians and researchers can easily use machine learning models and several contributions in this thesis.

1.2 Outline

This thesis is divided into two parts. The first part introduces the basics of deep learning, the basics of several research areas of deep learning relevant to this thesis, and some concluding remarks. The basics of deep learning are presented by approaching the field from three perspectives: data, model, and evaluation. The data perspective looks for answers on how to use existing data so that models can make accurate predictions. The model perspective answers how a machine can find a function f that infers information from input data. Finally, the evaluation perspective addresses how to determine the performance of these functions (models).

The machine learning research areas relevant to this thesis, self-supervised learning, automated model design (including neural architecture search), and uncertainty quantification, are introduced in sections 2.2, 2.3, and 2.4, respectively. Section 2.5 briefly introduces deep learning methods in these areas applied to genomic data, along with machine learning-related software developed for this data type. Chapter 3 provides a final discussion of the contributions of this thesis and possible future research directions.

The second part of the thesis contains the articles and their appendices, if any. The chapters are organized by research area so that the articles are as close as possible to each other. In particular, contributions for genomics are included in these specific machine learning areas: self-supervised learning (chapter 4), automated design of deep learning models (chapter 5), uncertainty and calibration (chapter 6), and software development (chapter 7).

2 Methodological Background

2.1 Introduction to Deep Learning

This section introduces the main concepts of deep learning from three perspectives: data, model, and evaluation.

Data Perspective

Deep learning seeks data-driven answers to the questions it seeks to answer, but how can existing data be used to make the models (more) accurate in their predictions? The answer to this question may depend on several aspects, such as the amount of data available or the modality of the data. However, a common way to answer this question is closely related to whether or not the data is labeled. The data can be labeled, unlabeled, or partially labeled. Furthermore, methods focus on how to label the unlabeled data automatically and without human annotation, from which the trained models can learn further. All these cases have different learning methods, and each case will be studied in this part of the thesis.

i. Supervised learning: Supervised learning deals with situations where the data is labeled, i.e. each data sample x_i is paired with a label y_i. The goal is to find a function f that can infer these labels based on the inputs. For example, the data may consist of images of cats and dogs, and the labels of each image as to whether it is a cat or a dog. Then the image of a sample i is x_i, its annotation is y_i, and the goal is to have a classifier f that correctly classifies y_i from x_i such that $y_i = f(x_i)$. Such tasks, where an input sample is assigned to one of several options (classes), are called $classification$ tasks (LeCun *et al.*, 1998). The annotation can also be a continuous variable, such as the weight of cats and dogs, and if such annotations are to be predicted, the task is called a $regression$ task (Sykes, 1993).

ii. Unsupervised learning: Unsupervised learning deals with cases where there are no annotations. Typical tasks for unsupervised learning are clustering, association, and dimensionality reduction. The clustering algorithms aim to group the uncategorized data samples by finding similarities (Xu and Tian, 2015). The goal of association algorithms is to find relationships between different variables in the data set, such as whether two products are often purchased together (Agrawal *et al.*, 1993). Dimensionality reduction reduces the number of features (dimensions) that a data input has, while preserving as much data integrity as possible (Maćkiewicz and Ratajczak, 1993; Van der Maaten and Hinton, 2008). This can be used to visualize data with many inputs by reducing the dimension size to a visualizable value, such as two dimensions, or it can be used to reduce the irrelevant or random features in the data.

iii. Self-supervised learning: Self-supervised methods can be considered as a subset of unsupervised models since they do not require human labeling, similar to (other) unsupervised learning

methods. Nevertheless, these methods propose ways to create labels from the unlabeled input data itself (Devlin *et al.*, 2018; Chen *et al.*, 2020). The models are then trained using these labels, similar to supervised learning methods. A simple example is next token prediction, where, for example in NLP, the word that comes after a text can be predicted. In this example, the data may not have any human annotations, e.g. no human had to read some or all of the existing texts in the data and had to come up with labels based on the desired task. However, the labels are generated from the originally unlabeled data. After self-supervised pre-training, models can transfer knowledge learned from unlabeled data to supervised *downstream* tasks. In this way, self-supervised methods use unlabeled data to improve the performance of supervised learning tasks. A typical use case for self-supervised learning is when a lot of unlabeled data is available. For example, the vast availability of natural language data (text data), computer vision data (videos and images), and audio data make it possible to use self-supervised methods to improve state-of-the-art deep learning models and reduce the need for human annotation (Floridi and Chiriatti, 2020; Henaff, 2020; Liu *et al.*, 2022).

iv. Semi-supervised learning: Similar to self-supervised learning, semi-supervised learning methods aim to reduce the amount of labeled data. However, semi-supervised learning involves learning from both labeled and unlabeled data simultaneously to improve model performance (Berthelot *et al.*, 2019).

v. Reinforcement learning: Interactive learning in machine learning is called reinforcement learning. Reinforcement learning methods are based on the interaction between a controller and an environment (Mnih, 2013). The controller decides what action to take in the next step. The environment then feeds back the new state and the reward to the controller. Next, the controller can decide the next action based on the state and reward coming from the environment. This process is repeated until the learning process is stopped.

Model Perspective

How does a machine find a function f that can infer information from input data? For example, how does a machine find a function that can successfully infer the label y_i from an input sample x_i in supervised learning? A short answer is that a model is optimized using the data. To understand what is being optimized, it is necessary to introduce the building blocks of deep learning: layers. Next, architectures formed by combining these building blocks in different orders will be discussed. Finally, the methods for optimizing the weights and hyperparameters of these architectures/models are discussed.

Model - From Layers to Architectures: The building blocks of a deep learning model are called layers. A layer is a node that takes information from previous layers, modifies it, and passes it on to the next layer. Terminologically, the layer that takes x_i is called the input layer, the layer that predicts y_i is called the output layer, and the remaining layers between these two layers are called hidden layers. How these layers modify the received information is determined by the type of the layer. For example, a $fully$ connected layer (also called a *linear* or *dense* layer) performs the following operation:

$$y = \sigma\left(Wx + b\right) \tag{2.1}$$

where σ is the activation function, which is a nonlinear function applied to the vector element-wise, W is a matrix of weights connecting x to y, which are the input and output vectors (i.e. representation vectors) respectively, and b is the bias vector. It can be shown that without an activation function to introduce non-linearity, N linear layers stacked on top of each other are equivalent to a linear layer with different weights. There are several commonly used activation functions (Maas *et al.*, 2013; Bridle, 1990; Han and Moraga, 1995; He *et al.*, 2015). A simple and common activation function is Rectified Linear Unit (ReLU) (Nair and Hinton, 2010):

$$ReLU(x) = \begin{cases} x, & \text{if } x \geq 0, \\ 0, & \text{if } x < 0. \end{cases} \tag{2.2}$$

Another common layer type is *convolutional* layers. Convolutional layers are often used in computer vision tasks, among others. Convolutional layers consist of filters (also called kernels) that iterate over different regions of the input data. For example, for a filter of size 3×3 and an image of size 32×32, different regions of size 3×3 are multiplied by the filter element-wise, and then the output of this multiplication is summed and fed into an activation layer to produce the output. Some advantages of convolutional networks are local sparse connections, parameter sharing, and translation equivariance. These allow convolutional networks to search for complex features locally rather than globally, to reduce memory requirements, to learn efficiently, and to detect features regardless of their position in the image.

Recurrent layers are another popular layer type to process sequential data. These layers process data sequentially, feeding the input to the layer bit by bit, from one end of the sequence to the other. A simple (also called *vanilla*) recurrent layer computes a hidden vector that embeds the information of the previously processed portion of the input data. This vector and the next part of the input are used to compute the hidden vector that will be used in the next step and, if desired, an output vector. Since the hidden vector received from the previous step is needed to perform the computations, the process must be sequential, and the computations are thus usually slow because they cannot be parallelized. They are also prone to the vanishing gradient problem, which makes learning long-range effects difficult. More popular recurrent layers than the vanilla version, such as LSTM (Hochreiter, 1997) and GRU (Cho, 2014), address this problem by introducing additional gates.

Attention layers determine the importance of each component/position relative to the others. They gained popularity while mostly being used as an additional component to recurrent layers like LSTM (Parikh *et al.*, 2016). Vaswani et al. proposed the first transformer model in their influential paper entitled "Attention is all you need" (Vaswani, 2017). As its name partly suggests, the introduced transformer model consisted of $self-attention$ layers and did not use any recurrent layers, yet the model outperformed the state-of-the-art techniques.

Deep learning architectures are built by stacking layers on top of each other, which means that the output of one layer is often fed into another (subsequent) layer as input. Similarly, the output of that layer is fed into another layer as an output. For example, an early convolutional network LeNet consists of two blocks of a convolutional and a pooling layer, followed by three fully connected layers (LeCun *et al.*, 1998). AlexNet contained five convolutional layers followed by three fully connected layers (Krizhevsky *et al.*, 2012). ResNet proposed residual blocks to train a substantially larger number of layers end-to-end and successfully trained a state-of-the-art model

with 152 layers (He *et al.*, 2016). Besides stacking fully connected layers on top of convolutional layers, it is also possible to have other successful architectures with other combinations, such as using recurrent layers between convolutional and fully connected layers (Gündüz *et al.*, 2024).

Optimization of the Models: The weights of deep learning models are optimized for specific datasets and tasks. Here the methodology for optimizing hyperparameters and weights is discussed.

i. Weights Optimization: In machine learning, the model performance is estimated and optimized by minimizing the empirical risk R_{emp} computed by averaging the loss function over the training set. Thus, in a standard supervised learning setting, a machine learning model f with learnable weights θ aims to minimize a defined loss function L over the training data consisting of input-label pairs (x^i, y^i) indexed by i:

$$R_{emp} = \frac{1}{n} \sum_{i=1}^{n} L(y^{(i)}, f(x^{(i)}|\theta)). \tag{2.3}$$

The loss function is typically defined to be minimized as $f(x^{(i)}|\theta)$ approximates $y^{(i)}$ on average over the training data. For more information on different loss functions, see the subsection 2.1: "Evaluation Perspective".

A simple method for determining θ that minimizes empirical risk is called gradient descent. Gradient descent updates the weights towards the direction that the risk is minimized at maximum for an infinitesimal step. This direction is calculated by computing the negative of the empirical risk's gradient. The direction is multiplied by a scalar value called the learning rate (α), which adjusts the size of the taken step. While this iteration is repeated several times, typically until the empirical risk converges, an iteration at time t is mathematically given as follows

$$\theta_{t+1} = \theta_t - \alpha \nabla R_{emp}(\theta) \tag{2.4}$$

Since computing the gradient over the entire training set is computationally expensive, it is typical to use small stochastic subsets instead of the entire set. This method is called stochastic gradient descent. Stochastic gradient descent can be further accelerated by using a method called momentum, which accumulates the moving average of the computed gradient descents. (Polyak, 1964; Sutskever *et al.*, 2013; Goodfellow, 2016).

ii. Hyperparameter Optimization: Hyperparameters are typically set before weight optimization to define the underlying model and training procedure. A model consists of hyperparameters that affect the overall layout of the architecture, such as the number of layers or the number of neurons in a layer. In addition, some hyperparameters are related to the optimization procedure, such as the learning rate. Hyperparameter optimization is typically done by selecting some hyperparameters and training the weights of the model given those hyperparameters. This process is repeated as long as the computational budget allows. The simplest hyperparameter optimization method is probably random search. Here, the user defines a search space, which is a list of hyperparameters to optimize and the range of values these hyperparameters can take. Then, models are trained with randomly and independently sampled sets of hyperparameters until a predefined computational budget is reached. The selected set of hyperparameters are those that belong to the model

with the highest performance. A more detailed discussion of hyperparameter optimization can be found in the section 2.3: “Automated Model Design”.

Evaluation Perspective

Given that one or more models are optimized on a dataset, how do you determine the performance of these models and which one to use for a particular use case scenario? Evaluation metrics provide insight into model performance and make it easier to compare different models.

Some evaluation metrics are differentiable and can be used to train model weights using modern optimization methods based on gradient descent. Such metrics include mean square error (MSE) and mean absolute error (MAE) for regression tasks, and negative log-likelihood (NLL) loss for classification tasks.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2.5}$$

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{2.6}$$

$$NLL = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij} log(\hat{y}_{ij}) \tag{2.7}$$

where $\hat{y}_i = f(x_i)$ is the prediction, y_i is the ground truth (label) value or vector, and j in $\hat{y}_{ij}$ and y_{ij} indicates the indice value of the vector and corresponds to the predicted probability or ground truth probability that data point i belongs to a particular class indexed by j. However, it should be noted that the training (weight optimization) of the model is done on the training set, and a more realistic performance indicator of the model on unseen data is to evaluate a metric on the test data, as the model may overfit the training set.

Some evaluation metrics are not differentiable and cannot be used for the optimization. Nevertheless, they offer valuable insight into the performance of the model. A popular and simple metric is accuracy for classification tasks, evaluated by the number of correct classifications divided by the number of all classifications. For binary classification tasks, labels are typically called positive and negative labels, where the positive label often refers to an anomaly such as an illness or fraudulent action. With two possible ground truth labels and two possible predicted labels (positive or negative), the four possible combinations of ground truth and prediction pair are called true positive (TP), true negative (TN), false positive (FP), and false negative (FN). Here, the first word designates if the ground truth and the predicted label match, and the second word refers to the predicted label. Therefore, the accuracy for the binary classification can be given as (TP+TN)/(TP+TN+FP+FN). Sensitivity and recall are given as TP/(TP+FN). Specificity is given as TN/(TN+FP) and precision is defined as TP/(TP+FP). Precision-recall and sensitivity-specificity are typically used as pairs and deliver more information than accuracy, especially when the data is imbalanced. F_1 score is computed by the harmonic mean of precision and recall. However, all these metrics require a classification to either the positive or the negative class, and therefore threshold selection is needed to divide data samples into two groups based on

the numeric outcome of the machine learning model. The area under the receiver characteristic (ROC) curve (AUC) is obtained by plotting *sensitivity* against $1 - specificity$ for all threshold values and evaluating the area under the plotted curve (Rainio *et al.*, 2024). Thus, this metric is threshold-independent, i.e. a specific threshold is not needed to be chosen to evaluate this metric. The precision-recall AUC is also similarly threshold-independent.

For multi-class classification, TP, TN, FP, and FN can be derived to form a confusion matrix similar to the binary classification tasks uniquely for each class (Rainio *et al.*, 2024). To compute *macro-averaged* metrics, metrics can be evaluated for each class and then the evaluated results for each class are averaged. For example, macro-averaged recall can be computed by averaging the recall values for each class. Thus, the macro-averaged recall can also be interpreted as class-balanced accuracy, since the recall value for a class is computed by dividing the true classifications (TP) by the number of samples (TP+FN) from that class, being equivalent to the accuracy for that class, and then the values computed for each class are averaged. *Micro-averaged* metrics can be evaluated by summing up the number of observations over each class when the elements of the confusion matrix are formed.

For regression tasks, Pearson's correlation coefficient (r) can be used as an evaluation metric that measures the correlation between the ground truth and the predicted values.

In addition, the uncertainty of the predictions can also be quantified by various techniques, which can be an important component of the performance, especially for safety-critical areas. Recently introduced NLL, ECE (Naeini *et al.*, 2015) to be introduced in a following subchapter, static calibration error (SCE) (Nixon *et al.*, 2019), Brier score (Brier, 1950), and threshold adaptive calibration error (TACE) (Nixon *et al.*, 2019) are some metrics to give insights regarding the uncertainty of models.

2.2 Self-Supervised Learning

Motivation

Annotating data and thus creating manually labeled datasets relying on human labeling is often expensive and bound to errors. This is especially the case for genomics or similar data types such as health data, as labeling for such tasks depends heavily on human expertise or spending on laboratory equipment thus making it even more expensive than average labeling cases. Additionally, it is often in these fields that an annotation by humans is wrong, and thus harmful for the machine learning models that are trained on this data. Therefore, self-supervised learning methods are proposed to increase the overall performance of deep learning models and to reduce the costs by making use of unlabeled data and thus reducing the need for larger amounts of clean and correct data annotated by humans.

General Structure

Unlike supervised methods, self-supervised learning techniques use unlabeled data to learn meaningful representations that contain information about the properties of the data. A typical use

case for self-supervised learning consists of two parts (Oord *et al.*, 2018; Henaff, 2020; Chen *et al.*, 2020; Devlin *et al.*, 2018):

i. Self-supervised training: A model is trained on unlabeled data, which is often very large. In self-supervised learning methods, labels are automatically generated from unlabeled data based on defined tasks. These tasks are called pretext tasks. Then, similar to supervised training, the model is trained using these generated ground truth labels from the pretext task.

ii. Evaluation on downstream tasks: A machine learning model consisting of one or more layers is initialized on top of the pre-trained model, i.e., the pre-trained model typically feeds its output to this model. This model, possibly together with the pre-trained model, is then fine-tuned in a typical supervised fashion on labeled data, which is often a much smaller dataset, to perform the intended task of the model. This fine-tuned model typically performs much better than the case where the model is not pre-trained or equally where the model weights are randomly initialized.

In the following subsections, different methods for self-supervised training and the evaluation protocols for training on downstream tasks will be discussed.

Types of Self-Supervised Learning

Self-supervised learning methods can be roughly divided into two categories, which will be discussed in this subsection.

Generative Self-Supervised Learning: While modern self-supervised learning methods can use more complex structures to achieve superior performance, the basic idea of generative self-supervised training is to predict some tokens of text based on other tokens. For NLP tasks, these tokens are typically words. In this way, labels that can be used for supervised training are automatically generated from the "self" of the unlabeled data, and no further human annotation is required.

A generative self-supervised learning method is the auto-regressive model, which is trained by iteratively predicting the upcoming data based on the previous data. For example, the next word can be predicted based on the previous words, and the following word can also be predicted based on the predicted word in addition to the previous words (Floridi and Chiriatti, 2020; Achiam *et al.*, 2023). Similarly, based on some pixels of the images, the rest of the image can be generated pixel by pixel (Van den Oord *et al.*, 2016; Van Den Oord *et al.*, 2016) or given the initial part of an audio sample, the rest can be generated (Van Den Oord *et al.*, 2016). Trained in this way, these models are generative models by their nature, but they can also be used for other tasks if they are tuned for those tasks.

Another generative self-supervised learning method is the auto-encoding model, where a complete input is fed into the model in a distorted way and the model tries to predict the original input. For example, the popular BERT (Devlin *et al.*, 2018) method randomly modifies some of the existing tokens to be a special mask token and tries to predict the original tokens while training the model using the cross-entropy loss, which is a typical loss used for supervised classification tasks.

Contrastive Self-Supervised Learning: While effective for NLP tasks, non-contrastive approaches have not been as successful in several deep learning application areas, such as computer vision or audio processing. For example, models trained by predicting missing pixels for image data, or similarly by predicting missing samples for audio, performed suboptimally and were later outperformed by contrastive learning in many domains and cases (Oord *et al.*, 2018). A contrastive self-supervised learning method typically minimizes the distance between the learned representations of two distorted copies of input data, while maximizing the distance between the representations of different input samples. For example, given two distorted copies of an image of a cat and many other images, such as an image of a mouse, the model learns that the copies of the cat image have more in common, such as similar tails or paws, than a cat and a mouse, by learning to embed the features of the images into the representations. In practice, these distorted copies are extensions of the original image.

A contrastive self-supervised method uses a contrastive loss to minimize the distance between the matched representations. An example is the Contrastive Predictive Coding (CPC) method introduced by Oord *et al.* (2018), which used contrastive loss to train self-supervised deep learning models and demonstrated the efficiency of contrastive loss on audio, images, natural language data, and reinforcement learning tasks.

In general, the anchor, positive, and negative samples x^a, x^+, and x^- are encoded by an encoder network f_e to compute their representations $z^a = f_e(x^a)$, $z^+ = f_e(x^+)$, and $z^- = f_e(x^-)$ in a contrastive loss given by

$$L_{contrastive} = -\mathbb{E}\left[\log \frac{f_s(z^a, z^+)}{f_s(z^a, z^+) + \sum_{j=1}^{N} f_s(z^a, z_j^-)}\right] \tag{2.8}$$

where f_s is a similarity function. It is also possible to split a data sample into multiple patches, which is particularly useful when the data type is, for example, audio, text, or sequential genomics data (Oord *et al.*, 2018; Gündüz *et al.*, 2023). The representations of these patches can be computed similarly by feeding them into f_e. The anchor sample can consist of multiple patches, and the representation of the anchor samples can be computed by feeding in the representations of the patches that make up the anchor sample. Similarly, the positive and negative samples can be a group of patches, but it is more common for these samples to be defined as a single patch. The similarity function can also be represented by a neural network or a linear layer.

Evaluation Protocols for Training on Down-stream Tasks

After self-supervised pre-training, the learned representation of this self-supervised model is trained or tested on supervised, usually smaller, datasets. Thus, training on downstream tasks is a supervised training scheme, where the main difference from typical supervised training is the use of pre-trained weights as initial weights instead of random initialization. A self-supervised learning model typically achieves much higher performance on small labeled datasets. These labeled datasets are often called *downstreamtasks* or *downstreamdatasets* and can either be collected or imitated.

i. Transfer learning: The self-supervised model is trained on labeled datasets, which are usually of small size. These datasets are assumed to have a different data distribution than the dataset

used for the self-supervised training of the model. For example, Chen *et al.* (2020) uses 12 popular computer vision datasets for transfer learning.

ii. Semi-supervised learning / Data-scarce evaluation: In this setting, a limited portion of the dataset used during the self-supervised pre-training is used as the downstream dataset. For example, 1% and 10% of the dataset used for self-supervision is used as the downstream datasets, along with the labels (Chen *et al.*, 2020; Gündüz *et al.*, 2023). By providing a considerably large dataset, with only a limited portion of that dataset available with labels, data scarcity is mimicked.

iii. Linear evaluation / linear classification: While other evaluation protocols do not impose any constraint on the model added on top of the self-supervised model, in linear classification protocol, the model added on top of the self-supervised model is a linear classifier. In addition, the model trained in the self-supervised pre-training is not updated during the downstream training (Chen *et al.*, 2020; Bardes *et al.*, 2021; Goyal *et al.*, 2021; Gündüz *et al.*, 2023). This implies that the representations evaluated for a data sample remain unchanged after training on the downstream task. With these measures, this protocol ensures that the effect of training on the downstream task on the representations is zero, and the superiority of the learned representations is tested in a difficult setting limited to a linear layer on top of the representations, which implies testing the superiority of self-supervised methods.

2.3 Automated Model Design

Automated machine learning (AutoML) describes the automation of the machine learning pipeline. A standard AutoML workflow includes steps such as data preprocessing, feature extraction, model generation, and model evaluation (He *et al.*, 2021). Neural architecture search (NAS), the focus of this thesis, is a subset of AutoML that aims at the automation of neural network design. Therefore, NAS is particularly concerned with the model generation and evaluation phases of AutoML. The first of these, the automated generation of machine learning models, contains two essential parts: search space and optimization methods. The search space refers to the possible architectures to be searched and optimized. The optimization methods or search strategy can be divided into hyperparameter optimization, including learning rate optimization, and architecture optimization, including optimization of parameters such as filter size of convolutional layers. In line with these points, Elsken *et al.* (2019) divides a typical NAS framework into three stages: search space, search strategy, and performance evaluation strategy.

Search Space

The search space defines the potential models that a NAS algorithm, or more specifically the algorithm's optimization methods, can discover. Designing a good search space is therefore a challenging but crucial problem. A good search space should include a wide variety of model architectures to reduce human bias (He *et al.*, 2021), while larger search spaces with fewer constraints are usually more computationally expensive.

A simple search space can be a chain-structured neural network, where each hidden layer in the architecture receives its input from only one (previous) layer and feeds its output to only one

(following) layer. On the other hand, a more complex search space can be obtained by removing the chain structure constraint and thus including additional possibilities such as multi-branching or skip connections. He *et al.* (2021) introduces both such search spaces under the umbrella term fully structured search space. The search space has to be parameterized in order to be searchable. These parameters can be, for example, the number of layers, the type of operations in the existing layers, or the hyperparameters of the operations (Elsken *et al.*, 2019; Gündüz *et al.*, 2024).

Deep architectures based on human expert designs often rely on repeating blocks (Szegedy *et al.*, 2016; He *et al.*, 2016; Huang *et al.*, 2017). Inspired by this, Zoph *et al.* (2018) and Zhong *et al.* (2018) proposed cell-based search spaces so that optimized cells can be stacked to form the final architecture. For example, Zoph *et al.* (2018) proposed optimizing two different repeating cells as part of the architecture: a normal cell that preserves the input dimensionality, and a reduction cell to reduce the spatial dimension. The cell-based approaches reduce the size of the search space, thus reducing the computational burden and speeding up the optimization. While cell-based methods reduce the complexity of the search space by reducing the number of possible architectures, they suffer from a gap between the models trained and evaluated during the search phase and the models selected as the final architecture and evaluated during the evaluation phase. For example, DARTS consists of eight cells in the search phase and 20 cells during the evaluation phase (Liu *et al.*, 2018b). While the preference for a larger architecture for the evaluation phase is based on the fact that they tend to perform better than the smaller ones and the evaluation of many architectures during the search phase is computationally expensive, there is a chance that the found architecture could be suboptimal or the larger architecture could even perform worse than the one discovered in the search phase. This has been addressed in Chen *et al.* (2021) by gradually increasing the number of cells during the search phase.

Architecture Optimization

Architectures within the defined search space should be efficiently evaluated to discover the near-optimal architectures within the defined search space with many model possibilities. Therefore, several optimization methods have been proposed. Some important architecture optimization methods include evolutionary algorithms, reinforcement learning methods, differentiable models optimized with gradient descent, and surrogate model-based optimization methods (He *et al.*, 2021).

Evolutionary algorithms are designed to be similar to biological evolution. A part of the generated models are selected based on their performance and they generate offspring as pairs by forwarding a fraction of their material (Real *et al.*, 2017; Elsken *et al.*, 2018; Real *et al.*, 2019; Irwin-Harris *et al.*, 2019). Copied genetic material can be randomly modified, similar to mutations in biological processes and some underperforming models can be removed from generated networks. These steps are repeated several times. Evolutionary algorithms are well-established methods with robust performance, but they are usually computationally expensive (He *et al.*, 2021).

As explained earlier, reinforcement learning is interactive learning between a controller and an environment. In the context of NAS algorithms, the controller of the reinforcement learning algorithm can be an ML model such as an RNN that decides the next architecture to be evaluated, while the environment of the reinforcement algorithm is the training and evaluation of this selected model. The performance of this architecture is then fed back to the controller to decide the next

architecture to be evaluated (Zoph and Le, 2016; Baker *et al.*, 2016; Pham *et al.*, 2018; Zoph *et al.*, 2018).

Gradient descent-based architecture search algorithms emerged when Liu *et al.* (2018b) defined a differentiable search space by assigning weights to candidate operations and using a softmax function on these weights. In particular, these assigned weights are also optimized to decide useful operations to keep between nodes and useless ones to discard.

Surrogate Model-Based Optimization (SMBO) uses a surrogate model that attempts to predict the most promising architecture based on previous evaluation results. The surrogate models used can be based on Bayesian optimization, such as Gaussian processes. (Wistuba, 2017; Kandasamy *et al.*, 2018) or neural networks such as LSTM (Liu *et al.*, 2018a) or multi-layer perceptron (Luo *et al.*, 2018).

Model Evaluation

Although it would be reliable to evaluate models on the full test set after training them to convergence on the full training set, this is computationally expensive, especially considering the potential number of architectures that can be evaluated. In practice, model evaluation can even take tens of thousands of GPU days, representing a significant amount of computational resources (Zoph and Le, 2016). A GPU (graphics processing unit) is a powerful computational resource used to train and run deep learning models. A GPU day refers to the computing capacity of a GPU used for an entire day. Due to such high computational requirements, several works have been done to reduce the evaluation time.

He *et al.* (2021) divides model evaluation methods into four groups: low fidelity, weight sharing, surrogate, and early stopping. Examples of low fidelity cases include when a smaller subset of the dataset is used for training (Klein *et al.*, 2017), the resolution of the data is lowered (Chrabaszcz *et al.*, 2017), or when a similar but smaller model is used (Zoph *et al.*, 2018). Weight sharing includes transferring the weights trained for other tasks (Wong *et al.*, 2018) or other architectures (Chen *et al.*, 2015), or can be among child networks (Pham *et al.*, 2018). Surrogate models can be used to control the search method (Liu *et al.*, 2018a). Similar to its use case to prevent overfitting, early stopping refers to stopping training early, but in neural architecture search it is used instead to discard unpromising models after a limited training time (Klein *et al.*, 2022).

2.4 Uncertainty Quantification

Uncertainty quantification and calibrated models are critical, especially in safety-critical areas such as automated healthcare. In such applications, model prediction performance at a desired level may not be sufficient to use the model because the risk of wrong decisions may be too high. In such cases, the use of the model may only be possible if the model can accurately detect high-risk situations, such as when there is a high risk of a wrong model decision for a specific input sample so that human expert intervention in the decision process is possible and thus significantly risky situations are avoided. In other words, even in safety-critical situations, a model may need to accurately determine how likely its predictions are to be right or wrong.

Sources of Uncertainty

Sources of uncertainty are commonly divided into two main categories called aleatoric uncertainty (also called data uncertainty) and epistemic uncertainty (also called model uncertainty) (Hüllermeier and Waegeman, 2021).

i. Aleatoric uncertainty: Aleatoric (from *alea*, which means *dice* in Latin) uncertainty refers to the variability of outcomes due to inherent randomness in the experiments. Some good examples of such random processes are fair coin flipping or dice rolling. In these situations, the uncertainty of the outcome is due to the randomness of the experimental process. For such cases, even with collecting an almost infinite amount of samples, it would not be possible to predict the outcome of an observation with almost complete certainty, e.g. it is not possible to know if a flipped fair coin would be head or tail with certainty, no matter how many times the experiment is previously repeated. For this reason, it is also known as irreducible uncertainty, as the uncertainty in the experiment outcome is not due to a limited number of observations, but is due to the inherent randomness.

For machine learning systems, noise and error in measurement systems are an example of aleatoric uncertainty. An example of aleatoric uncertainty is a low-resolution image of an animal that could be a dog or a wolf but cannot be determined because of the low resolution. Another example could be that there is both a dog and a wolf in the image, but the decision maker (for example a machine learning model) needs to decide if it is a dog or a wolf. As a result, falsely labeled data is also an example of aleatoric uncertainty (Gawlikowski *et al.*, 2023).

ii. Epistemic uncertainty: Epistemic (from *episteme*, meaning *knowledge* in ancient Greek) uncertainty refers to uncertainty caused by a lack of knowledge, or in other words, uncertainty due to ignorance. Sticking with the coin example, an example of epistemic uncertainty might be the uncertainty in the experiment to determine whether a coin is fair or not - perhaps with the further assumption that the probability of having a head or a tail has a constant value. In this experiment, the coin fairness is not inherently random and can be determined with increased certainty simply by collecting additional data, i.e., flipping the coin again.

For machine learning systems, distribution, and domain shifts are examples of epistemic uncertainty (Gawlikowski *et al.*, 2023). For example, a model trained on images of adult dogs and wolves may perform poorly in deciding whether a puppy is a wolf or a dog. Similarly, a cat image is also unknown to the model and would not be classified as such. In both cases, the uncertainty is not caused by the images (data) themselves. For example, if the images are not so blurred that it is impossible to tell what is in them, which would be an example of aleatoric uncertainty, the image is clear to people or machines who know what puppies or cats are. So the uncertainty about whether the puppy in the picture is a dog or a wolf is due to modeling, or in other words, the model's lack of knowledge.

Uncertainty Quantification Methods For Neural Networks

For each output to be predicted, standard deep learning models predict a single scalar value, but such predictions provide limited information about the confidence of the prediction. For example, the $softmax$ output layer can provide confidence interpretations. The $softmax$ layer is already widely used in classification tasks. It normalizes the scalar values associated with a potential class

to which a data sample may belong, and the sum of these scalar values to one. Therefore, the output of the softmax layer can be interpreted as a probability distribution over all possible classes. Different confidence metrics are defined over these softmax outputs interpreted as probability distributions (Zhang *et al.*, 2020). For example, the highest value of the softmaxed vector can be considered as the confidence of the prediction, since this value is the value associated with the predicted class (Pearce *et al.*, 2020). However, these values are not robust and often overestimate the true probability of being classified correctly (Guo *et al.*, 2017). Therefore, these metrics are not fully reliable. In addition, soft-max is not used for regression tasks.

An often better way to evaluate the confidence of deep learning models is to predict a distribution in the output space, rather than predicting a single estimate. One group of methods for predicting a distribution is Bayesian neural networks, which propose to learn the weights of the neural network as probability distributions instead of scalar values. In this way, the output of the model also becomes a probability distribution for a given input. Bayesian neural networks use the Bayes rule to infer the distribution of weights $p(\theta|D)$ given the data D:

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} = \frac{p(D|\theta)p(\theta)}{\int p(D|\theta)p(\theta)d\theta}. \tag{2.9}$$

However, the integral required for the calculation is often intractable and is therefore approximated.d (Kingma, 2013; Wen *et al.*, 2018).

Monte Carlo Dropout (Gal and Ghahramani, 2016) (MC-Dropout for short) approximates Bayesian neural networks by using dropout layers not only during training but also during prediction, where they are classically disabled. Standard dropout layers are not used during prediction time to use all neurons without dropping any of them, resulting in a full model performance. However, with random dropping on during prediction, the same input can be fed into the model to make unequal predictions, and therefore an output distribution can be predicted for an input sample. While there is almost no additional training time, prediction for MC-Dropout is computationally expensive due to multiple forward passes.

Another way to predict a distribution is deep ensembles, where the same architectures are trained independently and with different initialization weights (Lakshminarayanan *et al.*, 2017). While deep ensembles are easy to implement and often perform well, they are expensive in terms of computation and memory because each independent model is trained from scratch. They are also expensive in terms of prediction time due to the number of forward passes as much as the number of individual networks in the ensemble. To alleviate this problem, several methods have been proposed that make minor architectural changes to mimic deep ensembles while having significantly lower computational and memory requirements (Turkoglu *et al.*, 2022; Wen *et al.*, 2020).

On the other hand, some methods allow the user to the user to specify a desired level of certainty, in other words, a predefined confidence level. The method then returns a prediction (e.g. prediction interval) containing the ground truth value with that predefined confidence (Romano *et al.*, 2019).

Calibration

In safety-critical situations, it can be critical to determine the certainty of model predictions accurately. Deep learning models often use soft-max activation for classification tasks, which can provide a probability distribution as a prediction. In addition, a higher value of the output for a particular class means that the model is more confident in its prediction. Thus, the output of a soft-max activation is typically interpreted as the confidence score. For example, an output vector $y_i = f(x_i) = [0.1, 0.2, 0.7]$ for an input sample x_i and the model f can be interpreted as saying that the model predicts that the chances of x_i being correctly classified in each class are 10%, 20%, and 70%. However, this confidence interpretation of softmax outputs is overconfident for typical modern deep learning networks (Guo *et al.*, 2017). For the example above, this means that the expected probability of x_i being correctly classified in class 3^{rd} is significantly less than 70% for most modern deep learning networks. When this is the case, the model is said to be "uncalibrated".

Formally, assuming that $f(X) = (\hat{Y}, \hat{P})$, where $\hat{Y}$ is a class prediction and $\hat{P}$ is the confidence of the prediction, i.e. the probability that it is correctly classified, Guo *et al.* (2017) defines perfect calibration as:

$$\mathbb{P}(\hat{Y} = Y|\hat{P} = p) = p, \qquad \forall p \in [0, 1] \tag{2.10}$$

Evaluation of calibration quality: Calibration quality can be examined and measured by using reliability diagrams and expected calibration error, which will now be discussed.

i. Reliability Diagrams: Estimating the expected accuracy from a limited number of samples can be done by grouping predictions into N equally sized interval bins and calculating the accuracy of each bin (DeGroot and Fienberg, 1983; Niculescu-Mizil and Caruana, 2005; Guo *et al.*, 2017). Let B_n be the indices of samples whose prediction confidence falls in the interval $I_n = \left(\frac{n-1}{N}, \frac{n}{N}\right]$. Let $\hat{y}_i$ and y_i be the predicted and true class labels for sample i, the accuracy of the B_n is

$$\text{acc}(B_n) = \frac{1}{|B_n|} \sum_{i \in B_n} \mathbf{1}(\hat{y}_i = y_i). \tag{2.11}$$

Where $\hat{p}_i$ is the confidence for sample i, the average confidence within bin B_n is defined as

$$\text{conf}(B_n) = \frac{1}{|B_n|} \sum_{i \in B_n} \hat{p}_i \tag{2.12}$$

Note that a perfectly calibrated model will have $\text{acc}(B_n) = \text{conf}(B_n)$ for all $n \in 1, ..., N$.

ii. Expected Calibration Error: While reliability diagrams are useful for visualization and deeper analysis, a scalar performance metric for calibration allows an effective comparison of different methods and models. A scalar metric that satisfies this requirement is the expected value of the absolute difference between accuracy and confidence, formally expressed as (Guo *et al.*, 2017)

$$\mathop{\mathbb{E}}_{\hat{P}} \left[\left| \mathbb{P}\left(\hat{Y} = Y|\hat{P} = p\right) - p \right| \right]. \tag{2.13}$$

An approximation of 2.13 with a limited number of samples, similar to the reliability plots, can be made by grouping the predictions into N equally sized bins. In this way, the expected calibration error (ECE) Naeini *et al.* (2015) evaluates the expected value of the absolute difference between accuracy and confidence. More specifically, where M is the total number of samples, ECE is formally given by

$$\mathrm{ECE} = \sum_{n=1}^{N} \frac{|B_n|}{M} \Big| acc(B_n) - conf(B_n) \Big| \tag{2.14}$$

There are other similar calibration error metrics besides ECE. The static calibration error (Nixon *et al.*, 2019) uses within-bin confidence and accuracy metrics similar to ECE, but instead of evaluating the calibration error over the predicted class like ECE, it evaluates the average calibration error over all classes. The adaptive calibration error (Nixon *et al.*, 2019) shapes the bins so that each has the same number of predictions, rather than grouping them at equal intervals.

Methods to improve calibration: One way to improve the calibration is to modify the training procedure. Label smoothing reduces the overconfidence of the model and thus improves the calibration by changing the false labels to be small equal values instead of zero and reducing the true label value accordingly Szegedy *et al.* (2016); Müller *et al.* (2019). Thulasidasan *et al.* (2019) and Patel *et al.* (2021) use data augmentation techniques to improve the calibration. They use the mixup (Zhang *et al.*, 2017), which randomly sums parts of two input samples and their labels, and on-manifold adversarial data as data augmentation methods, respectively.

After training the machine learning models, post-hoc calibration methods are applied to improve the calibration. A small subset of the training set can be omitted for post-hoc calibration. It is desired that the accuracy of the model is not affected by the calibration process. Temperature scaling Guo *et al.* (2017) is an effective post-hoc calibration method for classification tasks, where the elements of the input vector fed into the softmax function are divided by a positive scalar value called the temperature. Similar to temperature scaling, Levi *et al.* (2022) suggests standard deviation scaling for regression tasks.

A reduced model uncertainty means better modeling of the existing data distribution and therefore leads to a better calibration. Therefore, methods that reduce the uncertainty by reducing the model uncertainty, such as Bayesian-based methods, deep ensembles, and similarly efficient implementations of these methods, improve the calibration (Turkoglu *et al.*, 2022).

Out-of-Distribution Detection

Out-of-distribution (OOD) detection is a critical task for the applicability of machine learning systems in real-world scenarios. Machine learning models are typically trained using a closed-world assumption, where the test set is assumed to be independent and identically distributed. It is assumed that the samples are not connected and therefore do not influence each other and that they are drawn from the same distribution.

However, the models are used in real-world or open-world scenarios where test samples may be out-of-distribution. OOD samples can be due to semantic shift, where the samples may belong to different classes, or covariate shift, where the samples may be from a different domain (Yang *et al.*, 2021).

There is a high risk of misclassification for out-of-distribution (OOD) samples, so ideally the model should refrain from classifying and inform the user that it cannot confidently estimate the result. The machine learning model trained for in-distribution (ID) scenarios is denoted f. The selective classifier then consists of a pair of (f, g), where g is a selector for whether the prediction should be made (El-Yaniv *et al.*, 2010). Specifically

$$(f, g)(x) = \begin{cases} f(x), & \text{if } g(x) = 1 \\ \text{No classification}, & \text{if } g(x) = 0. \end{cases} \tag{2.15}$$

Let $\kappa_f(x)$ be the confidence of the model f for an input sample x. $g(x)$ can be modelled as

$$g(x) = \begin{cases} g(x) = 1, & \text{if } \kappa_f(x) \geq \Theta \\ g(x) = 0, & \text{otherwise.} \end{cases} \tag{2.16}$$

where Θ is a threshold value (Geifman and El-Yaniv, 2017). For a multi-class classification task using a softmax output activation, even if the model is not well calibrated, e.g. ECE is high, it is assumed that the confidence of one input is greater than or equal to the confidence of another input, i.e. $\kappa_f(x_1) \geq \kappa_f(x_2)$, if the output value of the input is greater than or equal to the other input, i.e. $f(x_1) \geq f(x_2)$. In this case, it is also possible to model $g(x)$ as follows:

$$g(x) = \begin{cases} g(x) = 1, & \text{if } f(x) \geq \theta \\ g(x) = 0, & \text{otherwise.} \end{cases} \tag{2.17}$$

$g(x) = 1$ in the above equation can be seen as an "ID" scenario and $g(x) = 0$ can be seen as an "OOD" scenario (Sun *et al.*, 2022).

Hendrycks and Gimpel (2016) proposed a simple baseline for detecting OOD samples without having to choose a threshold like θ: First, the maximum softmax probability, i.e. the softmax probability for the predicted class, is computed for test samples of the ID and OOD datasets. Then, using these datasets as different groups, threshold-independent metrics such as ROC-AUC and PR-AUC are computed from the maximum softmax probabilities of the test samples.

2.5 Deep Learning for Genomics

Deep learning methods for genomic applications will be introduced in this section, focusing on DNA data, and the deep learning areas to which this thesis contributes: self-supervised learning, model design, uncertainty quantification, and software applications.

Self-Supervised Learning for Genomic Applications

Self-supervised learning in genomics is dominated by models adapted from methods developed for NLP or computer vision applications. Here is a brief introduction to how self-supervised learning methods work and how they are adapted to genomics.

i. An example of such architectures is DNA-BERT (Ji *et al.*, 2021), where BERT (Devlin *et al.*, 2018), a self-supervised method developed for NLP tasks, is adapted to genomic tasks. One challenge in adapting BERT to perform well on DNA data is the mismatch between the complexity of words in natural language processing and nucleotides in DNA. In particular, words, which are fed into the model as so-called tokens, are inherently complex with thousands of them, whereas there are only 4 different nucleotides in a DNA sequence and these nucleotides alone do not contain complex information. Therefore, the tokenization of DNA sequences requires special care. DNA-BERT adopts BERT by tokenizing k-mers instead of nucleotides, with values of k = 3,4,5,6. This means that k-mers are one-hot encoded into a representation vector of size k^4. A different self-supervised model is trained for each value of k. However, multiple values with none clearly outperforming the others may indicate that the application of NLP models to sequential genomic data is still an active area of research (Zhou *et al.*, 2023).

ii. Another example is CPCProt (Lu *et al.*, 2020), which incorporates the contrastive loss to the genomics data and trains a deep learning model for genomics with very similar training settings compared to the work introducing contrastive predictive coding (Oord *et al.*, 2018). Specifically, it is trained by predicting future sequence patches, each patch consisting of 11 amino acids.

iii. Contrastive-sc (Ciortan and Defrance, 2021) is a method that utilizes contrastive loss for cell clustering based on single-cell RNA sequencing data. Specifically, two randomly masked copies of a sequence are trained in such a way that the agreement between the representations of these copies is maximized, while the agreement between copies from different sequences is minimized. In this setting, the training has substantial similarity to the common use of contrastive loss for computer vision tasks. One difference between the adapted method and computer vision tasks is that in contrastive-sc, the copies derived from the same sequence are randomly masked, whereas, in computer vision tasks, many more complicated data augmentation techniques are used, such as cropping, resizing, rotating, adding noise, blur, color distortion, or Sobel filtering (Chen *et al.*, 2020). Since such augmentation techniques used in computer vision do not produce meaningful augmented samples for DNA sequences, they are not used for data augmentation and can therefore be mentioned as a limitation of the adapted techniques.

iv. There are certain features of genome sequences that self-supervised approaches developed for other fields and then adapted to genomics usually do not take into account, leading to less effective results and limited applications. Gündüz *et al.* (2023) aims to address this by exploiting the reverse-complement property of DNA sequences. Reverse-complement sequences have already been used as part of various supervised techniques for DNA data (Shrikumar *et al.*, 2017; Quang and Xie, 2019), but it had not been integrated to self-supervised learning for this type. In addition, the work takes into account the fact that nucleotides and multi-nucleotide k-mers contain less complex information compared to words in human languages and addresses this issue by proposing a loss function that maximizes the mutual information between the learned representations and much longer sequences compared to multi-nucleotide k-mers.

v. Another example of self-supervised learning for the DNA data type is auto-regressive models trained by predicting the next nucleotides or k-mers, which is also used in Self-GenomeNet paper (Gündüz *et al.*, 2023) as baseline methods.

vi. DNABERT-2 (Zhou *et al.*, 2023) addresses several problems associated with k-mer tokenization, such as information leakage to neighbouring tokens after tokenization, and proposes to use a data compression algorithm called byte pair encoding (Sennrich, 2015; Gage, 1994) instead of k-mer tokenization.

Model Design for Genomics

Experts design and develop deep learning architectures based on a deep understanding of both machine learning and the specific domain in which they are used. In addition, a significant amount of testing and improvement is required.

Models developed for sequential genomic data based on convolutional layers include DeepVirFinder (Ren *et al.*, 2020), ViraMiner (Tampuu *et al.*, 2019), CHEER (Shang and Sun, 2020), PPR-Meta (Fang *et al.*, 2019), DeepSEA (Zhou and Troyanskaya, 2015), Basenji (Kelley *et al.*, 2018), DeepBind (Alipanahi *et al.*, 2015), Coda (Koh *et al.*, 2017), and models in Fiannaca *et al.* (2018), Bartoszewicz *et al.* (2022), and Zeng *et al.* (2016). Depending on the task or dataset, there may be global average or global max pooling after the convolutional layer, especially if the predicted label of the data represents a feature that belongs to the complete sequence data, such as whether the data sequence is either viral or bacterial nucleic acid, rather than a label expressed as a function of position in the sequence.

Some genomics architectures developed by experts include recurrent layers such as Seeker (Auslander *et al.*, 2020) or DeepMicrobes (Liang *et al.*, 2020). DanQ (Quang and Xie, 2016) and the work of Wang *et al.* (2019) use recurrent layers on top of convolutional layers, i.e. the output tensor of convolutional layers is fed into a recurrent layer so that some repeating motifs like k-mers can be learned by convolutional layers and the sequential relationship between these patterns can be learned by a recurrent layer.

The above models can provide a good basis for a meaningful search space for automated neural architecture search. Contributions in this thesis by Gündüz *et al.* (2023), Scheppach *et al.* (2023) define search spaces based on recurrent layers or global pooling layers (only the first work) placed after convolutional layers, which proves to be very effective, outperforming all expert-designed architectures in the performed benchmarks.

There are also other attempts to develop effective automated neural architecture search for genomics. Automated Modeling for Biological Evidence-based Research (AMBER) (Zhang *et al.*, 2021b), applied the ENAS (Pham *et al.*, 2018) method to the genomics dataset for automatic design of convolutional networks. AMBIENT (Zhang *et al.*, 2021a) aims to reduce the need for computational power by generating architectures for new tasks based on data descriptors such as the type of regulatory feature. However, these NAS methods are based only on convolutional networks and do not incorporate recurrent layers into their search space.

Uncertainty Quantification for Genomics

Uncertainty quantification can be essential in safety-critical areas such as genomics or medicine, where high uncertainty in the models developed can lead to the models not being used. Therefore, performance benchmarking of uncertainty quantification methods on tasks such as out-of-distribution detection or calibration are quite common on data types such as genomics data like DNA or RNA sequences (Turkoglu *et al.*, 2022; Gündüz *et al.*, 2023; Vahidi *et al.*, 2024; Bajwa *et al.*, 2024) or medical image data (Turkoglu *et al.*, 2022; Mehrtash *et al.*, 2020). Some studies carried out on DNA and RNA sequences are further examined as follows:

i. Bajwa *et al.* (2024) points out a high uncertainty in the predictions on variant sequences, indicating that the developed models have limited generalization ability to out-of-distribution samples for genomic sequence-to-activity tasks. They also suggest focusing on developing calibrated uncertainty estimates for genomic sequence-to-activity models using methods such as conformal prediction that generate statistically rigorous uncertainty intervals.

ii. Gündüz *et al.* (2023) shows that the conformal prediction coverage on the test set is 89.6% for the coverage target, which has an expected coverage of 90%, optimized to cover the amount of 90% on the validation set. Since these values are very close, it should be noted that the coverage on the validation set generalizes well to the test set.

iii. MacDonald *et al.* (2023) shows that simple Bayesian deep learning models applied to oncology can significantly improve generalization, which is valuable as overconfidence of deep learning models under data distribution shifts remains a major issue preventing their use in production.

iv. Turkoglu *et al.* (2022), including the author of this thesis, show that the state-of-the-art implicit ensemble technique FiLM-Ensemble, also proposed in the same paper, can also be successfully applied to DNA data, effectively yielding significantly better-calibrated models compared to deep ensemble models.

v. Hie *et al.* (2020) demonstrates how uncertainty can improve the experimental lifecycle by helping researchers prioritize sample-efficient experiments in biological areas such as single-cell transcriptomics and protein engineering.

Software Libraries for Applied Deep Learning for Genomics

As access to more powerful computing resources has become easier and machine learning methods have proven their success in various fields, deep learning methods are on the rise in the field of genomics. Software libraries such as Selene (Chen *et al.*, 2019) and Janggu (Kopp *et al.*, 2020) have emerged along with this trend. However, the focus of these libraries is on human genomics, and their capabilities for other genomes, such as microbial, are limited. In addition, these libraries are written in Python, a commonly used language for deep learning.

The software library deepG, which is a contribution in this thesis (Mreches *et al.*, 2024), is mainly based on the R language, another commonly used language among bioinformaticians. Two other contributions of this thesis, Self-GenomeNet, and GenomeNet-Architect (Gündüz *et al.*, 2023; Gündüz *et al.*, 2023), are also integrated into the library, allowing these methods to be easily used, trained, tested, and deployed.

3 Conclusion

As concluding remarks, the contributions of this thesis are discussed along with potential future work.

3.1 Discussion of Contributions

All data, model, and evaluation-related aspects of applied deep learning in genomics are addressed in this thesis, providing a comprehensive range of contributions to its main topic, which is the design and optimization of deep learning methods for genomic sequencing data.

i. Contributions to self-supervised learning for genomics: The contribution to the data-related part focuses on how large amounts of unlabeled data can be utilized, which is particularly beneficial when labeled data is limited. This can often be the case in genomics, as data labeling can be expensive. Self-GenomeNet (Gündüz *et al.*, 2023) shows that self-supervised methods more suitable to genomic data can be proposed based on expert knowledge in genomics. Specifically, a self-supervised method is proposed that makes more effective use of reverse complement sequences to improve the quality of learned representations during self-supervised training of the machine learning model. The effectiveness of the method is demonstrated in a wide range of settings such as semi-supervised (also called label-scarce) setting, transfer learning, and linear evaluation, and on a wide range of tasks such as bacteriophage detection, effector gene prediction, fungal-protozoa classification, and chromatin feature estimation, outperforming the standard supervised deep learning training despite using 10 times less annotated data samples.

ii. Contributions to automated model design for genomics: The contribution to the model-related part focuses on the automated design of model architectures. Although there are several standard models in different fields of deep learning, such as ResNets (He *et al.*, 2016) or visual transformers (Dosovitskiy, 2020) in computer vision or BERT (Devlin *et al.*, 2018) in natural language processing, there is no widely accepted standard model in genomics. Moreover, even in fields with such standard models, neural architecture search methods that optimize the models based on specific datasets are beneficial to improve predictive performance. The concurrent work of GenomeNet-Architect (Gündüz *et al.*, 2024) and the neural architecture search methods adapted in the work of Scheppach *et al.* (2023) both show the effectiveness of automated model design methods for genomics data, especially when an appropriate search space is proposed, which may require expert knowledge. A major contribution of Scheppach *et al.* (2023) is that several neural architecture search methods developed for other areas of deep learning can outperform the expert-designed methods in genomics and that performance differences between separate neural architecture search methods can often be insignificant. Because running neural architecture search algorithms separately is computationally expensive, this study could be performed on only one dataset (Zhou and Troyanskaya, 2015). It could also be argued that an additional potential improvement point of the work was the limited number of expert-designed models as baselines.

This study showed the need for proposing a well-defined search space along with an appropriate architecture search strategy, which should be empirically proven to outperform on a wider range of datasets and expert-designed baselines, which are points fully addressed by the work of Gündüz *et al.* (2024). The proposed method (Gündüz *et al.*, 2024) outperforms a considerable amount of expert-designed models on a newly introduced benchmark on virus classification tasks for different sequence length values and outperforms all baselines on another existing benchmark for pathogenicity prediction in which the baseline results are taken from the work of Bartoszewicz *et al.* (2020).

iii. Contributions to uncertainty quantification and calibration for genomics: Contributions related to the evaluation part will be made in the area of uncertainty quantification, i.e. the uncertainty in the evaluated performance metrics together with the relevant uncertainty quantification subtasks such as calibration improvement and out-of-distribution detection. Being informed about the risk of obtaining unwanted results based on the model decisions can be highly beneficial for genomic datasets and tasks, as the use cases can be health-related and thus safety-critical. The contributions of the thesis in this area (Gündüz *et al.*, 2023; Turkoglu *et al.*, 2022; Vahidi *et al.*, 2024) are applications of uncertainty quantification methods on genomic datasets. Gündüz *et al.* (2023) analyzes uncertainty quantification approaches on a multi-target regression task in genomics for both in-distribution and out-of-distribution scenarios. The author of this thesis contributes to the work of Turkoglu *et al.* (2022) by evaluating the performance of the proposed state-of-the-art implicit deep ensemble method on genomics data and shows that the calibration of genomics tasks can be significantly improved. In addition, the trade-off between calibration and prediction performance that can be achieved by adjusting the hyperparameter introduced by the method is analyzed on the genomics dataset. Last but not least, the DNA dataset in the thesis also proves that the proposed FiLM-Ensemble also works with one-dimensional convolutional layers, serving a more general purpose. Similarly, the author of this paper contributes to the work of Vahidi *et al.* (2024) by testing the proposed method in the article, which can be built upon existing self-supervised learning methods to improve the calibration of the models. The proposed method is built on another contribution of this thesis, Self-GenomeNet, a self-supervised learning model developed for genomics. Therefore this work (Vahidi *et al.*, 2024) can further be attributed to contributions made in the field of self-supervised learning. Such analyses of uncertainty quantification methods on genomics datasets are valuable for testing existing deep learning methods on this data type, an area where these methods are invaluable. However, recently proposed methods in deep learning often omit to test the method on this data type and often test the proposed methods only in more mainstream deep learning fields such as computer vision and natural language processing.

iv. Contributions to software development for genomics: The main contribution of Mreches *et al.* (2024) is a software library called "deepG". This library is written in R, a programming language commonly used by bioinformaticians. The goal of this work is to facilitate the training, testing, and deployment of machine learning models in genomics. In line with this goal, two major contributions of this thesis, Self-GenomeNet (Gündüz *et al.*, 2023) and GenomeNet-Architect (Gündüz *et al.*, 2024) are integrated into the deepG library. This integration makes the use of these methods more accessible to researchers, allowing them to use unlabeled data more effectively, train models on pre-trained self-supervised models, optimize data-specific model architectures, or use models that have been shown to produce remarkable results on multiple datasets.

3.2 Future Directions

Foundational models and the development of unlabelled neural architecture search techniques for genomic data can be potentially powerful successors of this work. This section discusses future research ideas in these directions.

i. Foundational models designed specifically for genomic data: There is substantial quality work in deep learning, especially in fields such as natural language processing. One recent breakthrough in deep learning is natural language processing (NLP) with large language models (LLMs) (Min *et al.*, 2023). LLMs are trained with a huge amount of unlabeled text data using self-supervised training methods. In genomics, unlabeled data is abundant, similar to NLP. Therefore, research into this direction is very promising and can be very influential. In fact, such models have already become influential for the genomics field (Rives *et al.*, 2021; Dalla-Torre *et al.*, 2023; Zhou *et al.*, 2023; Mendoza-Revilla *et al.*, 2024).

One direction of LLMs for genomics can be designed based on the idea presented in Self-GenomeNet (Gündüz *et al.*, 2023), a contribution of this thesis. Gündüz *et al.* (2023) shows that integrating reverse-complement effectively into the self-supervision task can improve the quality of the learned representations. Like contrastive predictive coding (Oord *et al.*, 2018), Self-GenomeNet uses an autoregressive model enabled by a recurrent layer. LLMs are transformer models, which are based on self-attention layers. The connection between recurrent networks and transformer models is also well-studied. "Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention" (Katharopoulos *et al.*, 2020) and "Transformers are Multi-State RNNs" (Oren *et al.*, 2024) illustrate the close connections between transformer models and RNNs, even in the title of papers. Autoregressive LLMs such as the method in Katharopoulos *et al.* (2020) to utilize the reverse-complement sequences as Self-GenomeNet (Gündüz *et al.*, 2023) describes could be worth investigating.

ii. Unlabeled neural architecture search for genomics: Some studies investigate neural architecture search methods for cases where human-annotated data is not available and conclude that neural architecture search methods using self-supervised learning targets achieve comparable results to those using standard supervised learning on labeled data (Kaplan and Giryes, 2020; Liu *et al.*, 2020). Therefore, these methods address the need for labeled data for neural architecture search by using already existing SSL techniques and without significant loss in performance. In section 2.2, it was mentioned that self-supervised learning methods create labels using the unlabeled data sample and then the model is trained similarly to the supervised learning case using these generated labels. Considering that self-supervised methods indeed create some labels (although computer-designed and automated) and are effective in learning good representations, it is not surprising to the author of this thesis that these methods find such successful architectures. Both Liu *et al.* (2020) and Kaplan and Giryes (2020) use a NAS method called DARTS (Liu *et al.*, 2018b) along with various self-supervised learning methods. For example, Kaplan and Giryes (2020) uses SimCLR (Chen *et al.*, 2020) as a self-supervised learning method. Nevertheless, a NAS method can only perform well if it has a good search space, which is already designed by Gündüz *et al.* (2024) and Scheppach *et al.* (2023), which are contributions of this thesis. In addition, Scheppach *et al.* (2023) already proves that the DARTS algorithm used in the mentioned work (Liu *et al.*, 2020; Kaplan and Giryes, 2020) can be successfully applied to genomic data using its defined search space. Finally, the self-supervised learning method designed for genomic data, Self-GenomeNet (Gündüz *et al.*, 2023), which is another contribution of this thesis, combined with

GenomeNet-Architect (Gündüz *et al.*, 2024) or a NAS model from Scheppach *et al.* (2023) would be a good fit and could be worth investigating. Both these automated model design methods and the self-supervised learning method explore the setting with recurrent layers on top of convolutional layers, so combining these methods may be possible without major changes in model design or search space. Last but not least, this research could also impact the field of deep learning in genomics, especially due to the cost of expert annotated data in the field of health and genomics.

Part II

Contributions

4 Contributions to Self-Supervised Learning for Genomics

4.1 A Self-supervised Deep Learning Method for Data-efficient Training in Genomics

Contributing Article

Gündüz HA, Binder M, To XY, Mreches R, Bischl B, McHardy AC, Münch PC, Rezaei M (2023). "A self-supervised deep learning method for data-efficient training in genomics." *Communications Biology*, **6**(1), 928. doi:10.1038/s42003-023-05310-2

Declaration of Contributions

Hüseyin Anil Gündüz contributed to this paper as the first author with the following significant contributions:
Hüseyin Anil Gündüz introduced the main idea, proposed the baselines, designed the experiments, and was chiefly responsible for coding. Hüseyin Anil Gündüz trained, evaluated, and analyzed the proposed method and the baselines. Hüseyin Anil Gündüz authored most of the manuscript.

Contribution of the coauthors:
Martin Binder contributed to the supervision of the project. Xiao-Yin To contributed to code development, mostly for the CPC baseline and the training function. Rene Mreches contributed to the code development, particularly regarding data processing and language model baselines. The paper is mainly supervised by Mina Rezaei and Philipp C. Münch, from a machine learning and bioinformatics perspective, respectively.

All authors helped with the editing of the paper.

communications biology

ARTICLE

https://doi.org/10.1038/s42003-023-05310-2 OPEN

A self-supervised deep learning method for data-efficient training in genomics

Hüseyin Anil Gündüz[1,2], Martin Binder[1,2], Xiao-Yin To[1,2,3,4], René Mreches[3,4], Bernd Bischl[1,2], Alice C. McHardy[3,4], Philipp C. Münch[3,4,5,6,7✉] & Mina Rezaei[1,2,7✉]

Deep learning in bioinformatics is often limited to problems where extensive amounts of labeled data are available for supervised classification. By exploiting unlabeled data, self-supervised learning techniques can improve the performance of machine learning models in the presence of limited labeled data. Although many self-supervised learning methods have been suggested before, they have failed to exploit the unique characteristics of genomic data. Therefore, we introduce *Self-GenomeNet*, a self-supervised learning technique that is custom-tailored for genomic data. *Self-GenomeNet* leverages reverse-complement sequences and effectively learns short- and long-term dependencies by predicting targets of different lengths. *Self-GenomeNet* performs better than other self-supervised methods in data-scarce genomic tasks and outperforms standard supervised training with ~10 times fewer labeled training data. Furthermore, the learned representations generalize well to new datasets and tasks. These findings suggest that *Self-GenomeNet* is well suited for large-scale, unlabeled genomic datasets and could substantially improve the performance of genomic models.

[1] Department of Statistics, LMU Munich, Munich, Germany. [2] Munich Center for Machine Learning, Munich, Germany. [3] Department for Computational Biology of Infection Research, Helmholtz Center for Infection Research, 38124 Braunschweig, Germany. [4] Braunschweig Integrated Centre of Systems Biology (BRICS), Technische Universität Braunschweig, Braunschweig, Germany. [5] German Center for Infection Research (DZIF), partner site Hannover Braunschweig, Braunschweig, Germany. [6] Department of Biostatistics, Harvard School of Public Health, Boston, MA, USA. [7] These authors jointly supervised this work: Philipp C. Münch, Mina Rezaei. ✉email: philipp.muench@helmholtz-hzi.de; mina.rezaei@stat.uni-muenchen.de

In bioinformatics, using unlabeled data to augment supervised learning can reduce development costs for many machine learning (ML) applications that would otherwise require large amounts of annotation that are expensive to acquire, such as functional annotation of genes[1] or chromatin effects of single nucleotide polymorphisms. This is particularly the case in genomics due to the availability of large quantities of unlabeled sequence data from large databases and metagenomic studies.

In contrast to supervised methods, self-supervised learning (SSL) techniques learn representations that contain information about the properties of the data without relying on human annotation. The concept of SSL has been studied for several years in the field of ML. These SSL methods are unsupervised tasks, that are trained prior to the actual supervised training. Then, instead of training a supervised model from scratch, the representations learned from the SSL method can be used as a starting point for downstream supervised tasks such as taxonomic prediction or gene annotation. In this way, the pre-trained models can be used as a starting point that contains meaningful representations from the SSL tasks[2,3]. Several different methods for self-supervised representation learning have been proposed, e.g., in the fields of natural language processing (NLP)[4–8] and computer vision (CV)[3,9–11]. However, only a limited number of SSL methods have been developed for bioinformatics, and even fewer for omics data[12–15]. Thus, SSL has not yet seen such widespread adoption and remains an important and challenging endeavor in this field.

Existing methods for representation learning on omics-data have typically been adapted from other application fields of DL such as NLP or CV[16–18]. For example, DNA-Bert[19], which identifies conserved sequence motifs and candidate functional genetic variants, is an adaptation of BERT[2], which is a form of language model (LM)[20] that predicts masked tokens. These tokens are words in NLP, and nucleotides or k-mers in genome sequences. Contrastive-sc[18] is a method adapted from CV used for cell clustering based on single-cell RNA sequencing data. It creates two copies of each sequence with randomly masked nucleotides and then trains the network to maximize the agreement between the copies using a contrastive loss function, a method commonly used in CV[3]. CPCProt[17] is an adaptation of the contrastive predictive coding (CPC)[21] to protein data and is trained by predicting future amino acid sequence patches. However, there are specific properties of genome sequences that these methods do not take into account, resulting in non-optimal representations and limited use. Although used in several supervised methods, reverse-complement (RC) sequences have not been integrated into SSL methods. Additionally, nucleotides and k-mers contain low-content information compared to words in natural languages, and this is not taken into account when SSL methods developed for NLP are applied to genome data.

Self-GenomeNet overcomes these limitations. First, *Self-GenomeNet* uses RC sequences to create symmetry in the architecture. This increases predictive performance and reduces the number of model parameters. This may also have the desirable side effect of implicitly encoding RC-awareness in our architecture. Secondly, *Self-GenomeNet* predicts targets of different lengths as an SSL task. This way, a wider range of semantic relationships within the DNA data is learned. Finally, due to the way recurrent networks process their data, representations of many subsequences at different length scales are evaluated simultaneously within a single training step, leading to increased computational efficiency.

Self-GenomeNet makes more efficient use of unannotated genomic data to substantially improve various genomic tasks when limited labeled data is available, making it more suitable for genomic applications than existing SSL methods. In computational biology, such pre-training learning schemes could benefit a wide range of ML tools, where large amounts of unlabeled data such as metagenomic sequences are available to improve supervised models built on nucleotide-level training datasets.

Results

***Self-GenomeNet* is an efficient self-supervised pre-training method, tailored for genomics**. *Self-GenomeNet* is a SSL method, where the network is trained without the need of labels on available sequential genome data. Then this network, particularly the trained weights of this network, can be used as the initial point of the model that will be trained for the supervised tasks, which are also called downstream tasks. We provide a model, which is trained on bacteria, virus, and human data without using labels. This model, named *generic Self-GenomeNet*, demonstrates robust performance across diversified tasks, providing researchers a readily accessible solution to leverage the power of our model in their own studies, particularly for their own supervised tasks. We have uploaded the trained generic *Self-GenomeNet* model to GitHub for easy access (*see self.genomenet.de*). Additionally, we have prepared interactive coding notebooks that provide detailed instructions on how to use this model to obtain embeddings of data and how to apply it to other datasets.

Self-GenomeNet learns representations of genome sequences through a defined pre-training task that does not require labels. This task is as follows: For a given input sequence of length N, $S_{1:N}$, an embedding of a subsequence $S_{1:t}$, predicts the embedding of the RC of the remaining subsequence $\bar{S}_{N:t+1}$. Thus, the model encodes in the learned representation of the given subsequence the essential information necessary to predict the RC of a neighboring subsequence. *Self-GenomeNet* encodes these two subsequences through a representation network consisting of a convolutional encoder network f_θ, and a recurrent context network C_ϕ (Fig. 1a). The architecture of *Self-GenomeNet* is implemented to perform this prediction for multiple values of t in one iteration, enabling a more efficient training procedure.

The network of *Self-GenomeNet* takes both $S_{1:N}$ and $\bar{S}_{N:1}$ as inputs. *Self-GenomeNet* encodes these two subsequences through a representation network consisting of a convolutional encoder network and a recurrent context network. As a result of the proposed architecture, the representations of subsequences $S_{1:t}$ and $\bar{S}_{N:t+1}$ are computed for multiple values of t as intermediate outputs of the context network, while the whole sequences $S_{1:N}$ and $\bar{S}_{N:1}$ are encoded. Later, on top of the embedding representation, a linear prediction layer q_η estimates the embedding of $\bar{S}_{N:t+1}$ from the embedding of $S_{1:t}$ using a contrastive loss against other random subsequences. Due to the symmetry of this design, q_η is also used to predict the embedding of $\bar{S}_{N:t+1}$ from the embedding of $S_{1:t}$. Although only one prediction is shown in the figure for visual simplicity, the prediction is computed for multiple values of t. Contrastive loss is used for the optimization, meaning that the network is optimized so that the sequences (e.g., $S_{1:t}$) aims to predict the representation of the RC of its own neighbor (e.g., $\bar{S}_{N:t+1}$) among other representations in the training batch. The convolutional encoder network, the recurrent context network, and the linear prediction layer each consist of a single layer in our experiments to keep the architecture simple; however, more complex architectures are possible. The hyperparameters of the convolutional and recurrent networks are mentioned later in the paper, in the "Network Architecture Design" section.

After self-supervised training of the representations, these representations can be used for downstream supervised tasks by constructing a supervised deep learning model consisting of f_θ and C_ϕ, followed by a fully connected output. The weights of f_θ

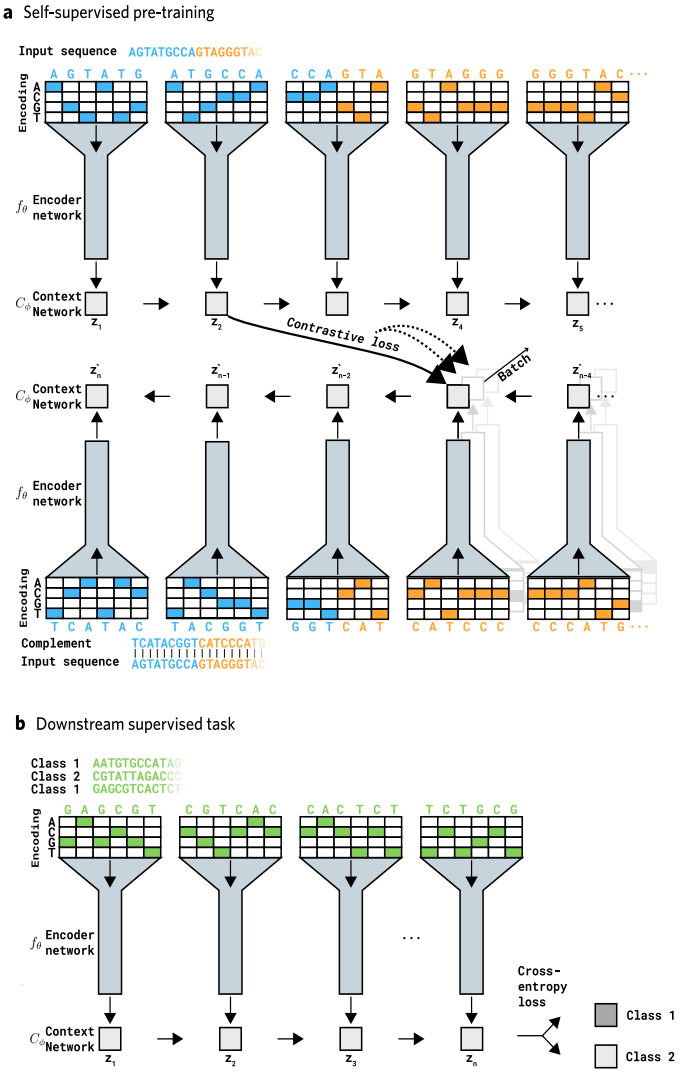


Fig. 1 Pre-training of *Self-GenomeNet* and using the learned weights on a down-stream task. a *Self-GenomeNet* takes part of a sequence as input and predicts the reverse-complement of the remaining sequence. The representations are learned by dividing unlabeled DNA sequences and their reverse-complements into patches, each of which is given as an input to an encoder network f_θ. The outputs of f_θ are then fed sequentially to a recurrent context network C_ϕ, resulting in representations of the input sequence up to a point t ($S_{1:t}$) and representations of the reverse-complement of the input sequence going from $(t+1)$ to the end (i.e., $\bar{S}_{N:t+1}$). The representations are computed for multiple values of t simultaneously. Finally, the representations of $S_{1:t}$ (z_i) and $\bar{S}_{N:t+1}$ ($z'_{(n-1-i)}$) predict each other for multiple values of t by using a contrastive loss, i.e, these sequences are matched among existing sequences in the training batch. Thus, in one iteration of the training of *Self-GenomeNet*, each of the computed representations z_i and $z'_{(n-1-i)}$ are utilized efficiently since z_i predicts $z'_{(n-1-i)}$ and $z'_{(n-1-i)}$ predicts z_i for $i \in (1, 2, .., n-2)$ in one iteration of training. In the figure, we only show that z_2 predicts $z'_{(n-3)}$ for visual simplification. **b** The weights of f_θ and C_ϕ are initialized with the training results from the self-supervised learning task, but they are further trained (fine-tuned), along with the linear layer on the new supervised task.

and C_ϕ are initialized with the training results from the SSL task, but they are further trained (fine-tuned) together with the linear layer on the new supervised task (Fig. 1b).

We evaluated the performance of the representations obtained via *Self-GenomeNet* on different benchmarks (supervised tasks) using data of either viral, bacterial, or human origin: (i) The virus dataset contains viral genomes from GenBank[22] and RefSeq[23], where the task is to classify prokaryotic viruses (bacteriophages) and eukaryotic viruses (termed "non-phages"). (ii) For bacterial data, we designed a supervised task on type VI secretion system identification (T6SS), where the task is to identify effector proteins among T6SS immunity proteins, T6SS regulators, and T6SS accessory proteins (SecReT6[24]). (iii) For the human dataset, we focus on the DeepSEA dataset[25]. The task is to classify 919

Table 1 Experimental results in terms of class-balanced accuracy performance for design choices of Self-GenomeNet.

Target Sequences		Pre-training Dataset / Dataset of Down-stream Task		
Length	Targets	Virus / Virus (1000 nt.)	Bacteria / Virus	Bacteria / T6SS
Fixed Length	Reverse-Complement	83.3	-	-
Varying Length	Reverse	87.8	76.9	70.2
Varying Length	Forward	85.8	-	-
Varying Length	Reverse-Complement	88.6	82.1	79.3

binary chromatin features such as transcription factor binding affinities, histone marks, and DNase I sensitivity.

We evaluated the performance of the representations obtained via *Self-GenomeNet* on different benchmarks (supervised tasks) using data of either viral, bacterial, or human origin: (i) The virus dataset contains viral genomes from GenBank[22] and RefSeq[23], where the task is to classify prokaryotic viruses (bacteriophages) and eukaryotic viruses (termed "non-phages"). The bacteriophage class contains approximately 1.0 billion nucleotides, the non-phage virus dataset ~0.5 billion nucleotides. (ii) For bacterial data, we designed a supervised task on type VI secretion system identification (T6SS), where the task is to identify effector proteins among T6SS immunity proteins, T6SS regulators, and T6SS accessory proteins (SecReT6[24]). This task is provided to demonstrate that our method works well on a dataset with real label scarcity, where the training set contains only 75 FASTA entries and ~0.3 million nucleotides. (iii) For the human dataset, we focus on the DeepSEA dataset[25]. It contains approximately 5 million subsequences of the human genome, with each sample containing 1000 nucleotides as input and a label vector for 919 binary chromatin features such as transcription factor binding affinities, histone marks and DNase I sensitivity. (iv) For the fungi-protozoa classification task, we downloaded DNAs of fungi and protozoa that may be pathogenic to humans from RefSeq[23]. Here, the training set contains approximately 2.7 billion nucleotides. (v) Finally, the bacteria data contains genomes from GenBank[22] and RefSeq[23], comprising ~83 billion nucleotides. It is used only for self-supervised pre-training.

In the results section, we will initially justify the choices we have made in our architectural design. First, we design an experiment to show the superiority of predicting targets of varying lengths over targets of fixed length. Then, we compare having the RC of neighboring subsequences as targets to be predicted with neighboring subsequences or their reverse. After justifying our design choices, we test *Self-GenomeNet* in data-scarce settings, where the labeled data is limited to a certain amount of the unlabeled data, and in transfer learning settings, where the pre-trained models are trained on different smaller datasets. Finally, we test *Self-GenomeNet* using the linear evaluation protocol[3,9,10,21,26–28], where the weights learned by self-supervision are frozen and thus not updated in the down-stream tasks. Here, only the fully connected layer on top of the frozen layers is trained. In these experiments, we compare *Self-GenomeNet* to four SSL baselines and the supervised baseline where the model is not pre-trained.

In all experiments except DeepSEA dataset, we report class-balanced accuracy and not precision/recall/F1 scores because these metrics put an emphasis on positive samples and also choosing a positive class. However, artificially choosing a positive class is harmful as detection of both classes holds equal importance for phage/non-phage classification and fungi/protozoa classification tasks. For the effector protein prediction task, assigning a positive class is also hard as the number of "effector protein" samples are more in both training, validation, and test set. For our experiments on the DeepSEA dataset, we opted for average PR AUC as a metric, based on the findings of Quang and Xie[29], who demonstrated that the sparsity of positive binary targets in this dataset can artificially inflate the ROC AUC and thus PR AUC is a more suitable indicator of performance.

Predicting the sequences of varying lengths improves the performance and has theoretical justifications. *Self-GenomeNet* is a genome-tailored SSL method that aims to train meaningful representations for various genomic tasks by capturing the unique properties of genomic data. However, current models, such as LMs or CPC[21], use a fixed target sequence length, i.e., the part of the input sequence they are trying to predict has a constant size (up to 50 nucleotides). As supported by experiments comparing our method with these methods (Figs. 2, 3, and 4), training with such small subsequences does not yield optimal results, which may be because nucleotides and n-mers contain less information than words in natural languages. Optimal training for genomics also requires longer sequences with higher information content. Therefore, *Self-GenomeNet* predicts sequences of varying lengths, ranging from small to the maximum length of sequences that the model can capture.

Self-GenomeNet outperforms all other SSL methods (Figs. 2, 3, and 4) that predict sequences of fixed length, which may indicate the effectiveness of having targets of different lengths. However, we design an additional experiment that closely examines the effect of having target sequences of different lengths, which is unique to our model compared to the baselines. We show that having target sequences of different lengths helps the network to learn better representations. Thus, two self-supervised models are trained on the virus dataset with a sequence length of 1000 for the phage classification task. The first model, *Self-GenomeNet*, used varying-length targets as subsequences with a length range of $x \in \{40, 60, 80, \ldots 960\}$, which predicted a subsequence with a length of $1000 - x$. The second model, also *Self-GenomeNet*, but with one modification, uses only two fixed-length sequences of 500 nt to predict each other, and all other settings are the same as the first model. Our evaluations show that predicting subsequences of varying lengths instead of fixed-length subsequences results in a considerable improvement in model accuracy. Specifically, the class-balanced accuracy on the test set increased from 83.3% to 88.6% when the weights learned through self-supervision (without using any labels) were frozen and only a fully connected layer was trained on top of these frozen layers using the same dataset as the downstream task (Table 1).

Predicting the reverse-complement of the neighbor sequence improves the performance. Most SSL methods were originally developed for NLP or CV tasks and did not consider the unique properties of genomic data. *Self-GenomeNet*, on the other hand, takes advantage of specific characteristics of genomic data by exploiting the fact that the reverse complement (RC) of a DNA sequence is also a valid DNA sequence. This allows for a symmetric construction of the SSL method, which reduces the

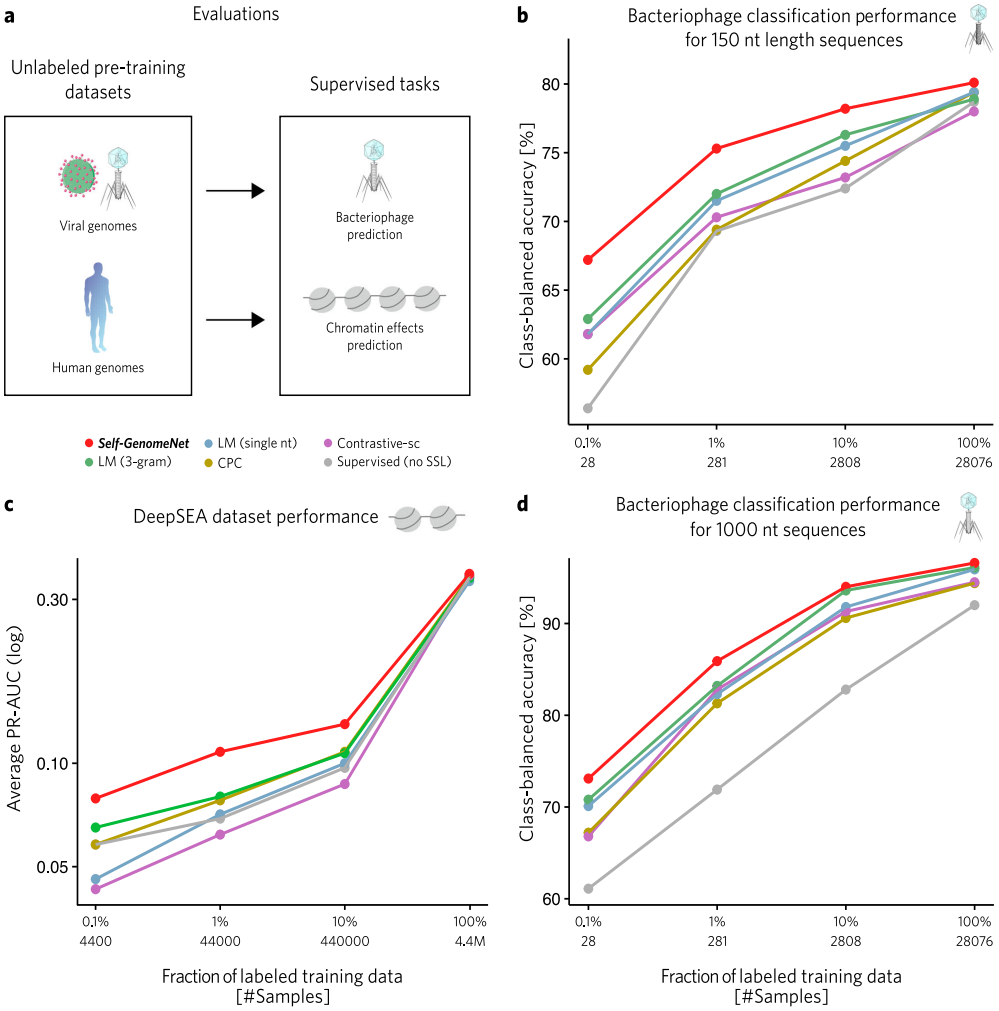


Fig. 2 Comparison of self-supervised methods in data-scarce settings. *Self-GenomeNet* representations outperform other baseline methods, such as language models[20] trained by predicting single nucleotides or 3-grams, Contrastive Predictive Coding[21], and Contrastive-sc[18], especially when a large fraction of available labels are omitted. We train the models in the datasets without using labels and then successively withhold labeled samples to mimic scenarios where labels are scarce (from 100% of available labeled samples to 0.1%). Each point in the plots is trained separately using the corresponding amount of labeled data. The weights of the context and encoder models are initialized with the training results from the SSL task, but they are trained further (fine-tuned), together with the linear layer, on the new supervised task. The label "Supervised" corresponds to the setting without any pre-training, where the weights are initialized randomly for the supervised task. **a** Overview of dataset and tasks used for evaluation. **b** The results of the viral dataset for 150 nt sequences, (**c**) the DeepSEA dataset, (**d**) and the viral dataset for 1000 nt sequences. The human icon representing the patient was created by Marcel Tisch and is available under a CC0 license. Original icon sourced from Bioicons. Twitter link for Marcel Tisch | CC0 License. The phage icon was created by DBCLS and is licensed under a CC-BY 4.0 Unported license. Modifications were made. Original icon sourced from Bioicons. DBCLS | CC-BY 4.0 License. The virus icon representing hepatitis was created by Servier and is licensed under a CC-BY 3.0 Unported license. Modifications were made. Original icon sourced from Bioicons. Servier | CC-BY 3.0 License. The chromatin structure icon was created by DBCLS and is licensed under a CC-BY 4.0 Unported license. Modifications were made. Original icon sourced from Bioicons. DBCLS | CC-BY 4.0 License.

number of model parameters and mitigates the risk of overfitting. Furthermore, we incorporate RC awareness into our learned representations by predicting RC targets.

We conducted an experiment to compare the effectiveness of different potential target sequences in self-supervised pre-training. Specifically, we compared the use of RC neighbor sequences $\bar{S}_{N:t+1}$ (which we refer to as "RC") with the use of neighbor sequences $S_{t+1:N}$ (referred to as "Forward") and the reverse of neighbor sequences $S_{N:t+1}$ (referred to as "Reverse"). We examine both settings on a viral dataset (1000 nt) using the linear evaluation protocol[3,9,10,21,26–28], meaning that we freeze the weights trained on the viral dataset without using labels and then train a linear layer on top of these weights using the labels. We find that using RC targets results in a relative class-balanced

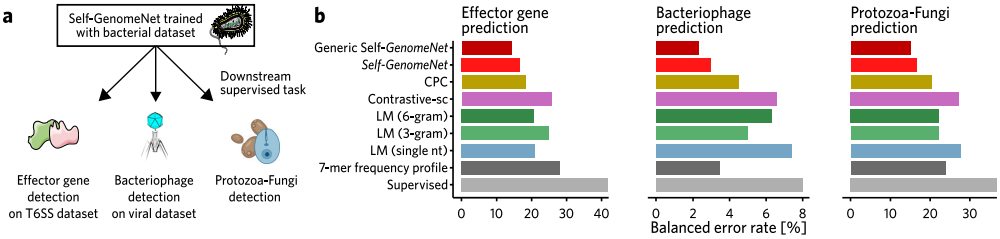


Fig. 3 Comparison of self-supervised methods for transfer learning tasks. *Self-GenomeNet* representations outperform other baseline methods, such as language models[20] trained by predicting single nucleotides, 3-grams or 6-grams, Contrastive Predictive Coding[21], and Contrastive-sc[18], when pre-trained with the bacteria dataset and then fine-tuned for effective gene detection and bacteriophage classification tasks. We also provide an additional evaluation, where we train *Self-GenomeNet* on a wider range of datasets, which includes bacteria, virus and human data (*generic Self-GenomeNet*). This model achieves even higher performance compared to *Self-GenomeNet*, showing that a wider range of data improves the performance of *Self-GenomeNet*. The context and encoder model weights are initialized with training results from the SSL task, but are further trained (fine-tuned) on the new supervised task along with an additional linear layer on top. The label "Supervised" and "7-mer frequency profile" corresponds to the setting without any pre-training, where the weights are randomly initialized for the supervised task. Here, the first model is the same architecture used in SSL settings, which similarly takes the one-hot encoded sequences. The second model is the CNN model developed by Fiannaca et al. [31], and it uses a 7-mer frequency profile as input. **a** Overview of the dataset and tasks used for evaluation. **b** The class-balanced accuracy performance for the effector gene detection task, the bacteriophage detection task, and for the protozoa-fungi prediction task. This figure was created in part with BioRender.com. The phage icon was created by DBCLS and is licensed under a CC-BY 4.0 Unported license. Modifications were made. Original icon sourced from Bioicons. DBCLS | CC-BY 4.0 License.

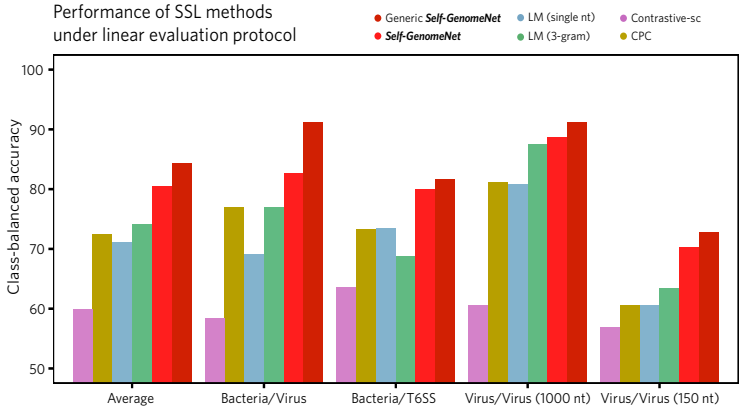


Fig. 4 Comparison of self-supervised methods using the linear evaluation protocol. Self-GenomeNet outperforms the baselines in all experiments using the linear evaluation protocol. First, the SSL methods are pre-trained on the bacteria and virus datasets. Then, the weights of the encoder and decoder networks learned in the pre-training are frozen and a linear layer on top of the model is trained on the T6SS and virus datasets. The relative increase in class-balanced accuracy over the second best-performing method is 9% on average, demonstrating the effectiveness of *Self-GenomeNet*. We also provide an additional evaluation, where we train *Self-GenomeNet* on a wider range of datasets, which includes bacteria, virus, and human data, which we call "*generic Self-GenomeNet*". This model achieves even higher performance compared to *Self-GenomeNet*, showing that a wider range of data improves the performance of *Self-GenomeNet*.

accuracy performance increase of 0.9% compared to using reverse targets and 3.3% compared to using forward targets (Table 1).

Since the difference between RC and reverse targets was small in the experiment on the virus dataset, we investigate further. In the following experiments, we freeze the self-supervised weights trained on a bacteria dataset and train a linear layer on top of these weights for downstream tasks on the viral pre-training dataset, where the task is phage and non-phage classification and T6SS effector gene prediction. Our results show that using RC targets improved relative performance by 6.8% and 13.0% compared to using reverse sequences for these tasks, respectively. Therefore, our results suggest that using RC targets leads to better performance without increasing the number of parameters (Table 1). We also discuss potential reasons for this performance change in the Discussion section.

***Self-GenomeNet* outperforms baselines in data-scarce settings, reducing the need for additional labeled data**. Generating large labeled datasets requires a substantial investment of resources that may not be feasible in computational biology. This limits the effectiveness of DL techniques. The use of unlabeled data is especially necessary when the labeled data is scarce since the accuracy of supervised DL models drops considerably in the low-data regime. We propose *Self-GenomeNet* as a data-efficient learning method to reduce the need for annotated data samples.

We mimic label scarcity scenarios by using the full datasets of virus and DeepSea datasets without labels and artificially reducing

the available labeled samples. Specifically, we test *Self-GenomeNet* by training on the full virus and DeepSEA datasets without using labels. We then quantify how model performance decreases as we successively withhold labeled samples to imitate scenarios where labels are scarce (from 100% of available labeled samples to 0.1%)[3,30]. Three different models are trained using *Self-GenomeNet* and baseline methods. Specifically, two models for 150 nt and 1000 nt long sequences are trained on virus data and one model is trained on the DeepSEA dataset. Then, each of these models is trained on data-scarce settings.

For the virus dataset in both cases of 150 nt (Fig. 2b) and 1000 nt (Fig. 2c) sequence length and the DeepSEA dataset (Fig. 2d), our method outperforms the four SSL baselines adapted from NLP or CV as well as the supervised baseline without any self-supervised training, at all evaluated fractions of available labels. We observe the most pronounced improvement in extreme data-scarce settings, such as 0.1% and 1%, with an average relative improvement of 11% and 14% over the second-best SSL method, respectively. *Self-GenomeNet* outperforms the supervised baseline that is trained with ten times more data (0.1% vs. 1%, 1% vs. 10%) on average across all experiments. This highlights that *Self-GenomeNet* representations are particularly effective in scenarios where labels are expensive to obtain—for example, in settings where genomic features must be manually validated in the lab.

Learned representations of *Self-GenomeNet* can be transferred and adapted to new tasks and datasets. We have quantified the transfer-learning capacity of representations trained on a large dataset of genome sequences to perform downstream supervised tasks on different and smaller genome datasets (Fig. 3)[3,30]. This evaluation particularly important because (i) for a given supervised downstream task, there may be little or no matching unlabeled training data (e.g., in the case of a newly discovered taxon, there may be no suitable training data available to train representations), and (ii) while performing the downstream task (training an arbitrary supervised model on top of the representations) is computationally fast, training the representation usually requires specialized hardware that is not available to many researchers.

For the transfer-learning tasks, we trained *Self-GenomeNet* and baseline models on a broad bacterial dataset. We then evaluated their predictive power on two tasks. First, we tested the transfer-learning ability of the pre-trained models on a very specific case: the effector gene prediction task, where we used the T6SS dataset[24]. The goal is to determine whether the pre-training regime works when the final supervised task is only a small subset of the self-supervised task. Next, we evaluated whether a biased training set for the pre-training task affects the prediction of the final supervised model. Here, the models (pre-)trained on the bacterial dataset (which might contain integrated prophages) are applied to the downstream tasks of separating bacteriophages from eukaryotic viruses.

Self-GenomeNet representations outperform those generated by the four competing baseline methods as well as the non-pretrained model. *Self-GenomeNet*, pre-trained on the bacteria dataset reduces the misclassification rate by 9% on the effector gene prediction task and by 33% on the phage identification task compared to the best performing SSL baseline. Compared to the non-pretrained baseline, the improvement on these tasks is as high as 60% and 63%, respectively. This shows that trained representations generalize well and are transferable to tasks with labeled data, even when this data differs from the self-supervised training data. This allows for applications where no suitable pre-training data is available.

***Self-GenomeNet* outperforms all baselines in linear evaluation method**. We have shown that the representations learned by *Self-GenomeNet* exhibit transfer-learning capacity and that *Self-GenomeNet* excels in data-scarce settings, consistently outperforming SSL baselines and supervised models over different fractions of available labels. However, all of the above evaluations include fine-tuning of the pre-trained weights (of the encoder and context networks). While the fine-tuning improves the performance in most of the cases, it makes it more difficult to evaluate the direct contribution of the SSL method due to the updated weights. To evaluate the quality of the embeddings learned by the SSL methods without making any modification on them, specifically by not fine-tuning them on downstream tasks, we use the linear evaluation protocol[3,9,10,21,26–28]. This method requires freezing the weights learned by self-supervision and thus not updating them in the downstream task, and training a fully connected layer is trained on top of the frozen layers on the downstream task. Therefore, the embeddings, which are the output of the pre-trained model, remain unchanged for a given input after the training on the downstream task. This simple and efficient method thus compares the effectiveness of SSL methods by directly comparing the embeddings themselves. In addition, training only the last linear layer is less computationally intensive than training the entire network, and achieving high performance without training the entire network may be useful for researchers with limited computational resources.

We performed this experiment and compared our method to the baselines on the virus dataset (phage classification task) for both 150 nt and 1000 nt, where the virus dataset is used for the pre-training and the downstream task. Additionally, we similarly froze the representations learned during the self-supervised pre-training on the bacteria dataset (for 1000 nt sequences) and evaluated the performance of these representations on the T6SS and virus datasets. We show that *Self-GenomeNet* outperforms the baselines in all experiments, and the relative increase in class-balanced accuracy over the second best method is 9% on average (Fig. 4).

Discussion

We introduced *Self-GenomeNet*, a SSL technique designed specifically for genomic data. By leveraging RC sequences and predicting targets of different lengths, *Self-GenomeNet* overcomes the limitations of previous SSL methods and offers a more efficient use of unannotated genomic data. In our experiments, we compare *Self-GenomeNet* with several SSL baselines. We have shown that *Self-GenomeNet* outperforms CPC[21], which is potentially the most similar SSL method to ours, since both methods predict a target subsequence with contrastive loss. It also outperforms Contrastive-sc[18] as well as LMs[20] based on predicting single nucleotides and 3-grams. Both of these were originally proposed for CV and NLP, respectively, and have been applied in several cases in bioinformatics (Supplementary Methods, Supplementary Figs. 1, 2).

We have shown that *Self-GenomeNet* can also outperform supervised baselines that take normalized k-mer frequency as input. Specifically, we compare our model to the CNN model proposed by Fiannaca et al. [31]. The input of this model is a 7-mer frequency profile—the normalized frequency of 7-mers observed in the sequence. This input is fed into the model consisting of two convolutional layers with max-pooling layers, a flattened layer, and two fully connected layers. While this model requires an additional pre-processing step (in order to create the histogram based on 7-mers) and has approximately six times the number of parameters compared to our *Self-GenomeNet* model, our approach consistently outperforms this baseline in all experiments (Virus dataset (1000 nt) for phage/non-phage classification task, on T6SS dataset) (Fig. 3). Notably, *Self-Genomenet* archives substantially superior performance, particularly in data-scarce settings (Supplementary Fig. 4).

We tested the learned representations in the data-scarce regime, with only 0.1%, 1%, and 10% of the labeled data, and showed that *Self-GenomeNet* outperforms standard supervised training with only ~10 times less labeled training data. We tested the transfer learning capability and showed that our method achieves better performance compared to other SSL methods on our datasets. Finally, we show that the use of varying target lengths (based on a theoretical explanation) and the use of RC targets improve accuracy.

The effectiveness of the *generic Self-GenomeNet* model, which incorporates pre-training on virus, bacteria, and human data, consistently outperformed models pre-trained solely on virus data or bacteria data across all tested datasets on transfer learning tasks, under the linear evaluation protocol and across all proportions of labeled in data-scarce settings (Figs. 3, 4 and Supplementary Fig. 3). These results confirm the expected advantage of a network that is pre-trained on multiple sources of data compared to a network pre-trained on a single source. This aligns with the fundamental principles of ML, where leveraging diverse pre-training data often leads to improved performance.

Our study reveals an adaptation in SSL techniques that makes them well-suited for analysis of genomic datasets, allowing for more efficient use of genomic data. One adaptation is the use of the RC for data processing. Typically, DNA sequences are read from one end, but genes can be located on either strand of the DNA molecule. By accounting for RCs, *Self-GenomeNet* can efficiently learn powerful representations using the symmetry in the design. Processing both sequences with the same ML model and evaluating the average of the model's decisions in order to predict regulatory and taxonomic features is observed in several models in supervised training[32,33]. Therefore, feeding these two sequences to the same model such as a CNN and RNN model is a well-established practice for supervised learning tasks. However, the goal in these tasks is not to learn self-supervised representations, unlike our method, which to the best of our knowledge is the first method to use RC to make an SSL method more effective and efficient. On the other hand, these works justify the weight-sharing strategy in the encoder and context networks of our architecture.

Our study shows that using shared weights for the reciprocal prediction of two sequences, both of which are the RC of the upcoming subsequences for each other, improves the overall performance. While *Self-GenomeNet* predicts the representation of the RC of a neighboring subsequence $\bar{S}_{N:t+1}$ from $S_{1:t}$, the symmetry of the setup allows for also predicting $S_{1:t}$ from $\bar{S}_{N:t+1}$ and using shared weights for the whole model (prediction network, context network, and encoder network). Importantly, among the subsequence pairs that we can consider and study, only the subsequence pair $S_{1:t}$ and $\bar{S}_{N:t+1}$ can use the same encoder network, context network, and prediction network to predict each other, ensuring that the ML network performs the same task in both predictions. Specifically, we use these networks to predict the RC of the upcoming data for both predicting $S_{1:t}$ using $\bar{S}_{N:t+1}$ and predicting $\bar{S}_{N:t+1}$ using $S_{1:t}$. For other subsequence pairs we considered, which are $S_{1:t}$ and neighbor sequences $S_{t+1:N}$ (referred to as "Forward"), and $S_{1:t}$ and the reverse of neighbor sequences $S_{N:t+1}$ (referred to as "Reverse"), these networks do not have the same task, which results in a decrease in performance. Specifically, in the "Reverse" condition, context network with shared weights read these subsequences $S_{1:t}$ and $S_{N:t+1}$ in opposite directions and in the "Forward" condition the prediction networks with shared weights predict upcoming neighboring sequence when $S_{1:t}$ predicts $S_{t+1:N}$ and past neighboring subsequence when $S_{t+1:N}$ predicts $S_{1:t}$. The use of shared weights, which is possible by our proposed strategy of using the RC of the neighboring subsequences to predict each other, reduces the number of learned parameters and the risk of overfitting and thus improves the performance.

Self-GenomeNet's ability to reduce computation time by exploiting symmetry and RC is important for genomic research, where large datasets must be explored to gain insight into complex biological systems. *Self-GenomeNet* allows for efficient training by generating representations of multiple subsequence pairs simultaneously. Specifically, the representations of $S_{1:t}$ and $\bar{S}_{N:t+1}$, are computed for multiple values of t in a single iteration. This is done by feeding both the input sequence $S_{1:N} = [s_1, s_2, \dots, s_N]$ and the RC of that input $\bar{S}_{N:1} = [\bar{s}_N, \bar{s}_{N-1}, \dots, \bar{s}_1]$ into the network, where $s_i \in \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$ and $\bar{s}_i$ is the complementary nucleotide, e.g., $\bar{\mathrm{A}} = \mathrm{T}$. The network then evaluates representations of $S_{1:t}$ and $\bar{S}_{N:t+1}$ for multiple values of t by design. We use all matching (neighbor) representations as pairs for self-supervised training. This leads to considerable efficiency in self-supervised pre-training. For example, in our experiments, there are 18 and 47 matching representations per each data sample in the batch respectively for 150 nt and 1000 nt sequences, respectively. Additionally, the number of predictions in an iteration is even double these values because the matching representations predict each other. All of these predictions are then used to optimize the model in one iteration (36 and 94 predictions, respectively) instead of having only one prediction per data sample in the batch, as is common with several other methods[3,18]. This makes *Self-GenomeNet* a computationally efficient SSL method.

Long short-term memory (LSTM) layers, which we use in our context network, are known to be less effective when they are fed inputs that contain much more time steps than 100[34]. Considering this, we designed our architectures to have 49 and 22 time steps fed into the context network, for our 1000 nt and 150 nt models, respectively. Specifically, we reduced the number of time steps by having a distance between the initial nucleotides of the created patches (Fig. 1) to be 20 and 6 respectively for these models. Having these values greater than 1 reduces the number of time steps considerably and using even greater values for this distance is recommended to be used for sequences that are much longer than 1000 nt. Therefore fairly limited short-term memory of LSTM can be managed. Additionally, it is also possible to change LSTM altogether with transformer-based models, which we will evaluate in the next version.

Self-GenomeNet has been shown to outperform other SSL methods in experiments with sequences of 150 and 1000 nt input lengths, demonstrating its effectiveness for sequences of varying lengths. Furthermore, *Self-GenomeNet* can be used to learn representations of sequences even longer than 1000 nt. However, pre-trained models that are trained on read-level sequences may not be effective for considerably longer sequences. Therefore, it may be necessary to pre-train a new model using *Self-GenomeNet* with longer sequences. It should be noted that training models on very long sequences can require a substantial amount of memory, making it difficult to fit many samples on a GPU. To ensure high batch size values, methods such as batch accumulation should be used instead of using a very small batch size (~10), which can negatively impact the effectiveness of self-supervised training. Therefore, practitioners should consider using batch accumulation when working with long sequences. Additionally, we suggest being cautious when interpreting results or masking low information sequences when the dataset contains a high repeat content such as transposable elements.

In our experiments, *Self-GenomeNet* showed resilience to changing architectural hyperparameters. Specifically, several architectural hyperparameters differ in the experiments with input length values of 150 nt and 1000 nt on the virus data, such as kernel and stride values of the convolutional layer. Despite the

different choices of hyperparameters, consistent improvements over other SSL methods are observed in both experiments, indicating the resilience of *Self-GenomeNet* to architectural changes. Therefore, we expect the performance of *Self-GenomeNet* to be superior to the other SSL methods for different model architectures. In addition, the performance of *Self-GenomeNet* can potentially be further enhanced by using different architectural modifications, such as deeper networks or alternative models for the encoder and context network, which we will evaluate in the next version.

In summary, our study shows that *Self-GenomeNet* can learn powerful representations from genomic datasets and can potentially be used to improve models trained on nucleotide-level data. Due to the improved performance in label-scarce settings, this method is of particular interest for the development of ML models, where the generation of labeled data is costly and only a limited number of labels are available. This could enable a new class of ML methods for domains such as functional prediction of genes, phenotypic or taxonomic prediction of genomes, or detection of loci of interest, such as pathogenicity islands.

Methods

Self-supervised training and contrastive loss in *Self-GenomeNet*. To create the representation, the sequence is first divided into P overlapping patches $S_{p(j)}$ with the range $p(j) = (j \cdot a + 1) : (j \cdot a + l)$ indexing a subsequence, where l is the patch length and a the patch stride value, and $a < l$ for overlapping patches. The patches are first encoded using a convolutional neural network $f_\theta(\cdot)$. The resulting sequence of vectors $[f_\theta(S_{p(0)}), f_\theta(S_{p(1)}), \ldots, f_\theta(S_{p(P-1)})]$ is fed into a recurrent context network $C_\phi(\cdot)$, yielding embeddings $z_i = C_\phi(\{f_\theta(S_{p(u)})\}_{u \le (i-(l/a))})$ for $(l/a) \le i < P + (l/a)$. The patches of the RC $\bar{S}_{\bar{p}(j)} = [\bar{s}_{j \cdot a+l}, \ldots, \bar{s}_{j \cdot a+1}]$ are also fed into f_θ first and then C_ϕ, giving rise to z_i is then a representation of $S_{1:i \cdot a}$ likewise, $\bar{z}_i$ represents $\bar{S}_{N:(i \cdot a+1)}$.

Training the encoder and context networks consists of predicting $\bar{z}_i$ from z_i contrastively against corresponding embeddings from other, negative example sequences $S^{(k)-}$, i.e., against $\bar{z}_i^{(k)-}$. This is done using a linear prediction layer q_η and the Noise Contrastive Estimation or InfoNCE loss[35], which maximizes the mutual information shared between the forward sequence and its matching RC sequence, is used:

$$L_i = -\log \frac{\exp\left(\bar{z}_i^T q_\eta(z_i)\right)}{\exp\left(\bar{z}_i^T q_\eta(z_i)\right) + \sum_k \exp\left(\left(\bar{z}_i^{(k)-}\right)^T q_\eta(z_i)\right)} \quad (1)$$

Negative samples $\bar{z}_i^{(k)-}$ are efficiently generated by comparing against representations of other sequences loaded in the same mini-batch. Each sequence in the minibatch produces two negative samples for other sequences, the sequences themselves and their RC, resulting in $2(B-1)$ negative samples when using the minibatch size B.

Embeddings are always contrasted only against embeddings of sequences of the same length as the positive sample. More specifically, $z_i(S^+)$ predicts $\bar{z}_i(S^+)$ against $\bar{z}_i(S^{(k)-})$ where $S^{(k)-}$ contains $2(B-1)$ negative samples (other samples in the batch and their RCs) and not against $\bar{z}_{i+n}(S^-)$ where $n \neq 0$. This is done to prevent the network from learning to encode the represented length directly to gain an advantage, which would not be an intrinsically interesting feature for downstream tasks.

A loss term is introduced for each index i, denoting the number of patches represented by z_i. Due to the symmetry of the setup, the model both predicts $\bar{z}_i$ from z_i, as well as z_i from $\bar{z}_i$. The corresponding loss $\bar{L}_i$, induced by predicting z_i from $\bar{z}_i$ then uses negative examples with the same length as z_i. The final loss for each individual sequence S is thus defined by $L = \sum_i (L_i + \bar{L}_i)$.

Self-GenomeNet maximizes the mutual information between varying-length targets and the representations, allowing the representations to effectively learn both short- and long-term information. Our theoretical analysis illustrates the advantages of predicting sequences of different lengths. The mutual information between the predicted subsequences and the learned representations is maximized during self-supervised training[21]. However, optimizing the mutual information only for sequences that should contain limited long-range information may reduce the effectiveness of the learned representations because they may not capture important long-range information. Therefore, we propose a self-supervision method that maximizes the lower bound of the mutual information between the embeddings z_i and varying-length RC targets $\bar{S}_{N:(i \cdot a+1)}$ for $i \in \{(l/a), ((l/a+1), \ldots, ((N-l)/a)\}$, where l is the length of each patch that is fed into the convolutional encoder network and a the patch stride length, and $a < l$ for overlapping patches. Using the theoretical proof of CPC[21], we derive the theoretical derivation of the lower bound for maximizing the mutual information as follows: $I(\bar{S}_{N:(i \cdot a+1)}, z_i) \ge \log(n) - L_i$, where $I(\bar{S}_{N:(i \cdot a+1)}, z_i)$ is the mutual information between the learned representation z_i and the RC of the consecutive subsequence $\bar{S}_{N:(i \cdot a+1)}$. n is the number of samples in the contrastive pre-training and is therefore $2(B-1)+1 = 2B-1$. The length of the predicted subsequence $\bar{S}_{N:(i \cdot a+1)}$ changes as the value of i changes, thereby allowing us to maximize the mutual information between targets of varying of length and the learned representations by optimizing the loss L_i for different values of i simultaneously. Moreover, this approach captures both short- and long-range semantics, and avoids the potential loss of long-range information that can occur when maximizing mutual information between learned representations and short patches that lack long-range information.

The theoretical derivation of the lower bound for maximizing the mutual information between targets of different lengths and the learned representations can be shown by adapting the equations in CPC[21] as follows:

Given that $\bar{S}_{N:(i \cdot a+1)}$ is predicted from the representation z_i, our loss is given by Eq. (2).

$$L_i = -\mathbb{E}_S \left[\log \frac{f\left(\bar{S}_{N:(i \cdot a+1)}, z_i\right)}{\sum_{S_m \in S} f\left(S_m, z_i\right)} \right] \quad (2)$$

where S includes $S^{(k)-}$(the set containing all negative samples of the contrastive loss) and $\bar{S}_{N:(i \cdot a+1)}$. f is the modeled density ratio and is given by Eq. (3).

$$f\left(\bar{S}_{N:(i \cdot a+1)}, z_i\right) = \exp\left(\bar{z}_i q_\eta(z_i)\right) \propto \frac{p\left(\bar{S}_{N:(i \cdot a+1)}, |, z_i\right)}{p\left(\bar{S}_{N:(i \cdot a+1)}\right)} \quad (3)$$

The optimal loss and a lower bound for this loss are then given by Eqs. (4) and (7) respectively.

$$L_i^{opt} = -\mathbb{E}_S \log \left[\frac{\frac{p\left(\bar{S}_{N:(i \cdot a+1)}|z_i\right)}{p\left(\bar{S}_{N:(i \cdot a+1)}\right)}}{\frac{p\left(\bar{S}_{N:(i \cdot a+1)}|z_i\right)}{p\left(\bar{S}_{N:(i \cdot a+1)}\right)} + \sum_{S_m \in S^{(k)-}} \frac{p\left(S_m|z_i\right)}{p\left(S_m\right)}} \right] \quad (4)$$

$$= \mathbb{E}_S \log \left[1 + \frac{p\left(\bar{S}_{N:(i \cdot a+1)}\right)}{p\left(\bar{S}_{N:(i \cdot a+1)}|z_i\right)} \sum_{S_m \in S^{(k)-}} \frac{p\left(S_m|z_i\right)}{p\left(S_m\right)} \right] \quad (5)$$

$$\approx \mathbb{E}_S \log\left[1 + \frac{p\left(\bar{S}_{N:(i\cdot a+1)}\right)}{p\left(\bar{S}_{N:(i\cdot a+1)}|z_i\right)}(n-1)\mathbb{E}_{S_m}\frac{p(S_m|z_i)}{p(S_m)}\right] \quad (6)$$

$$= \mathbb{E}_S \log\left[1 + \frac{p\left(\bar{S}_{N:(i\cdot a+1)}\right)}{p\left(\bar{S}_{N:(i\cdot a+1)}|z_i\right)}(n-1)\right] \geq \mathbb{E}_S \log\left[\frac{p\left(\bar{S}_{N:(i\cdot a+1)}\right)}{p\left(\bar{S}_{N:(i\cdot a+1)}|z_i\right)}n\right] \quad (7)$$

This is equal to $-I(\bar{S}_{N:(i\cdot a+1)}, z_i) + \log(n)$. Finally, we show that $I(\bar{S}_{N:(i\cdot a+1)}, z_i) \geq \log(n) - L_i$, where $I(\bar{S}_{N:(i\cdot a+1)}, z_i)$ is the mutual information between the learned representation z_i and the RC of the consecutive subsequence $\bar{S}_{N:(i\cdot a+1)}$. n is the number of samples in the contrastive pre-training and is therefore $2(B-1)+1 = 2B-1$. As the value of i changes, the length of the predicted subsequence $\bar{S}_{N:(i\cdot a+1)}$ changes. We therefore maximize the mutual information between varying-length targets and the learned representations by optimizing the loss L_i for different values of i and simultaneously. Thus, the representations learn both short and long-range semantics. The higher performance of *Self-GenomeNet* compared to the baselines indicates that our strategy of maximizing the mutual information between varying-length subsequences and the learned representations is effective.

Network architecture design. The particular architectures of the encoder network f_Θ and the context network C_ϕ are hyperparameters of the method and can be chosen according to the task at hand. We choose f_Θ to be a convolutional layer with 1024 filters and C_ϕ to be an LSTM layer with 512 units. The kernel size of the convolutional layer is set equal to the patch size, which is the number of nucleotides given to the encoder network (Fig. 1a), and the stride value of the convolutional layer is equal to the distance between the starting points of the patches. For experiments trained on 150 nt sequences, the patch size is set to 24, and the stride is set to 6, resulting in 75% overlapping patches. For experiments trained on 1000 nt sequences, the patch size is set to 40 and the stride to 20, resulting in 50% overlapping patches.

Model training process. We use the Adam optimizer[36] with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and a learning rate of 0.0001 for all experiments, except for the T6SS dataset fixed base network under transfer learning protocol, where the learning rate is set to 0.001 because we observe that 0.0001 is too low for this experiment. For weight initialization, Glorot uniform initialization[37] is chosen, which is the default Keras weight initialization. The size of the minibatch is chosen to be the largest possible for the used GPU, GeForce RTX 2080 Ti. Therefore, it is set to 128 for the self-supervised pre-training and 2048 for the supervised downstream tasks (only powers of 2 are considered). The hyperparameters, such as the hyperparameters of Adam[36] or learning rate, are set to the same values as our method for all baseline experiments. When some hyperparameters are unique to a baseline method, we follow the recommended values as in their papers.

In the transfer-learning experiments on the T6SS dataset and in the experiments with data-scarce settings, where 0.1% of the dataset is available, only the last linear layer of the model is trained with the labels as the first round of supervised training on downstream tasks. This means that the pre-trained layers are frozen at this stage, which is done to avoid rapid overfitting to the small labeled datasets, which results in low performance on the validation set. Then, in the second round of supervised training, the frozen layers are also fine-tuned, typically with considerably fewer iterations than in the first round due to quick overfitting. In other transfer-learning and data-scarce experiments, the initial training with frozen layers is skipped for a faster evaluation process, as the preliminary experiments showed that it did not contribute to the final performance. Thus, the entire network is fine-tuned directly.

In the experiments in which the raw input data are fasta files (all datasets except DeepSEA), as long as the existing unlabeled data files are long enough, we generate sequences up to a certain number from the same fasta file for both supervised and self-supervised training. Thus, not only one data sample is generated when the fasta file is opened, as it would be if generated samples were completely random, in order to ensure much faster preprocessing. Specifically, up to 512 samples are created for fungi-protozoa dataset due to very long fasta files (and thus longer processing time) and 64 for other experiments. Additionally, this may help to create harder negative samples in the contrastive self-supervised training, which is shown to be helpful for learning better representations. However, hard-negative mining is not explicitly enforced in our experiments, such as by modifying the loss function[38]. While incorporating such measures can further improve the performance of *Self-GenomeNet*, the fact that we achieved robust results without relying on these underscores the robustness and success of our approach.

Datasets. The *DeepSEA* dataset[25] is an open benchmark dataset that has been evaluated by many other DL models[25,29,39]. It contains approximately 5 million subsequences of the human genome, with each sample containing 1000 nucleotides as input and a label vector for 919 binary chromatin features such as transcription factor binding affinities, histone marks, and DNase I sensitivity.

The *Virus* dataset is used as a representative of taxonomic classification tasks often encountered in metagenomics, where DNA found in environmental samples is analyzed by next generation sequencing[40]. We downloaded all publicly available viral genomes from GenBank[22] and RefSeq[23], and divided the dataset into two taxonomic classes of bacteriophages vs. viruses that are not bacteriophages, based on the annotations provided. Unlike DeepSEA, which identifies properties of genomic regions, this task tries to differentiate an aspect of an entire given genome sequence. We divided the downloaded FASTA files into training, validation, and test sets in approximate proportions of 70%, 20%, and 10%, respectively. The bacteriophage class contained approximately 1.0 billion nucleotides, the non-phage virus dataset ~0.5 billion nucleotides. Samples were created from FASTA files by partitioning them into equal-length non-intersecting sequences.

The *bacterial* dataset contains bacterial genomes from GenBank[22] and RefSeq[23], comprising approximately 83 billion nucleotides. It is used only for self-supervised pre-training. To create this dataset, we downloaded all publicly available bacteria genomes from GenBank, comprising approximately 83 billion nucleotides, and processed them similarly to how we processed the Virus dataset.

The *T6SS* effector protein dataset is provided to demonstrate that our method works well on a dataset with real label scarcity, where the training set contains only 75 FASTA entries. It is based on publicly available bacteria data (SecReT6[24]) where we defined the task as the identification of effector proteins. T6SS effector proteins serve as the positive samples to identify, whereas T6SS immunity proteins, T6SS regulators, and T6SS accessory proteins are negative samples. We divided the training, validation, and test sets into approximate proportions of 60%, 20%, and 20%, respectively.

For the fungi-protozoa classification task, we downloaded nucleotide data of fungi and protozoa that may be pathogenic to humans from RefSeq[23] using the genome_updater.sh script from https://github.com/pirovc/genome_updater with the parameters -g "fungi" -d "RefSeq" -c "representative genome" -A species:1 -a -p -T '4930,74721,4753,4827,5052,5475,5206,33183,5042,5151, 34487,4859' -k. For protozoa, we downloaded nucleotide information with the same script with the parameters -g "protozoa" -d "RefSeq" -c "representative genome" -m -A species:1 -a -p -T '554915,255975,5878,5794'. We divided the downloaded FASTA files into training, validation, and test sets in approximate proportions of 70%, 20%, and 10%, respectively.

Reporting summary. Further information on research design is available in the Nature Portfolio Reporting Summary linked to this article.

Data availability

The data we used are publicly available and information about the data used is explained in the Datasets subsection. In addition, the training, validation and test sets can be found separately either as FASTA or RDS files, or as accession IDs on self.genomenet.de. Source data for figures can be found in Supplementary Data 1.

Code availability

Code to reproduce and apply the models and pre-trained models are available via interactive notebooks under self.genomenet.de.

Received: 8 February 2023; Accepted: 1 September 2023;
Published online: 11 September 2023

References

1. Gligorijević, V. et al. Structure-based protein function prediction using graph convolutional networks. *Nat. Commun.* **12**, 3168 (2021).
2. Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proc. of Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* Vol. 1 (*Long and Short Papers)* 4171–4186 (Association for Computational Linguistics, 2019).
3. Chen, T., Kornblith, S., Norouzi, M. & Hinton, G. A simple framework for contrastive learning of visual representations. In *Proc. 37th International Conference on Machine Learning* 1597–1607 (JMLR.org, 2020).
4. Zaheer, M. et al. Big bird: Transformers for longer sequences. In *Advances in Neural Information Processing Systems (NeurIPS)* Vol 33, 17283–17297 (2020).
5. Vaswani, A. et al. Attention is all you need. In *Advances in Neural Information Processing Systems* (eds Guyon, I. et al.) Vol. 30 5998–6008 (2017).
6. Beltagy, I., Peters, M. E. & Cohan, A. Longformer: the long-document transformer. Preprint at https://arxiv.org/abs/2004.05150 (2020).
7. Kitaev, N., Kaiser, Ł. & Levskaya, A. Reformer: the efficient transformer. In *Proc. 8th International Conference on Learning Representations* 1–12 (2020).
8. Dai, Z. et al. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics* 2978–2988 (2019).
9. Zbontar, J., Jing, L., Misra, I., LeCun, Y. & Deny, S. Barlow twins: self-supervised learning via redundancy reduction. In *Proc. 38th International Conference on Machine Learning* (eds. Meila, M. & Zhang, T.) Vol. 139 12310–12320 (PMLR, 18–24 Jul 2021).
10. Chen, X., Xie, S. & He, K. An empirical study of training self-supervised vision transformers. In *Proc. IEEE/CVF International Conference on Computer Vision (ICCV)* 9640–9649 (IEEE, 2021).
11. Wang, X., Zhang, R., Shen, C., Kong, T. & Li, L. Dense contrastive learning for self-supervised visual pre-training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* 3024–3033 (2021)
12. Aakur, S. N. et al. Metagenome2Vec: Building Contextualized Representations for Scalable Metagenome Analysis. In *2021 International Conference on Data Mining Workshops (ICDMW)* 500–507 (IEEE, 2021).
13. Indla, V. et al. Sim2Real for Metagenomes: accelerating animal diagnostics with adversarial co-training. in *Advances In Knowledge Discovery and Data Mining* 164–175 (Springer International Publishing, 2021).
14. Aakur, S. N. et al. Mg-net: Leveraging pseudo-imaging for multi-modal metagenome analysis. In *International Conference on Medical Image Computing and Computer-Assisted Intervention* 592–602 (Springer, 2021).
15. Queyrel, M., Prifti, E., Templier, A. & Zucker, J.-D. Towards end-to-end disease prediction from raw metagenomic data. *bioRxiv* 2020.10.29.360297 https://doi.org/10.1101/2020.10.29.360297 (2021).
16. Rives, A. et al. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proc. Natl. Acad. Sci. USA*. **118** (2021).
17. Lu, A. X., Zhang, H., Ghassemi, M. & Moses, A. Self-supervised contrastive learning of protein representations by mutual information maximization. Preprint at *bioRxiv* https://doi.org/10.1101/2020.09.04.283929 (2020).
18. Ciortan, M. & Defrance, M. Contrastive self-supervised clustering of scRNA-seq data. *BMC Bioinforma.* **22**, 280 (2021).
19. Ji, Y., Zhou, Z., Liu, H. & Davuluri, R. V. DNABERT: pre-trained bidirectional encoder representations from transformers model for DNA-language in genome. *Bioinformatics* https://doi.org/10.1093/bioinformatics/btab083 (2021).
20. Dai, A. M. & Le, Q. V. Semi-supervised sequence learning. In *Advances in Neural Information Processing Systems*, Vol 28 (eds Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M. & Garnett, R.) 3079–3087 (Curran Associates, Inc. Red Hook, NY, 2015).
21. Oord, A. V., Li, Y. & Vinyals, O. Representation learning with contrastive predictive coding. Preprint at https://doi.org/10.48550/arXiv.1807.03748 (2018).
22. Clark, K., Karsch-Mizrachi, I., Lipman, D. J., Ostell, J. & Sayers, E. W. GenBank. *Nucleic. Acids Res.* **44**, D67–D72 (2016).
23. O'Leary, N. A. et al. Reference sequence (RefSeq) database at NCBI: current status, taxonomic expansion, and functional annotation. *Nucleic. Acids Res.* **44**, D733–D745 (2016).
24. Li, J. et al. SecReT6: a web-based resource for type VI secretion systems found in bacteria. *Environ. Microbiol.* **17**, 2196–2202 (2015).
25. Zhou, J. & Troyanskaya, O. G. Predicting effects of noncoding variants with deep learning-based sequence model. *Nat. Methods* **12**, 931–934 (2015).
26. Kolesnikov, A., Zhai, X. & Beyer, L. Revisiting self-supervised visual representation learning. In *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition* 1920–1929 (IEEE, 2019).
27. Zhang, R., Isola, P. & Efros, A. A. Colorful image colorization. In *Computer Vision – ECCV 2016* 649–666 (Springer International Publishing, 2016).
28. Bachman, P., Hjelm, R. D. & Buchwalter, W. Learning representations by maximizing mutual information across views. In *Advances in Neural Information Processing Systems* (eds Wallach, H. et al.) 15509–15519 (NIPS, 2019).
29. Quang, D. & Xie, X. DanQ: a hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences. *Nucleic. Acids Res.* **44**, e107 (2016).
30. Henaff, O. Data-efficient image recognition with contrastive predictive coding. In *Proc. 37th International Conference on Machine Learning* (eds. Iii, H. D. & Singh, A.) Vol. 119, 4182–4192 (PMLR, 13–18 Jul 2020).
31. Fiannaca, A. et al. Deep learning models for bacteria taxonomic classification of metagenomic data. *BMC Bioinforma.* **19**, 198 (2018).
32. Quang, D. & Xie, X. FactorNet: A deep learning framework for predicting cell type specific transcription factor binding from nucleotide-resolution sequential data. *Methods* **166**, 40–47 (2019).
33. Shrikumar, A., Greenside, P. & Kundaje, A. Reverse-complement parameter sharing improves deep learning models for genomics. Preprint at https://www.biorxiv.org/content/early/2017/01/27/103663 (2017).
34. Géron, A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. ('O'Reilly Media, Inc., 2022).
35. Gutmann, M. U. & Hyvarinen, A. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. https://www.jmlr.org/papers/volume13/gutmann12a/gutmann12a.pdf (2012).
36. Kingma, D. P. & Ba, J. L. Adam: a method for stochastic optimization. Preprint at https://arxiv.org/abs/1412.6980 (2015).
37. Glorot, X. & Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (eds. Teh, Y. W. & Titterington, M.) Vol. 9, 249–256 (PMLR, 13–15 May 2010).
38. Robinson, J. D., Chuang, C., Sra, S., Jegelka, S. Contrastive learning with hard negative samples. In *International Conference on Learning Representations (ICLR)* (2021).
39. Kelley, D. R. et al. Sequential regulatory activity prediction across chromosomes with convolutional neural networks. *Genome. Res.* **28**, 739–750 (2018).
40. Pust, M.-M. & Tümmler, B. Identification of core and rare species in metagenome samples based on shotgun metagenomic sequencing, Fourier transforms and spectral comparisons. *ISME Commun.* **1**, 1–4 (2021).

Acknowledgements
We thank Curtis Huttenhower and Eric A. Franzosa for helpful discussions. This work was funded in part by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A and under GenomeNet Grant No. 031L0199A/031L0199B. P.C.M. received funding from the German Research Foundation (Grant number 405892038). X.-Y.T. received funding from the German Center for Infection Research (DZIF) TI BBD.

Author contributions
The main idea of *Self-GenomeNet* is proposed by H.A.G. He developed and implemented the algorithm and he prepared the experimental results. X.-Y.T. has contributed to code development, mostly for the CPC baseline. R.M. also contributed to the code development, particularly regarding the processing of the data and language model baselines. M.B., B.B., and A.C.M. contributed to the writing of the manuscript. M.B. additionally contributed to the supervision of the project. The project is mainly supervised by M.R. and P.C.M. from a machine learning and bioinformatics perspective, respectively.

Funding
Open Access funding enabled and organized by Projekt DEAL.

Competing interests
The authors declare no competing interests.

Additional information
Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s42003-023-05310-2.

Correspondence and requests for materials should be addressed to Philipp C. Münch or Mina Rezaei.

Peer review information *Communications Biology* thanks Sathyanarayanan Aakur and the other, anonymous, reviewer(s) for their contribution to the peer review of this work. Primary Handling Editor: Gene Chong. A peer review file is available.

Reprints and permission information is available at http://www.nature.com/reprints

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

© The Author(s) 2023

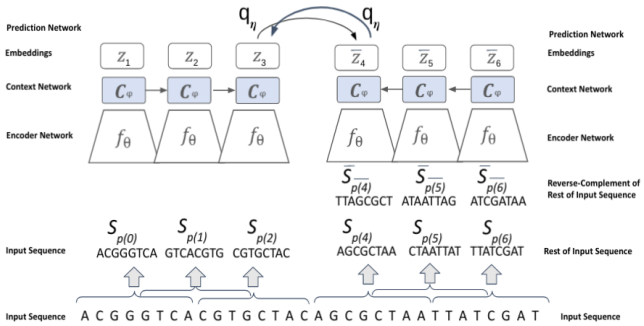
Supplementary Information

Supplementary Results

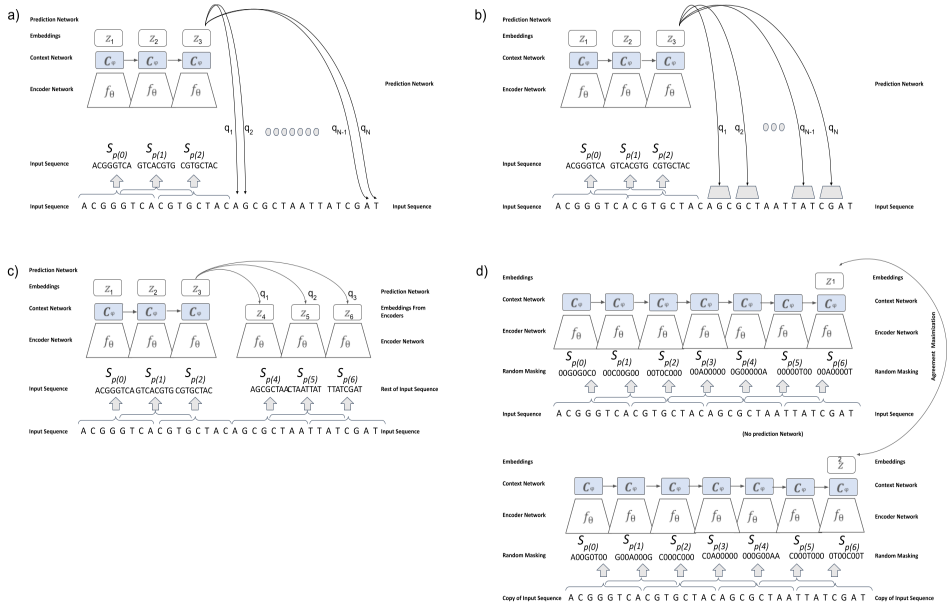
Comparison of Self-GenomeNet to baselines

The self-supervised baselines and our method are trained without using any labels (**Supplementary Fig. 1, Supplementary Fig. 2**). These models are then used for the downstream tasks. In the "supervised" baseline, there is no self-supervised pre-training, and randomly initialized weights are trained for downstream tasks.

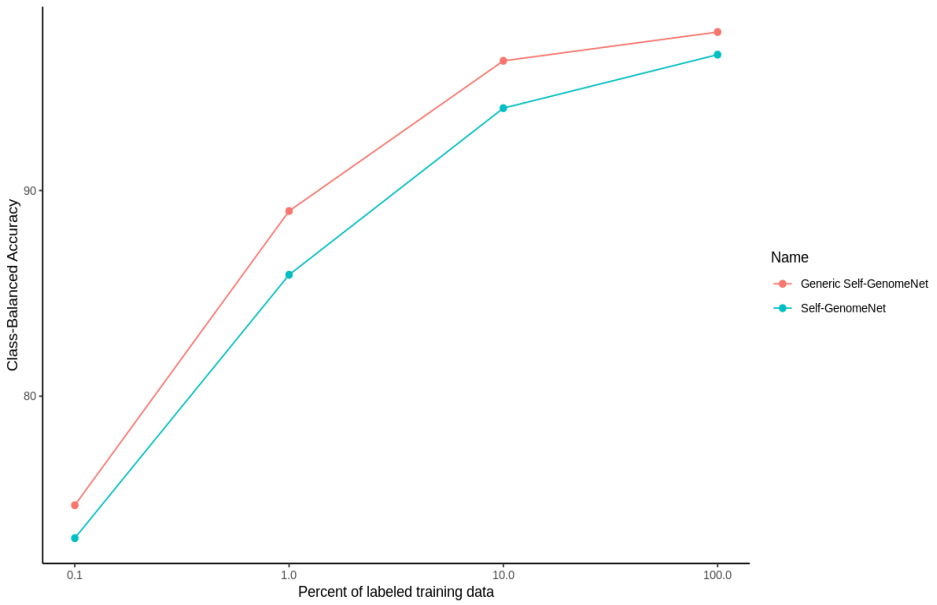
Supplementary Figures



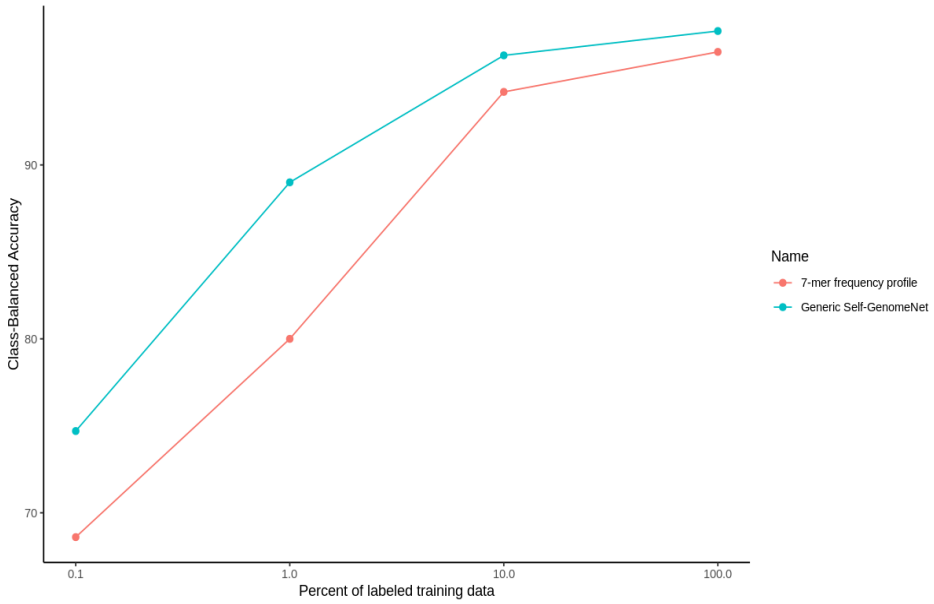
Supplementary Figure 1: Self-GenomeNet. The prediction network is used to predict representations of the reverse-complement of the neighboring sequence. The predicted sequences are of different lengths in *Self-GenomeNet*. A contrastive loss is used, which means that the matching sequence is predicted among other non-matching sequences in the batch.



Supplementary Figure 2: Self-supervised baselines. a) Language Model - single nucleotide prediction (denoted as LM - single nt)[1], where each nucleotide is one-hot encoded separately as a vector of 4 and predicted separately using cross-entropy loss. b) Language Model - 3-gram prediction (LM - 3-gram), where each 3-gram (a group of 3 nucleotides) is one-hot encoded separately as a vector of 64 and predicted separately using cross-entropy loss. c) In Contrastive Predictive Coding[2,3], the representations of the short and non-changing length neighboring sequences are predicted (in the figure below, these sequences contain 8 nucleotides) using contrastive loss, which means that the matching sequence is predicted among other non-matching sequences in the batch. The representations of the predicted sequences are obtained by feeding them into the encoder network. d) Contrastive-sc[4] takes a sequence and a copy of the sequence and randomly masks the input sequence. Then, the agreement between the embeddings of two masked copies is maximized using contrastive loss, which means that the copied sequences predict each other among other mismatching copied sequences in the batch.



Supplementary Figure 3: Data-scarce settings performance comparison on the virus dataset. Generic Self-GenomeNet and Self-GenomeNet trained only on this dataset are evaluated. The generic *Self-GenomeNet*, which incorporates pre-training on virus, bacteria, and human data consistently outperformed the model pre-trained solely on virus data (the whole dataset) across all proportions of labeled data-scarce settings. We train the models without using labels and then successively withhold labeled samples of the viral dataset for 1,000 nt sequences to mimic scenarios where labels are scarce (from 100% of available labeled samples to 0.1%). Each point in the plots is trained separately using the corresponding amount of labeled data. The weights of the context and encoder models are initialized with the training results from the SSL task, but they are trained further (fine-tuned), together with the linear layer, on the new supervised task.



Supplementary Figure 4: Data-scarce Settings Performances of generic Self-GenomeNet and a supervised model that takes 7-mer profile as input. The generic *Self-GenomeNet* outperforms the CNN model proposed by Fiannaca et al[5]. *The input of this model is a 7-mer frequency - the normalized frequency of 7-mers observed in the sequence. While this model requires an additional pre-processing step (in order to create the histogram based on 7-mers) and has approximately six times the number of parameters compared to our Self-GenomeNet model, our approach consistently outperforms this baseline*, particularly in data-scarce settings. We train the generic Self-GenomeNet using virus, bacteria, and human data without using labels and then successively withhold labeled samples of the viral dataset for 1,000 nt sequences to mimic scenarios where labels are scarce (from 100% of available labeled samples to 0.1%). Each point of the generic Self-GenomeNet in the plots is trained separately using the corresponding amount of labeled data. The weights of the context and encoder models are initialized with the training results from the SSL task, but they are trained further (fine-tuned), together with the linear layer, on the new supervised task. The label "7-mer frequency profile" corresponds to the setting where we used the CNN model proposed by Fiannaca et al, where the weights are initialized randomly for the supervised task.

Supplementary References

1. Dai, A. M. & Le, Q. V. Semi-supervised sequence learning. *Adv. Neural Inf. Process. Syst.* **28**, (2015).
2. Lu, A. X., Zhang, H., Ghassemi, M. & Moses, A. Self-Supervised Contrastive Learning of Protein Representations By Mutual Information Maximization. *bioRxiv* 2020.09.04.283929 (2020) doi:10.1101/2020.09.04.283929.
3. van den Oord, A., Li, Y. & Vinyals, O. Representation Learning with Contrastive Predictive Coding. *arXiv [cs.LG]* (2018).
4. Ciortan, M. & Defrance, M. Contrastive self-supervised clustering of scRNA-seq data. *BMC Bioinformatics* **22**, 280 (2021).

5. Fiannaca, A. *et al.* Deep learning models for bacteria taxonomic classification of metagenomic data. *BMC Bioinformatics* **19**, 198 (2018).

5 Contributions to Automated Model Design for Genomics

5.1 Optimized Model Architectures for Deep Learning on Genomic Data

Contributing Article

Gündüz HA, Mreches R, Moosbauer J, Robertson G, To XY, Franzosa EA, Huttenhower C, Rezaei M, McHardy AC, Bischl B, Münch PC, Binder M (2024). "Optimized model architectures for deep learning on genomic data." *Communications Biology*, **7**(1), 516. doi:10.1038/s42003-024-06161-1

Declaration of Contributions

Hüseyin Anil Gündüz contributed to this paper as the first author with the following significant contributions:
Hüseyin Anil Gündüz proposed the design space of architectures to be optimized over, designed the experiments, proposed the baselines, and was chiefly responsible for coding. Hüseyin Anil Gündüz trained, evaluated, and analyzed the baselines and the optimized architectures. Hüseyin Anil Gündüz authored most of the manuscript.

Contribution of the coauthors:
Rene Mreches contributed to the code development. Julia Moosbauer helped with the model-based optimization part of the experiments. Gary Robertson helped with the IT administration. Eric A. Franzosa and Curtis Huttenhower contributed to virus identification and benchmarking. Xiao-Yin To, Mina Rezaei, Alice C. McHardy, and Bernd Bischl provided valuable feedback and input throughout the project. Martin Binder was the main developer of the hyperparameter optimization code. He proposed and developed the particular multi-fidelity approach chosen in this work. Martin Binder supervised and contributed to the code of the proposed model. Martin Binder also ran the model-based optimization part of the experiments to find optimal architectures. The project is mainly supervised by Martin Binder and Philipp C. Münch from a machine learning and bioinformatics perspective, respectively.

All authors contributed to editing the paper.

communications biology

Article

https://doi.org/10.1038/s42003-024-06161-1

Optimized model architectures for deep learning on genomic data

Check for updates

Hüseyin Anil Gündüz[1,2], René Mreches[3,4], Julia Moosbauer[1,2], Gary Robertson[3,4], Xiao-Yin To[1,2,3,4], Eric A. Franzosa[5], Curtis Huttenhower[5], Mina Rezaei[1,2], Alice C. McHardy[3,4,6], Bernd Bischl[1,2], Philipp C. Münch[3,4,5,6,7] ✉ & Martin Binder[1,2,7] ✉

The success of deep learning in various applications depends on task-specific architecture design choices, including the types, hyperparameters, and number of layers. In computational biology, there is no consensus on the optimal architecture design, and decisions are often made using insights from more well-established fields such as computer vision. These may not consider the domain-specific characteristics of genome sequences, potentially limiting performance. Here, we present GenomeNet-Architect, a neural architecture design framework that automatically optimizes deep learning models for genome sequence data. It optimizes the overall layout of the architecture, with a search space specifically designed for genomics. Additionally, it optimizes hyperparameters of individual layers and the model training procedure. On a viral classification task, GenomeNet-Architect reduced the read-level misclassification rate by 19%, with 67% faster inference and 83% fewer parameters, and achieved similar contig-level accuracy with ~100 times fewer parameters compared to the best-performing deep learning baselines.

Deep learning (DL) techniques have been shown to achieve exceptional performance on a wide range of machine learning (ML) tasks, especially when large training sets are available[1]. These techniques have been applied to a variety of challenges in bioinformatics[2–4]. For different ML problems and data modalities, different neural architectures have emerged that perform well in their respective domains, such as convolutional neural networks (CNN) for images or recurrent neural networks (RNN) for text. Architectural design choices are often made based on the experience of researchers and trial and error[5–13]. However, the optimal design and arrangement of these layers are highly domain-specific, problem-dependent, and computationally expensive to evaluate. Besides expert-driven design, it has therefore become increasingly popular to apply systematic approaches to finding neural network configurations, such as automated neural architecture search (NAS)[14]. The number of possible configurations of even small neural networks is very large, as the number of decisions to be made grows exponentially, and most practical NAS algorithms therefore impose various constraints on the search space.

To efficiently perform NAS for ML tasks in genomics, it is essential to identify DL network architecture designs for genomic sequence analysis that are widely recognized in the literature. These designs often start with one or several convolutional layers, followed by a global pooling layer, and conclude with a series of fully connected layers[6,7,9,11]. Recurrent layers offer an alternative to convolutional or global pooling layers. Their ability to propagate information across sequences allows recurrent layers to effectively summarize data, comparable to pooling layers. While numerous works in genomics use RNN layers[15–19], one example is *Seeker*[8], an RNN-based model that employs an LSTM layer for bacteriophage detection. Furthermore, by stacking them sequentially, integrating convolutional and recurrent layers enhances model capability. For instance, the model developed by Wang et al.[20] demonstrates this approach by placing an RNN on top of a convolutional layer, followed by two fully connected layers. A similar configuration is utilized in the DanQ model[21], showcasing the effectiveness of combining recurrent and convolutional layers.

One way to approach NAS is to consider it as a hyperparameter optimization (HPO) problem. Hyperparameters (HPs) are configuration settings that determine how an ML model works. In the context of DL, typical HPs are the choice of the gradient descent algorithm and its learning rate. However, the choice of neural network layers and their configuration can also be considered as HPs. NAS is then equivalent to optimizing HPs that define different

[1]Department of Statistics, LMU Munich, Munich, Germany. [2]Munich Center for Machine Learning, Munich, Germany. [3]Department for Computational Biology of Infection Research, Helmholtz Center for Infection Research, 38124 Braunschweig, Germany. [4]Braunschweig Integrated Centre of Systems Biology (BRICS), Technische Universität Braunschweig, Braunschweig, Germany. [5]Department of Biostatistics, Harvard School of Public Health, Boston, MA, USA. [6]German Centre for Infection Research (DZIF), partner site Hannover Braunschweig, Braunschweig, Germany. [7]These authors jointly supervised this work: Philipp C. Münch, Martin Binder. ✉e-mail: philipp.muench@helmholtz-hzi.de; martin.binder@stat.uni-muenchen.de

https://doi.org/10.1038/s42003-024-06161-1 **Article**

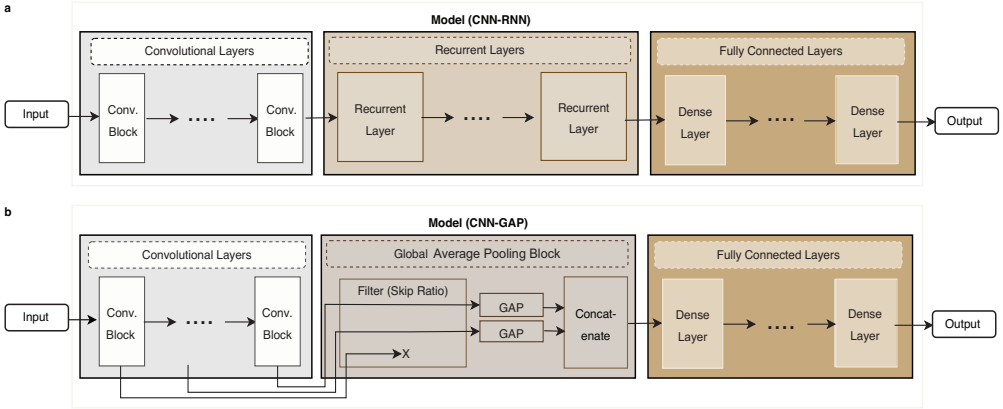


Fig. 1 | The network layout optimized by GenomeNet-Architect consists of three stages: (i) a stage of stacked convolutional layers, (ii) global average pooling (in the CNN-GAP model) or a stack of recurrent layers (in the CNN-RNN model), and (iii) a fully connected stage. a The CNN-RNN model feeds the output of the last convolutional layer into a block of recurrent layers. The output of the last recurrent layer is then flattened and fed into a fully connected neural network. **b** The CNN-GAP model groups the convolutional layers into convolutional blocks. While the output of some of these blocks is skipped (controlled by the "skip ratio" hyperparameter), the network performs global average pooling (GAP) on the remaining blocks and concatenates the result. This is then fed into the fully connected neural network.

architectures. A popular class of optimization algorithms used for HPO is model based optimization (MBO), also called Bayesian optimization[22]. It iteratively evaluates HP configurations and selects new configurations to try based on knowledge of which configurations have worked well in the past. This is done by fitting a regression model, the so-called surrogate model, to the observed performance values. New evaluations are made by considering the "exploration-exploitation tradeoff": new configurations should be tried if their expected performance is high (exploitation), or if the model's uncertainty about their performance is high (exploration). MBO-based methods such as BANANAS[23] have been shown to outperform methods based on other optimization paradigms such as ENAS[24] (which uses reinforcement learning) or DARTS[25] (which uses gradient descent).

The quality of the configurations evaluated by MBO increases gradually as the optimization progresses. The first configurations evaluated, which constitute the initial design, are randomly sampled from the search space without using any prior knowledge. By anticipating that early configurations are unlikely to perform as well as later ones, and by devoting fewer resources to their evaluation, it is possible to reduce the cost of the overall optimization process. Algorithms that speed up optimization by using cheaper approximations of the target objective are called multi-fidelity (MF) optimization algorithms. A simple way to approximate the performance of a DL model is to stop training the model after a certain amount of time, even though the model performance has not fully converged[26].

While there are libraries that perform NAS on genome datasets[27], we are not aware of any methods that use efficient multi-fidelity or MBO methods specifically for genome datasets. MBO has been used in the past to tune specifically designed genomic DL models[28], but only to optimize specific HPs, not as a general NAS framework. No general-purpose MBO-based NAS framework provides a search space specifically modified to fit genome sequence data; in fact, many focus on 2D image data instead.

In this work, we present GenomeNet-Architect, which optimizes DL network architectures by repeatedly constructing new network configurations, training networks based on these configurations on a given dataset, and evaluating the performance of the resulting models by predicting on held-out test data. It uses MBO as an efficient black-box optimization method, combined with a multi-fidelity approach that increases model training time after some initial optimization progress has been made. Unlike other general-purpose NAS frameworks, GenomeNet-Architect uses a search space specifically for genome data. It is made up of neural architectures and HP setups that build on top of and generalize various architectures for the genome data that have been successfully applied in the past. This approach allows us to efficiently explore a large space of possible network architectures and identify those that perform well for genome-related tasks, creating architectures that outperform expert-guided architectures. Our method can be used for a variety of DL tasks on genome sequence data, such as genome-level, loci-level, or nucleotide-level classification and regression.

Results

GenomeNet-Architect uses an efficient global optimization method

GenomeNet-Architect provides a predefined search space of hyperparameters (HPs) that are used to construct different network architectures. It needs to be given a specific ML task on genome sequence data. In our framework, we use model-based optimization (MBO)[22] to jointly tune the network layout and HPs, and generate a specific architecture that works well on the given task.

The result of the optimization process itself is a specific HP configuration that works well for the given task. The resulting architecture can be trained and evaluated on the given data, as well as used to make predictions on new data. However, the resulting architecture can also be used for other tasks that are similar to the task for which it was optimized. It is therefore possible to perform a single optimization run to solve multiple genome sequence DL tasks.

GenomeNet-Architect uses a search space that covers the most common layer types and hyperparameter settings

The search space of GenomeNet-Architect is based on our literature analysis of successful architectures developed for genome data, such as DeepVirFinder[6], ViraMiner[7], Seeker[8], CHEER[9], Fiannaca (CNN model)[10], PPR-Meta[11], and an adapted version of RC-ResNet-18[12]. A common type of architecture consists of convolutional layers followed by global pooling and fully connected layers[6,7,9,11]. An alternative to pooling, which also aggregates information across the entire sequence, is the use of recurrent (RNN) layers.

Inspired by these common patterns observed in many networks successfully applied to genome data, we build a template for an architecture consisting of three stages (Fig. 1): (i) a stage of stacked convolutional layers

https://doi.org/10.1038/s42003-024-06161-1 **Article**

Table 1 | Space of hyperparameters that affect the training and final layout of the model, along with the ranges over which they are optimized

Hyperparameter	Type	Range	Log-Search Space	Component
Learning Rate (l_r)	Float	$[10^{-6}, 10^{-2}]$	✓	General
Reverse-Complement as Additional Input	Boolean	{*True*, *False*}		
Optimizer	Categ	{*Adam*, *Adagrad*, *Rmsprop*, *Sgd*}		
Model Type	Categ	{*GAP*, *RNN*}		
Number of Convolutional Layers (n_c)	Integer	[1,20]		Convolutional Layers
Number of Convolutional Blocks (n_{cb})	Integer	[1,10]		
First Layer Kernel Size (k_0)	Float	$[2^4,2^{11}]$	✓	
Last Layer Kernel Size (k_{end})	Float	$[2^4, 2^{11}]$	✓	
First Layer Number of Filters (f_0)	Float	$[2^1,2^8]$	✓	
Last Layer Number of Filters (f_{end})	Float	$[2^1,2^8]$	✓	
Last Layer Dilation Factor (d_{end})	Float	$[2^0, 2^4]$ *for* $L = 150$ *or* 250, $[2^0, 2^7]$ *for* $L = 10000$	✓	
Total Max-Pooling (p_{end})	Float	$[2^0, 2^4]$ *for* $L = 150$ *or* 250, $[2^0, 2^7]$ *for* $L = 10000$	✓	
Momentum of Batch-Normalization	Float	[0,0.99]		
Leaky-ReLU Alpha Value	Float	[0,1]		
Residual Block (*res_block*)	Boolean	{*True*, *False*}		
Number of Dense Layers	Integer	[0,5]		Fully Connected Layers
Units of Dense Layers	Float	$[2^4,2^{11}]$	✓	
Dropout of Dense Layers	Float	[0,0.99]		
Activation of Dense Layers	Categ	{*ReLU*, *tanh*, *Sigmoid*}		
Recurrent Layer Type	Categ	{*LSTM*, *GRU*}		Recurrent Layers (CNN-RNN only)
Number of Recurrent Layers	Integer	[1,3]		
Uni-/Bidirectional Recurrent Layers	Boolean	{*True*, *False*}		
Number of Recurrent Units	Float	$[2^4, 2^{11}]$	✓	
Skip Ratio for Global Average Pooling (r_s)	Float	[0,1]		Global Average Pooling Block (CNN-GAP only)

Where indicated in the "Log-Search Space" column, hyperparameters are optimized on a logarithmic scale. Hyperparameters are grouped by the "Component" they control, corresponding to the different components shown in Fig. 1.

operating on one-hot encoded input sequences, (ii) a stage for embedding the sequential output data of the convolutional layers into a vector representation, using either global average pooling (GAP) (in a setup that we call the CNN-GAP model) or a stack of recurrent layers (which we call the CNN-RNN model), and (iii) a fully connected neural network stage operating on the embedded values.

Some of the properties we search over include the network layout, such as the number and size of convolutional, dense, and recurrent layers. Other HPs that we searched over influence the training process, such as the optimizer, and the behavior of specific layers, such as the dropout rate, the activation functions, and the batch normalization constant (Table 1). By introducing multiple HPs that influence the final layout of the model, our framework covers many successful architectures from the literature, while also making it possible to find architectures that have not yet been implemented. While the provided search space is our recommendation, our method also supports defining a custom search space, e.g. allowing more layers, or including GMP instead of GAP.

Our HPs cover both the overall architecture of the network (e.g., number of convolutional layers) and the setup of individual layers (e.g., the CNN kernel size). Having different HPs for each layer individually would introduce HP dependencies, which would make the optimization problem more difficult. Therefore, we use a setup where only the first and last layers are directly parameterized. f_0 and f_{end}, for example, specify the number of filters of the first and last convolutional layer. The setup of the intermediate layers is interpolated based on the first and last layers (see Methods for more details).

Model configurations are initially evaluated with shorter runtimes for more efficient search space exploration

Several challenges arise when optimizing DL architectures on complex data modalities such as genomics. First, for complex tasks, the time required for a single model to converge to a solution makes it impractically slow to optimize over a large search space such as the one we have designed. A simple way to speed up model evaluation would be to limit the time for which each proposed model is trained, even if models do not converge within a given timeframe, because models that perform well early in model fitting will continue to perform well after more training epochs[26]. While this reduces the time spent on individual evaluations, the resulting models can only approximate the true performance of a given HP configuration. Smaller models (which have fewer parameters and therefore converge faster) may falsely appear to be superior to larger models that run slower, complete fewer epochs, and cannot converge in the given time limit. However, the models trained for only a short time are still informative about which parts of the search space are more likely to contain models that perform well. We can therefore use them in a "warm start" method that speeds up the optimization process. This works by only partially evaluating initial configuration proposals at first, and using the resulting data about which HPs tend to perform well for short evaluation times to help determine which configurations are later evaluated for longer training times[29].

GenomeNet-Architect first runs the MBO with a fixed, low setting for the model training time $t = t_1$. After a given number of optimization iterations, a new MBO run is started with a higher training time setting $t = t_2$, where the surrogate model contains the performance result data for $t = t_1$ as

https://doi.org/10.1038/s42003-024-06161-1 **Article**

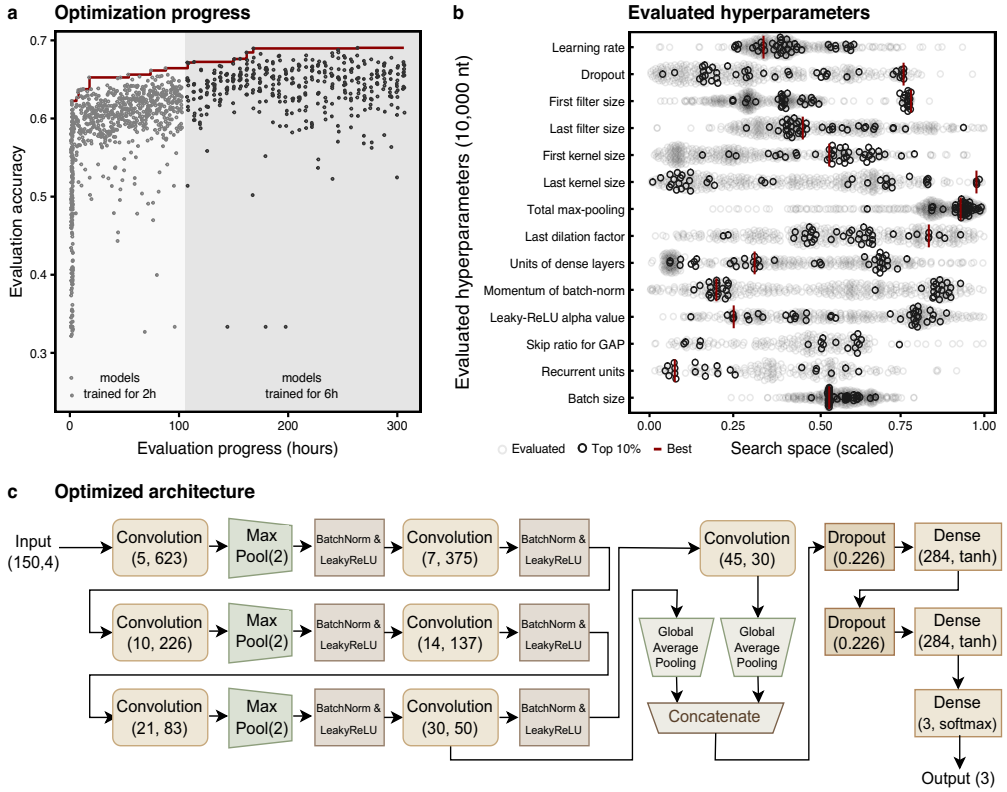


Fig. 2 | Overview of the optimization procedure and results on the viral classification task. We evaluated the performance of the models proposed by our optimization framework on a viral classification task and compared them to various baseline methods for sequence lengths of 150 and 10,000 nt to evaluate for application at the read and (large) contig level (Table 2). **a** The progress of hyperparameter optimization for 150 nt sequence length. Models are trained for 2 h and then 6 h at different stages of the optimization. A better set of hyperparameters is discovered as the optimization proceeds. **b** Evaluated values in the hyperparameter optimization for 6 h of training time and a sequence length of 10,000 nt. The search range (Table 1) of each hyperparameter is normalized in the plot. Dark circles indicate the top 10% of evaluated configurations clustered around favorable values. The best-selected configuration (vertical red lines) often lies within this cluster. **c** The model selected by the hyperparameter optimization stage for a sequence length of 150 is shown (CNN-GAP-6 h in Supplementary Table 1). The values in parentheses indicate kernel size and number of filters for convolutional layers, units and activation for dense layers, and both pool size and stride values for max-pooling layers. Since n_{cb} is 7, there are seven convolutional layers with increasing kernel size and decreasing number of filters along the model. Since the GAP skip ratio is 72%, only the outputs of the last two convolutional layers are pooled and concatenated in the GAP block. This is followed by two dense hidden layers with tanh-activation.

a "warm start". The optimization procedure can be restarted several times with higher values of t_n using all previous points evaluated at times $t_1, ..., t_{n-1}$ as warm start data. Our experiments started with $t_1 = 2$ h and then continued with $t_2 = 6$ h (Fig. 2a). Longer times were also tried, but did not lead to sufficient improvement to justify the additional resources required (Supplementary Note 1). Models evaluated after increasing the training time have higher performance than randomly sampled models at the beginning, showing that the information learned from models trained for a short time is useful for building models trained for a longer time.

GenomeNet-Architect makes use of parallel resources

GenomeNet-Architect parallelizes the HP tuning process across multiple GPUs to reduce the overall optimization time. There are a variety of multi-point proposal techniques that allow MBO to evaluate several different HP configurations simultaneously[30]. A particularly straightforward method is to use the UCB (upper confidence bound) infill criterion[31]: Given a parameter λ, it uses the mean prediction of the surrogate model and adds λ times the model's uncertainty, thereby giving an optimistic bias to regions that have high uncertainty and therefore potential for improvement. By sampling multiple instances of the λ-parameter from an exponential distribution[32], effectively making different tradeoffs between exploration and exploitation, one can generate different point propositions to be evaluated simultaneously. It is used by our method because, despite its simplicity, it is one of the best-performing MBO parallelization methods[30].

In addition to parallelizing individual MBO runs, we also identify setups that are likely to lead to different optimal configurations, and whose optimization can therefore be run independently and in parallel: Which HP settings are optimal, such as specific kernel sizes or number of filters, may vary for different sequence lengths. Similarly, optimal values may differ for CNN-GAP and CNN-RNN, and they may also change depending on whether residual blocks are used. Therefore, the optimization proceeds in a fully crossed design of these choices: The length of the training mini-sequence (in our experiments we investigated both 150 nt, 250 nt, and 10,000 nt), the architecture (CNN-GAP or CNN-RNN), and whether residual connections are used. These optimization runs are independent of each other and can be run in parallel.

https://doi.org/10.1038/s42003-024-06161-1

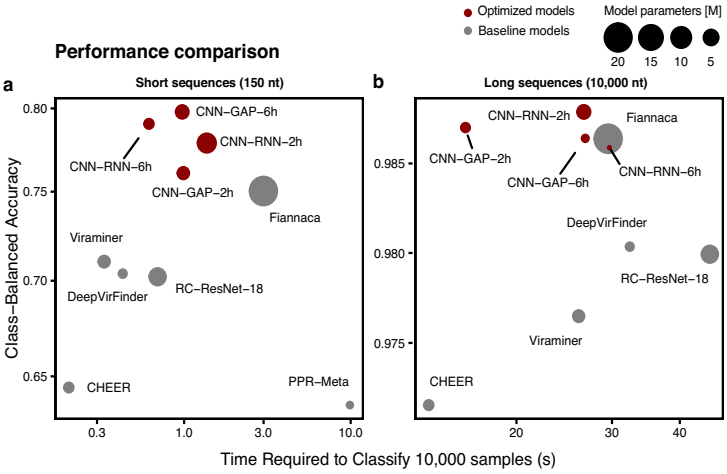


Fig. 3 | Predictive performance and characteristics of models found by GenomeNet-Architect and various baselines on the viral classification task. The inference time to classify 10,000 one hot encoded samples on a GPU and the class-balanced accuracy are shown. The size of the circles indicates the number of model parameters. We have not included Seeker[8] in both graphs and PPR-Meta[11] for long sequences because their performance was too low. **a** At the read-level (150 nt), the best-performing model selected by GenomeNet-Architect (CNN-GAP-6h) reduces the read-level misclassification rate by 19% relative to the best-performing deep learning baseline - Fiannaca[10], despite having 83% fewer parameters and 67% faster inference time. **b** At the contig-level (10,000 nt), models found by GenomeNet-Architect perform on par or better than the best-performing baseline, although all these models perform very well in terms of accuracy. However, much faster (CNN-GAP-2h) and much smaller (CNN-RNN-6h) models are found, all with balanced accuracy close to the best baseline (~ 98.6%).

GenomeNet-Architect finds models that outperform expert-designed baseline models in the viral identification task

GenomeNet-Architect demonstrated superior performance on the virus classification task compared to other deep learning (DL) and non-deep learning methods that we selected as baselines, effectively distinguishing between sequences originating from bacterial chromosomes, prokaryotic viruses (referred to as bacteriophages) and eukaryotic viruses (referred to as viral non-phage DNA). We have tested the effectiveness of GenomeNet-Architect against baselines for classification at the read-level (150 nt long sequences) and at the contig-level (10,000 nt) separately. At the read level, GenomeNet-Architect reduces the class-balanced misclassification rate, i.e., the misclassification rate averaged over all classes, by 19%, while having 83% fewer parameters and achieving 67% faster inference time compared to the best DL baseline (Fiannaca[10]) and outperforms k-mer-based and alignment-based approaches for sequence classification (Fig. 3a, Supplementary Table 2). At the contig-level, the best model found by our method achieves a class-balanced misclassification rate of 1.21%, outperforming the best baseline (1.36%) while being 82% smaller. GenomeNet-Architect also finds a model that performs comparably to the baseline (1.41%) while having a factor of 117 times fewer parameters (Fig. 3b, Supplementary Table 1). For a fair comparison, we trained and validated all DL baseline models on the same dataset and dataset splits. Additionally, we standardized the configuration by adapting the output layer of each model and employing multi-class cross-entropy as the loss function, aligning with our models to facilitate three-class classification. This approach allows for a direct comparison of algorithmic improvements.

We show the best configuration found for this classification task using 10,000 nt sequences (red lines), as well as the top 10% configurations (solid circles) in front of all evaluated configurations (transparent circles) (Fig. 2b) and a diagram visualizing the optimized architecture for the classification task using 150 nt (Fig. 2c). We also analyzed the performance of the optimized model stratified by the degree of genomic differences to the training data. Our findings show that it consistently outperforms the Fiannaca baseline, demonstrating that GenomeNet-Architect's performance does not come from overfitting on sequences that occur in the training dataset (Supplementary Fig. 1). The superior performance persists even when reads are simulated with the Illumina read error profile, underscoring the tool's effectiveness across diverse genomic sequencing challenges (Supplementary Fig. 2).

GenomeNet-Architect identifies models that outperform expert-designed baselines in the pathogenicity detection task

To further validate the versatility of GenomeNet-Architect, we extended our experiments to a second task: pathogen detection in bacteria, specifically to distinguish between pathogenic and non-pathogenic sequences in human hosts. We aligned our evaluation with the baseline values reported in the study of Bartoszewicz et al.[13], utilizing the same search space for the viral classification task and hyperparameter optimization stage, where models are optimized for 2 h. We also fine-tuned the pre-trained DNABERT[33] (6-mer model), using the suggested hyperparameter settings given for fine-tuning on the method's GitHub page. We added it as an additional baseline for this task to make our benchmark more comprehensive. GenomeNet-Architect's optimized models outperform all baseline models, showing substantial improvement in pathogenicity detection (up to 11% improvement, see Fig. 4).

Additionally, we adapted the models originally optimized for the viral classification (initially optimized for sequences of 150 nt, CNN-RNN-6h, and CNN-GAP-6h) by adjusting the input size to 250 nt to evaluate how well performance of an architecture optimized for one task transfers to a different task and conditions. These architectures are renamed to have "VC" (short for viral classification) as a suffix. The comparable performance of "VC" models to models detected by GenomeNet-Architect on this dataset "GAP-CNN" and "GAP-RNN" shows good transfer between related tasks on genome data (Fig. 4).

To enhance the predictive accuracy and robustness, we explored the efficacy of ensemble approaches, a technique that combines multiple model predictions, akin to the approach presented in Bartoszewicz et al.[13], merging RC-LSTM and RC-CNN models into an RC-CNN + LSTM ensemble. Our experiments with ensemble models, including GAP + RNN-VC, GAP + RNN, and the 4-model ensemble combining both CNN-RNN and CNN-GAP variants, demonstrate a notable decrease in misclassification rates. Specifically, the 4-model ensemble reduced misclassification rates by 11% compared to the RC-CNN + LSTM baseline, with a single model improvement of 8% for CNN-GAP-VC versus RC-CNN.

Discussion

GenomeNet-Architect defines an HP configuration search space for neural architectures that extends and generalizes successful genome data architectures from the past. This adaptable search space is coupled with an efficient black-box optimization method that can generate more optimal network architectures for genome-related tasks compared to expert-

https://doi.org/10.1038/s42003-024-06161-1 **Article**

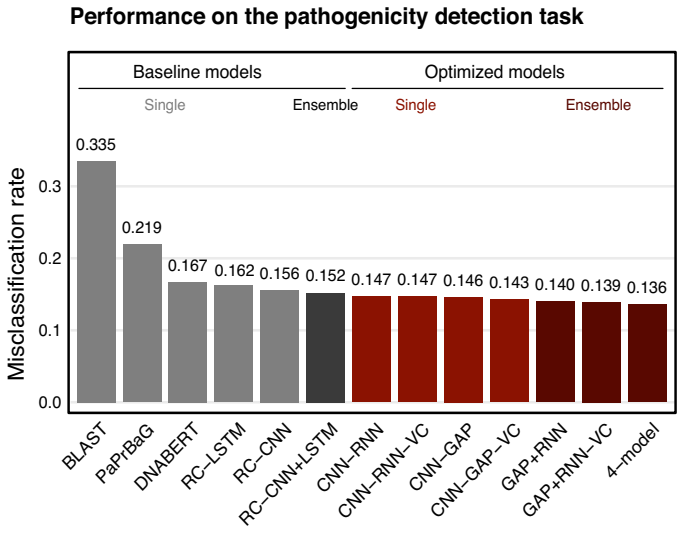


Fig. 4 | Comparative analysis of misclassification rates in the pathogenicity detection task. The baseline models are shown in gray, while the red bars indicate the models developed by GenomeNet-Architect. The data for the dataset itself and the baseline results, with the exception of DNABERT[33], were derived from the DeePaC study[13]. In addition, the pre-trained DNABERT[33] model is fine-tuned on this task and added as a baseline. The graph shows individual model performance along with the improved performance archived by the ensemble approaches and highlights the superior performance of the GenomeNet-Architect models over various baselines.

designed architectures. With GenomeNet-Architect, researchers can identify better models when applying DL to genomic datasets.

The search space used by GenomeNet-Architect leads to resulting architectures that are similar to other models used in the literature, and the individual components of the resulting architecture therefore have similar interpretations. The convolutional layer can be thought of as a pattern-matching method that encodes the presence of motifs in short subsequences. The global pooling layer then aggregates the information about specific patterns within the entire sequence. A global average pooling (GAP) layer measures the relative frequency of these patterns, as opposed to the encoding of the presence of individual patterns that are recorded in some models that use global max-pooling (GMP). Models using RNNs, on the other hand, are able to predict outcomes based on the spatial relationships between different patterns and can learn long-term dependencies[8,34]. The following fully connected layers are used to learn complex relationships between detected patterns and can be used in both GAP and RNN-based models[20].

The *DeepVirFinder*[6] model is an architecture that uses pooling, with a convolutional layer, followed by a global max-pooling layer and a fully connected layer. The *ViraMiner*[7] model builds on the DeepVirFinder model and proposes two branches called frequency and pattern branches, using either GAP (frequency branch) or GMP (pattern branch) after separate convolutional layers. In both branches, a fully connected layer follows, after which their output vectors are concatenated. Finally, another fully connected layer classifies whether a 300 nt sequence is human or viral DNA. Since Tampuu et al.[7] showed that the GAP alone achieves higher performance than the GMP alone, we did not include GMP in our search space. Another similarly structured architecture for viral classification is the *CHEER* model[9]. In this model, reads from 250 nt sequences are fed into four different convolutional layers with different kernel sizes: 3, 7, 11, and 15. Global max-pooling layers follow each convolutional layer, after which the paths are concatenated. Similar to other architectures, multiple fully connected layers follow the concatenation.

DeepMicrobes is another RNN-based DL model developed for viral identification. The model consists of a 12-mer embedding layer, a bidirectional LSTM layer, a self-attention layer, and several fully connected layers. The convolutional layer in this model learns local representations, and the recurrent layer can learn long-term dependencies within these local representations in a sequential manner.

Although DL models based on raw nucleotide sequences are common, there are also alternatives. One example is *Fiannaca-CNN*[10], which is a model for bacterial classification. The model uses the number of k-mer occurrences as input, which is fed into convolutional layers followed by max-pooling layers and fully connected layers. We used $k = 7$ (7-mers) in our experiments because they show that the highest accuracy is achieved using this HP. Another example is *PPR-Meta*[11], which is used to classify if the sequence is a plasmid, chromosome, or phage. The input to the model is both one-hot encoded nucleotides and 3-mers. In addition, the reverse-complements of the original inputs are concatenated to the original sequence for both inputs. The model consists of two different three convolutional layers, the outputs of which are global average pooled, concatenated, and fed into a fully connected layer. There are also max-pooling and batch-normalization layers after the first two convolutional layers.

Our HPO results provide valuable insight into the design and training of architectures for specific tasks and datasets. For example, increasing the kernel size, number of filters, and layers in convolutional networks can substantially increase both the number of trainable parameters and memory requirements, resulting in a trade-off. Many existing models, such as DeepVirFinder[6], ViraMiner[7] or CHEER[9], are limited to a single (or multiple but parallelized, not sequential) convolutional layer with a large kernel size (up to ~15) and a large number of filters (~ 1000). Although PPR-Meta[11] proposes a deeper model (3 sequential convolutional layers), it compensates by reducing the kernel size (down to 3). Our HPO framework has discovered an architecture that performs better on viral classification in terms of accuracy: deeper (7 convolutional layers) with a smaller number of filters in the final convolutional layers (as low as 30).

In examining model architectures that perform well across two datasets and three different sequence lengths, we sought to identify common trends and patterns in their architecture designs. Our analysis reveals that architectures with GAP layers typically incorporate more convolutional layers (5 to 7, as opposed to 1 to 5 in RNN models) and more fully connected layers (1 or 2, vs. 0 or 1) (Supplementary Table 1). The preference for GAP layers is likely due to their function in aggregating information learned by convolutional layers through averaging over the sequence, instead of learning representations by optimizing its own weights. Compared to CNN-RNN models, CNN-GAP models mainly use fully connected layers to integrate long-range information. Furthermore, LSTM layers are consistently preferred to GRU layers. Our findings also indicate a general avoidance of

https://doi.org/10.1038/s42003-024-06161-1 **Article**

multiple recurrent layers, while bidirectional RNN layers are preferred over unidirectional ones. In terms of training hyperparameters, the optimized learning rate is typically between 10^{-3} and 10^{-4} with the Adam[35] optimizer more commonly chosen over alternatives like Adagrad[36], Rmsprop, and SGD[37]. It is important to note, however, that these trends are observations and may not universally apply to every dataset or task. Therefore, we recommend running GenomeNet-Architect on the specific dataset and task in question to tailor the model architecture for optimal performance in each unique scenario.

The results of the 10,000 nt setting in contrast to the 150 nt setting are noteworthy in that there is a much smaller improvement in accuracy over the baselines in the 10,000 nt setting. This is because the viral identification task becomes "too easy" at 10,000 nucleotides, leaving little room for improvement. In such settings, where relatively simple models already perform sufficiently well, it may not be worth the considerable computational overhead of finding a specialized architecture.

Method

Hyperparameter search space

The hyperparameter space used for optimization is listed in Table 1 and described in more detail here.

The first part of the model constructed by GenomeNet-Architect consists of a sequence of convolutional blocks (Fig. 1), each of which consists of convolutional layers. The number of blocks (N_{cb}) and the number of layers in each block (s_{cb}) is determined by the HPs n_{cb} and n_c in the following way: N_{cb} is directly set to n_{cb} unless n_c (which relates to the total number of convolutional layers) is less than that. Their relation is therefore

$$N_{cb} = \begin{cases} n_c, & \text{if } n_c \le n_{cb} \\ n_{cb}, & \text{otherwise} \end{cases}$$

s_{cb} is calculated by rounding the ratio of the n_c hyperparameter to the actual number of convolutional blocks N_{cb}:

$$s_{cb} = round\left(\frac{n_c}{N_{cb}}\right).$$

This results in n_c determining the approximate total number of convolutional layers while satisfying the constraint that each convolutional block has the same (integer) number of layers. The total number of convolutional layers is then given by

$$N_c = N_{cb} \times s_{cb}.$$

f_0 and f_{end} determine the number of filters in the first or last convolutional layers, respectively. The number of filters in intermediate layers is interpolated exponentially. If residual blocks are used, the number of filters within each convolutional block needs to be the same, in which case the number of filters changes block-wise. Otherwise, each convolutional layer can have a different number of filters. If there is only one convolutional layer, $\lfloor f_0 \rfloor$ is used as the number of filters in this layer. Thus, the number of filters for the i^{th} convolutional layer is:

$$f_i = \left\lfloor f_0 \times \left(\frac{f_{end}}{f_0}\right)^{j(i)} \right\rfloor, \; j(i) = \begin{cases} \left\lfloor \frac{i}{s_{cb}} \right\rfloor \times \frac{1}{N_{cb}-1}, & \text{if res\_block} \\ \frac{i}{N_c-1}, & \text{otherwise} \end{cases}.$$

The kernel size of the convolutional layers is also exponentially interpolated between k_0 and k_{end}. If the model has only one convolutional layer, the kernel size is set to $\lceil k_0 \rceil$. The kernel size of the convolutional layer i is:

$$k_i = \left\lceil k_0 \times \left(\frac{k_{end}}{k_0}\right)^{\frac{i}{N_c-1}} \right\rceil.$$

The convolutional layers can use dilated convolutions, where the dilation factor increases exponentially from 1 to d_{end} within each convolutional block. Using "***rem***" as the remainder operation, the dilation factor for each layer is then:

$$d_i = \left\lceil d_{end}^{(\lfloor i \; \mathbf{rem} \; s_{cb} \rfloor)/(s_{cb}-1)} \right\rceil.$$

We apply max-pooling after convolutional layers, depending on the total max-pooling factor p_{end}. Max pooling layers of stride and a kernel size of 2 or the power of 2 are inserted between convolutional layers so that the sequence length is reduced exponentially along the model. p_{end} represents the approximate value of total reduction in the sequence length before the output of the convolutional part is fed into the last GAP layer or into the RNN layers depending on the model type.

For CNN-GAP, outputs from multiple convolutional blocks can be pooled, concatenated, and fed into a fully connected network. Out of N_{cb} outputs, the last $min(1, \lceil (1 - r_s) \times N_{cb} \rceil)$ of them are fed into global average pooling layers, where r_s is the skip ratio hyperparameter.

Hyperparameter optimization process

GenomeNet-Architect uses the mlrMBO software[38] with a Gaussian process model from the DiceKriging R package[39] configured with a Matérn-3/2 kernel[40] for optimization. It uses the UCB[31] infill criterion, sampling λ from an exponential distribution as a batch proposal method[32]. In our experiment, we proposed three different configurations simultaneously in each iteration.

For both tasks, we trained the proposed model configurations for a given amount of time and then evaluated them afterwards on the validation set. For each architecture (CNN-GAP and CNN-RNN) and for each sequence length of the viral classification task (150 nt and 10,000 nt), the best-performing model configuration found within the optimization setting (2 h, 6 h) was saved and considered for further evaluation. For the pathogenicity detection task, we only evaluated the 2 h optimization. For each task and sequence length value, the first $t = t_1$ (2 h) optimization evaluated a total of 788 configurations, parallelized on 24 GPUs, and ran for 2.8 days (wall time). For the viral classification task, the warm-started $t = t_2$ (6 h) optimization evaluated 408 more configurations and ran for 7.0 days for each sequence length value.

During HPO, the number of samples between model validation evaluations was set dynamically, depending on the time taken for a single model training step. It was chosen so that approximately 20 validation evaluations were performed for each model in the first phase (t = 2 h), and approximately 100 validation evaluations were performed in the second phase (t = 6 hours). In the first phase, the highest validation accuracy found during model training was used as the objective value to be optimized. In the second phase, the second-highest validation accuracy found in the last 20 validation evaluations was used as the objective value. This was done to avoid rewarding models with a very noisy training process with performance outliers.

The batch size of each model architecture is chosen to be as large as possible while still fitting into GPU memory. To do this, GenomeNet-Architect performs a binary search to find the largest model that still fits in the GPU and subtracts a 10% safety margin to avoid potential training failures.

Architecture evaluation and benchmarks

For the viral classification task, the training and validation samples are generated by randomly sampling FASTA genome files and splitting them into disjoint consecutive subsequences from a random starting point. A batch size that is a multiple of 3 (the number of target classes) is used, and each batch contains the same number of samples from each class. Since we work with datasets that have different quantities of data for each class, this effectively oversamples the minor classes compared to the largest class. The validation set performance was evaluated at regular intervals after training

https://doi.org/10.1038/s42003-024-06161-1 **Article**

Table 2 | Description of the datasets used in our experiments

	Number of FASTA Files			Number of Sequences (L = 150)			Number of Sequences (L = 10k)		
Class	Training	Validation	Test	Training	Validation	Test	Training	Validation	Test
Bacteria	15,826	4523	2263	373,404,076	118,398,785	59,111,408	5,579,451	1,772,121	881,031
Virus (non-Phage)	18,093	5171	2588	2,262,526	568,717	458,526	20,075	5552	5350
Bacteriophage	9987	2855	1428	4,702,821	1,301,375	609,088	64,937	18,031	8400

on a predetermined number of samples, set to 6,000,000 for the 150 nt models and 600,000 for the 10,000 nt models. The evaluation used a subsample of the validation set equal to 50% of the training samples seen between each validation. During the model training, the typical batch size was 1200 for the 150 nt models, and either 120, 60, or 30 for the 10,000 nt models. Unlike during training and validation, the test set samples were not randomly generated by selecting random FASTA files. Instead, test samples were generated by iterating through all individual files, and using consecutive subsequences starting from the first position. For the pathogenicity detection task, the validation performance was evaluated at regular intervals on the complete set, specifically once after training on 5,000,000 samples. The batch size of 1000 was used for all models, except for GAP-RNN, as it was not possible with the memory of our GPU. For this model, a batch size of 500 was used.

For both tasks, we chose a learning rate schedule that automatically reduced the learning rate by half if the balanced accuracy did not increase for 3 consecutive evaluations on the validation set. We stopped the training when the balanced accuracy did not increase for 10 consecutive evaluations. This typically corresponds to stopping the training after 40/50 evaluations for the 150 nt models, 25/35 evaluations for the 10,000 nt models for the viral classification tasks, and 5/15 evaluations for the pathogenicity detection task.

To evaluate the performance of the architectures and HP configurations, the models proposed by GenomeNet-Architect were trained until convergence on the training set; convergence was checked on the validation set. The resulting models were then evaluated on a test set that was not seen during optimization.

Datasets

For the viral classification task, we downloaded all complete bacterial and viral genomes from GeneBank and RefSeq using the genome updater script (https://github.com/pirovc/genome_updater) on 04-11-2020 with the arguments -d "genbank,refseq" -g "bacteria"/"viral" -c "all" and -l "Complete Genome". To filter out possible contamination consisting of plasmids and bacteriophages, we removed all genomes from the bacteria set with more than one chromosome. Filtering out plasmids due to their inconsistent and poor annotations in databases avoids introducing substantial noise in sequence and annotation since they can be incorrectly included or excluded in genomes. We used the taxonomic metadata to split the viral set into eukaryotic or prokaryotic viruses. Overall this resulted in three subgroups: bacteria, prokaryotic bacteriophages, and eukaryotic viruses (referred to as non-phage viruses, Table 2). To assess the model's generalization performance, we subset the genomes into training, validation, and test subsets. We used the "date of publishing" metadata to split the data by publication time, with the training data consisting mostly of genomes published before 2020, and the validation and test data consisting of more recently published genomes. Thus, when applied to newly sequenced DNA, the classification performance of the models on yet unknown data is estimated. For smaller datasets, using average nucleotide identity information (ANI) generated with tools such as Mashtree[41] to perform the splits can alternatively be used to avoid overlap between training and test data.

The training data was used for model fitting, the validation data was used to estimate generalization performance during HPO and to check for convergence during final model training, and the test data was used to compare final model performance and draw conclusions. The test data was not seen by the optimization process. The training, validation and test sets represent approximately 70%, 20%, and 10% of the total data, respectively.

The number of FASTA files in the sets and the number of non-overlapping samples in sets of the viral classification task are listed in Table 2. Listed is the number of different non-overlapping sequences that could theoretically be extracted from the datasets, were they split into consecutive subsequences. However, whenever the training process reads a file again, e.g. in a different epoch, the starting point of the sequence to be sampled is randomized, resulting in a much larger number of possible distinct (though overlapping) samples. Because the size of the test set is imbalanced, we report class-balanced measures, i.e. measures calculated for each class individually and then averaged over all classes.

For the pathogenicity classification task, we downloaded the dataset from https://zenodo.org/records/367856313. Specifically, the used training files are nonpathogenic_train.fasta.gz, pathogenic_train.fasta.gz, the used validation files are pathogenic_val.fasta.gz, nonpathogenic_val.fasta.gz, and the used test files are nonpathogenic_test_1.fasta.gz, nonpathogenic_test_2.fasta.gz, pathogenic_test_1.fasta.gz, pathogenic_test_2.fasta.gz.

Reporting summary

Further information on research design is available in the Nature Portfolio Reporting Summary linked to this article.

Data availability

The data for the virus identification task is available under https://research.bifo.helmholtz-hzi.de/downloads/deepg_refpacks/architect_training_data.tar.gz. Data for the pathogen detection task is taken from a study of Bartoszewicz et al.[13] https://zenodo.org/records/3678563 as mentioned in the Datasets subsection. Source data for figures can be found in Supplementary Data 1 as well as on https://github.com/GenomeNet/Architect (https://doi.org/10.5281/zenodo.10889923).

Code availability

The code, available at https://github.com/GenomeNet/Architect, enables users to apply the optimization process across various datasets and tasks. It is based on our R library deepG (deepg.de) and can therefore be adapted to a variety of genomics tasks that are supported by it. It uses the TensorFlow backend and can be made to run in parallel on multi-GPU-machines and compute clusters through the batchtools[42] R package.

Received: 8 February 2023; Accepted: 8 April 2024;
Published online: 30 April 2024

References

1. LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *Nature* **521**, 436–444 (2015).
2. AlQuraishi, M. AlphaFold at CASP13. *Bioinformatics* **35**, 4862–4865 (2019).
3. Ronneberger, O., Fischer, P. & Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015* 234–241 (Springer International Publishing, 2015).
4. Daoud, M. & Mayo, M. A survey of neural network-based cancer prediction models from microarray data. *Artif. Intell. Med.* **97**, 204–214 (2019).

https://doi.org/10.1038/s42003-024-06161-1 **Article**

5. Patterson, J. & Gibson, A. *Deep Learning: A Practitioner's Approach*. ('O'Reilly Media, Inc.' 2017).
6. Ren, J. et al. Identifying viruses from metagenomic data using deep learning. *Quant. Biol.* **8**, 64–77 (2020).
7. Tampuu, A., Bzhalava, Z., Dillner, J. & Vicente, R. ViraMiner: Deep learning on raw DNA sequences for identifying viral genomes in human samples. *PLoS One* **14**, e0222271 (2019).
8. Auslander, N., Gussow, A. B., Benler, S., Wolf, Y. I. & Koonin, E. V. Seeker: alignment-free identification of bacteriophage genomes by deep learning. *Nucleic Acids Res.* **48**, e121 (2020).
9. Shang, J. & Sun, Y. CHEER: HierarCHical taxonomic classification for viral mEtagEnomic data via deep learning. *Methods* **189**, 95–103 (2021).
10. Fiannaca, A. et al. Deep learning models for bacteria taxonomic classification of metagenomic data. *BMC Bioinformatics* **19**, 198 (2018).
11. Fang, Z. et al. PPR-Meta: a tool for identifying phages and plasmids from metagenomic fragments using deep learning. *Gigascience* **8**, giz066 (2019).
12. Bartoszewicz, J. M., Nasri, F., Nowicka, M. & Renard, B. Y. Detecting DNA of novel fungal pathogens using ResNets and a curated fungi-hosts data collection. *Bioinformatics* **38**, ii168–ii174 (2022).
13. Bartoszewicz, J. M., Seidel, A., Rentzsch, R. & Renard, B. Y. DeePaC: predicting pathogenic potential of novel DNA with reverse-complement neural networks. *Bioinformatics* **36**, 81–89 (2019).
14. Elsken, T., Metzen, J. H., & Hutter, F. Neural architecture search: A survey. *J. Machine Learn. Res.* **20**, 1–21 (2019).
15. Eraslan, G., Avsec, Ž., Gagneur, J. & Theis, F. J. Deep learning: new computational modelling techniques for genomics. *Nat. Rev. Genet.* **20**, 389–403 (2019).
16. Koumakis, L. Deep learning models in genomics; are we there yet? *Comput. Struct. Biotechnol. J.* **18**, 1466–1473 (2020).
17. Boža, V., Brejová, B. & Vinař, T. DeepNano: Deep recurrent neural networks for base calling in MinION nanopore reads. *PLoS One* **12**, e0178751 (2017).
18. Cao, R. et al. ProLanGO: Protein Function Prediction Using Neural Machine Translation Based on a Recurrent Neural Network. *Molecules* **22**, 1732 (2017).
19. Shen, X., Jiang, C., Wen, Y., Li, C. & Lu, Q. A brief review on deep learning applications in genomic studies. Front. Syst. Biol., **2**, 877717 (2022).
20. Wang, R., Zang, T. & Wang, Y. Human mitochondrial genome compression using machine learning techniques. *Hum. Genomics* **13**, 49 (2019).
21. Quang, D. & Xie, X. DanQ: a hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences. *Nucleic Acids Res.* **44**, e107 (2016).
22. Snoek, J., Larochelle, H. & Adams, R. P. Practical bayesian optimization of machine learning algorithms. *Adv. Neural Inf. Process. Syst*. **25**, (2012).
23. White, C., Neiswanger, W., & Savani, Y. Bananas: Bayesian optimization with neural architectures for neural architecture search. In Proceedings of the AAAI conference on artificial intelligence (Vol. 35, No. 12, pp. 10293–10301) (2021).
24. Pham, H., Guan, M., Zoph, B., Le, Q. & Dean, J. Efficient Neural Architecture Search via Parameters Sharing. **80**, 4095–4104 (2018).
25. Liu, H., Simonyan, K. & Yang, Y. DARTS: Differentiable Architecture Search. *arXiv [cs.LG]* (2018).
26. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A. & Talwalkar, A. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv [cs.LG]* (2016).
27. Zhang, Z., Park, C. Y., Theesfeld, C. L. & Troyanskaya, O. G. An automated framework for efficiently designing deep convolutional neural networks in genomics. *Nat. Machine Intell.* **3**, 392–400 (2021).
28. Kelley, D. R. et al. Sequential regulatory activity prediction across chromosomes with convolutional neural networks. *Genome Res.* **28**, 739–750 (2018).
29. Booker, A. J. et al. A rigorous framework for optimization of expensive functions by surrogates. *Struct. Optimization* **17**, 1–13 (1999).
30. Bischl, B., Wessing, S., Bauer, N., Friedrichs, K. & Weihs, C. MOI-MBO: Multiobjective Infill for Parallel Model-Based Optimization. in *Learning and Intelligent Optimization* 173–186 (Springer International Publishing, 2014).
31. Srinivas, N., Krause, A., Kakade, S. M. & Seeger, M. Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. *arXiv [cs.LG]* (2009).
32. Hutter, F., Hoos, H. H. & Leyton-Brown, K. Parallel Algorithm Configuration. in *Learning and Intelligent Optimization* 55–70 (Springer Berlin Heidelberg, 2012).
33. Ji, Y., Zhou, Z., Liu, H. & Davuluri, R. V. DNABERT: pre-trained Bidirectional Encoder Representations from Transformers model for DNA-language in genome. *Bioinformatics* https://doi.org/10.1093/bioinformatics/btab083. (2021).
34. Liang, Q., Bible, P. W., Liu, Y., Zou, B. & Wei, L. DeepMicrobes: taxonomic classification for metagenomics with deep learning. *NAR Genom. Bioinform.* **2**, lqaa009 (2020).
35. Kingma, D. P. & Ba, J. Adam: A Method for Stochastic Optimization. *arXiv [cs.LG]* (2014).
36. Stochastic Optimization. Adaptive Subgradient Methods for. https://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf (2011).
37. Robbins, H. & Monro, S. A Stochastic Approximation Method. *Ann. Math. Stat.* **22**, 400–407 (1951).
38. Bischl, B. et al. mlrMBO: A Modular Framework for Model-Based Optimization of Expensive Black-Box Functions. *arXiv [stat.ML]* (2017).
39. Roustant, O., Ginsbourger, D. & Deville, Y. DiceKriging, DiceOptim: Two R Packages for the Analysis of Computer Experiments by Kriging-Based Metamodeling and Optimization. *J. Stat. Softw.* **51**, 1–55 (2012).
40. Genton, M. G., Cristianini, N., Shawe-Taylor, J. & Williamson, R. Classes of kernels for machine learning: A statistics perspective. https://www.jmlr.org/papers/volume2/genton01a/genton01a.pdf?ref=https://githubhelp.com.
41. Katz, L. S. et al. Mashtree: a rapid comparison of whole genome sequence files. *J. Open Source Softw*. **4**, 44 (2019).
42. Lang, M., Bischl, B. & Surmann, D. batchtools: Tools for R to work on batch systems. *J. Open Source Softw.* **2**, 135 (2017).

Acknowledgements

This work was funded in part by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A and under GenomeNet Grant No. 031L0199A/031L0199B and by the Deutsche Forschungsgemeinschaft DFG EXC 2155. P.C.M. received funding from the German Research Foundation (Grant number 405892038). X.-Y.T. received funding from the German Center for Infection Research (DZIF) TI BBD. C.H. received funding from NIH U19AI110820.

Author contributions

H.A.G., M.B., and P.C.M. designed the study and experiments. H.A.G. was responsible for coding the baselines and optimized architectures, and for training, evaluating, and analyzing them. H.A.G. drafted the initial manuscript. R.M. and J.M. contributed to code development. M.B. was the main developer of the hyperparameter optimization code and the multi-fidelity approach used. M.B. also supervised and contributed to the model code, and ran the model-based optimization experiments to find optimal architectures. G.R. provided computational resources and support. E.A.F. and C.H. contributed to virus identification and benchmarking. M.R., A.C.M., B.B., and X.-Y.T. provided valuable feedback and input throughout the

https://doi.org/10.1038/s42003-024-06161-1 **Article**

project. The project was supervised by M.B. and P.C.M. All authors contributed to the writing of the manuscript.

Funding

Open Access funding enabled and organized by Projekt DEAL.

Competing interests

The authors declare no competing interests.

Additional information

Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s42003-024-06161-1.

Correspondence and requests for materials should be addressed to Philipp C. Münch or Martin Binder.

Peer review information *Communications Biology* thanks the anonymous reviewers for their contribution to the peer review of this work. Primary Handling Editors: Gene Chong and Luke R. Grinham. A peer review file is available.

Reprints and permissions information is available at http://www.nature.com/reprints

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

© The Author(s) 2024, corrected publication 2024

Supplementary Material

Supplementary Results

Task	Viral Classification								Pathogenicity Detection	
Sequence Length	150				10,000				250	
Metric	CNN-GAP-2h	**CNN-GAP-6h**	CNN-RNN-2h	CNN-RNN-6h	CNN-GAP-2h	CNN-GAP-6h	**CNN-RNN-2h**	CNN-RNN-6h	**CNN-GAP**	CNN-RNN
Learning Rate	0.0002295	**0.0002138**	0.0002630	0.0014382	0.0003255	0.00027436	**0.0001168**	0.0004349	**0.0002418**	0.0003251
RC as Additional Input	True	**True**	True	True	False	True	**True**	False	**True**	True
Optimizer	Adam	**Adam**	Adam	Adam	Adam	Adam	**Adam**	Adam	**Adam**	Adam
Model Type	GAP	**GAP**	RNN	RNN	GAP	GAP	**RNN**	RNN	**GAP**	RNN
Number of Conv. Layers (n_c)	6	**9**	5	3	5	5	**4**	1	**6**	5
Number of Conv. Blocks (n_{cb})	7	**7**	3	4	5	7	**9**	10	**7**	3
First Kernel Size (k_0)	3.7992	**4.1379**	2.6317	6.0517	7.8576	15.0923	**2.8262**	12.8539	**4.1487**	2.6463
Last Kernel Size (k_{end})	20.9910	**44.4341**	5.4452	53.4114	2.5291	2.6436	**2.7787**	58.9992	**19.2934**	3.4839
First Number of Filters (f_0)	777.8104	**622.1899**	262.3433	378.8905	146.1158	108.9821	**60.1536**	708.3316	**697.8605**	147.8401
Last Number of Filters (f_{end})	22.3111	**29.8014**	901.9673	57.5173	370.7959	117.3769	**401.0379**	147.4949	**49.6290**	748.9535
Last Dilation Factor (d_{end})	2.1027	**3.3495**	1.9187	3.2531	11.1640	22.2875	**6.0812**	57.2750	**2.7963**	4.6694
Total Max-Pooling (p_{end})	15.4875	**15.7883**	14.7550	7.0775	69.7147	83.9126	**115.4169**	90.8419	**21.0560**	3.1384
Momentum of Batch-Norm	0.6888	**0.7605**	0.4730	0.1840	0.7577	0.8495	**0.1288**	0.1993	**0.6945**	0.0261
Leaky-ReLU Alpha Value	0.2630	**0.0481**	0.3142	0.0302	0.8579	0.8018	**0.8324**	0.2514	**0.3399**	0.5381
Residual Block	False	**False**	False	True	False	False	**True**	True	**False**	False

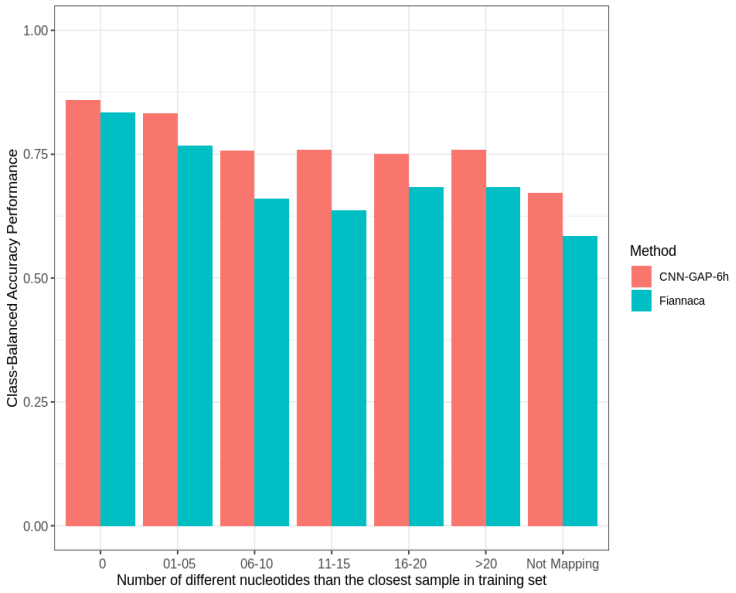
Number of Dense Layers	1	**2**	0	0	2	2	**1**	0	**1**	1
Units of Dense Layers	79	**284**	1474	32	365	470	**152**	73	**101**	1033
Dropout	0.1844	**0.2257**	0.3145	0.2108	0.1406	0.1424	**0.3110**	0.6830	**0.1773**	0.3122
Activation of Dense Layers	tanh	**tanh**	tanh	ReLU	ReLU	ReLU	**tanh**	tanh	**tanh**	tanh
Recurrent Type	-	**-**	LSTM	LSTM	-	-	**LSTM**	LSTM	**-**	LSTM
Number of Rec.Layers	-	**-**	3	1	-	-	**1**	1	**-**	1
Uni-/Bi- dir, Rec. Layers	-	**-**	True	True	-	-	**True**	True	**-**	True
Recurrent Units	-	**-**	19	58	-	-	**452**	23	**-**	65
Skip Ratio for GAP (r_s)	0.6496	**0.7169**	-	-	0.6063	0.5176	**-**	-	**0.6457**	-
Class-Balanced Accuracy	0.7609	**0.7978**	0.7788	0.7905	0.9870	0.9864	**0.9879**	0.9859	**0.8541**	0.8525
Number of Parameters	2,675,616	**3,471,893**	8,050,761	1,606,250	1,367,073	714,802	**3,704,628**	175,426	**3,226,246**	3,757,059

Supplementary Table 1: Specific hyperparameter values and performance of chosen network architectures. Shown are the optimized hyperparameters for both CNN-GAP and CNN-RNN models and both viral classification and pathogenicity detection tasks. For the viral classification task, models are optimized and evaluated to discriminate between bacteria, eukaryotic viruses, and prokaryotic viruses with two-hour and six-hour runs. 2h and 6h represent the training time allocated to each run during the hyperparameter optimization, where the 6h optimization was warm-started using the results of the 2h optimization. For the pathogenicity detection task, only two-hour runs are optimized. The evaluation was performed separately for sequence lengths of 150 nt (viral detection), 10,000 nt (viral detection), and for the pathogenicity classification task. The class-balanced accuracy was evaluated on a test set not seen during optimization. The best architectures in terms of class-balanced accuracy are shown in bold for all sequence length values.

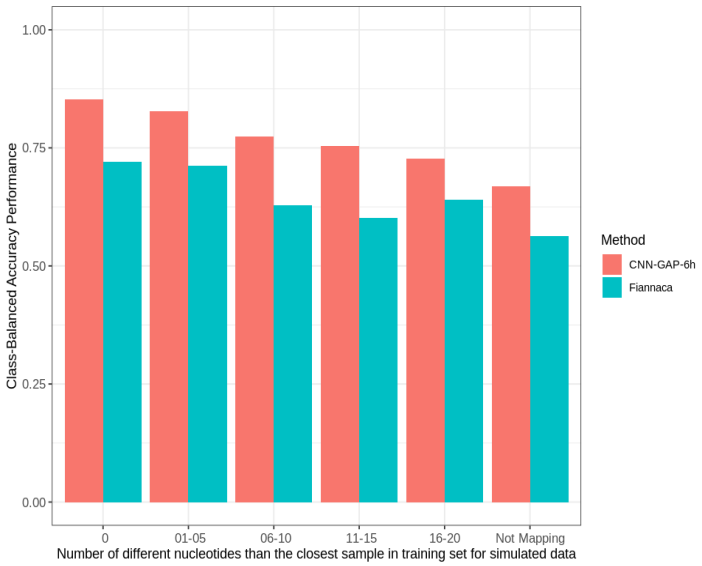
Model/Baseline	Balanced Accuracy	Length (nt)
Bowtie2[1]	71.7%	150
Kraken2[2]	76.0%	
Best ML baseline (Fiannaca[3])	75.0%	
Optimized model	**79.8%**	
Minimap2	68.1%	10k
Kraken2[2]	89.4%	

Best ML baseline (Fiannaca[3])	98.6%	
Optimized model	**98.8%**	

Supplementary Table 2: Balanced accuracy of baselines for the viral classification task, including non-machine learning baselines. The optimized GenomeNet-Architect model outperforms all baseline models. The bold values represent the best models.



Supplementary Figure 1: Model performance stratified similarity groups based on the closest sample in the training set via Bowtie2[1] alignment. We binned the samples based into the following groups: exact match (alignment) of a subsample from the test set to the combined training dataset (all groups); 1-5, 6-10, 11-15, and 16-20, and differences of more than 21 nucleotides (nt), respectively. The final bin denotes samples from the test set that had no alignment at all to the training database, denoted as not aligned (NA) group. We processed these sets from the test datasets using methods to the best-performing baseline Fiannaca and the optimized method CNN-GAP-6h. While by definition, all samples in the NA group fail to be processed by the alignment-based baseline, our optimized model CNN-GAP-6h outperformed the best-performing baseline model Fiannaca[3] by 8.6 percentage points. Furthermore, our optimized model showed a better performance compared to Fiannaca[3] in all bins. When tested simulated reads that have an Illumina error profile, our optimized models outperform Fiannaca by 10.5 percentage points.



Supplementary Figure 2: Model performance stratified similarity groups based on the closest sample in the training set via Bowtie2[1] alignment for simulated data. To ensure the robustness and applicability of our model to real sequencing scenarios, we conducted experiments with synthetic reads that included simulated error profiles. These synthetic reads were generated using the NGSNGS[4] tool, a tool designed to simulate sequencing reads with realistic error patterns. This approach allowed us to model the types of inaccuracies typically encountered in high-throughput sequencing technologies, such as substitutions, insertions, and deletions, thereby providing a more challenging and realistic test environment for our model.

Supplementary Note 1: Longer Optimization Runs. The optimization procedure can be restarted several times with increasing values of t (training time of each run in hyperparameter optimization), using all previous points as warm start data. After setting the training time of the models to 2 hours and then to 6 hours in the hyperparameter optimization, we further increased the training time to 20 hours. This optimization evaluated 528 more configurations, added 33.1 days of training, and thus increased the total hyperparameter optimization duration by 170%. However, we did not observe any increase in the accuracy performance for the sequence length of 150. For the sequence length of 10,000, the class-balanced misclassification error rate decreased from 1.21% to 1.11%, while the number of parameters also decreased from 3.7 million to 0.8 million (CNN-RNN-2h vs CNN-RNN-20h). However, we do not find this increase compelling, as a 0.1 percentage point change is rather small, considering the substantial increase in total optimization run-time.

Supplementary References

1. Langmead, B. & Salzberg, S. L. Fast gapped-read alignment with Bowtie 2. *Nat. Methods* **9**,

357–359 (2012).

2. Wood, D. E. & Salzberg, S. L. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome Biol.* **15**, 1–12 (2014).
3. Fiannaca, A. *et al.* Deep learning models for bacteria taxonomic classification of metagenomic data. *BMC Bioinformatics* **19**, 198 (2018).
4. Henriksen, R. A., Zhao, L. & Korneliussen, T. S. NGSNGS: next-generation simulator for next-generation sequencing data. *Bioinformatics* **39**, (2023).

5.2 Neural Architecture Search for Genomic Sequence Data

Contributing Article

Scheppach A, Gündüz HA, Dorigatti E, Münch PC, McHardy AC, Bischl B, Rezaei M, Binder M (2023). "Neural Architecture Search for Genomic Sequence Data." In *2023 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, pp. 1–10. doi:10.1109/CIBCB56990.2023.10264875

Note: Due to copyright requirements of the publisher (IEEE), the accepted version of the paper is available in this thesis. Thus, the version in the thesis is not the published version.

Declaration of Contributions

Hüseyin Anil Gündüz contributed to this paper as a co-author with the following significant contributions:
Hüseyin Anil Gündüz helped with the design of the experiments to a substantial degree, including proposing the dataset and the baselines of the experiments. Hüseyin Anil Gündüz helped define the search space and develop the proposed methods, which are some of the major contributions of the paper. Hüseyin Anil Gündüz contributed significantly to the writing and presentation of the paper. Hüseyin Anil Gündüz also presented the paper at the IEEE CIBCB 2023 main conference.

Contribution of the coauthors:
Amadeu Scheppach is the first author and main contributor to the project. Amadeu Scheppach designed most parts of the proposed methods, developed the code, and ran the experiments. Emilio Dorigatti helped analyze and interpret the results. Mina Rezaei helped with the supervision of the paper. Martin Binder was the main supervisor of the paper. He helped with the development of the proposed methods and with the choice of baseline methods. He also contributed significantly to the writing of the paper and created diagrams showcasing the architectures. All authors helped with the editing of the paper.

Note: The project was started as Amadeu Scheppach's master thesis, which he completed. The thesis was mainly supervised by Mina Rezaei and Martin Binder, who provided supervision and assistance throughout the entire process. Hüseyin Anil Gündüz, Bernd Bischl, and Philipp C. Münch helped with the supervision during the thesis.

Copyright License © 2023 IEEE. Reprinted, with permission, from Amadeu Scheppach, Hüseyin Anil Günddüz, Emilio Dorigatti, Philipp C. Münch, Alice C. McHardy, Bernd Bischl, Mina Rezaei, Martin Binder, Neural Architecture Search for Genomic Sequence Data, In 2023 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB), August 2023.

In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of LMU Munich's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink. If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

Neural Architecture Search for Genomic Sequence Data

Amadeu Scheppach
Department of Statistics, LMU Munich
Munich, Germany
scheppachamadeu@yahoo.com

Hüseyin Anil Gündüz
Department of Statistics, LMU Munich and
Munich Center for Machine Learning
Munich, Germany
anil.guenduez@stat.uni-muenchen.de

Emilio Dorigatti
Department of Statistics, LMU Munich and
Munich Center for Machine Learning
Munich, Germany
emilio.dorigatti@stat.uni-muenchen.de

Philipp C. Münch
Department for Computational Biology of Infection Research, Helmholtz Centre for Infection Research
Braunschweig, Germany
philipp.muench@helmholtz-hzi.de

Alice C. McHardy
Department for Computational Biology of Infection Research, Helmholtz Centre for Infection Research
Braunschweig, Germany
alice.mchardy@helmholtz-hzi.de

Bernd Bischl
Department of Statistics, LMU Munich and
Munich Center for Machine Learning
Munich, Germany
bernd.bischl@stat.uni-muenchen.de

Mina Rezaei
Department of Statistics, LMU Munich and
Munich Center for Machine Learning
Munich, Germany
mina.rezaei@stat.uni-muenchen.de

Martin Binder
Department of Statistics, LMU Munich and
Munich Center for Machine Learning
Munich, Germany
martin.binder@stat.uni-muenchen.de

***Abstract*—Deep learning has enabled outstanding progress on bioinformatics datasets and a variety of tasks, such as protein structure prediction, identification of regulatory regions, genome annotation, and interpretation of the noncoding genome. The layout and configuration of neural networks used for these tasks have mostly been developed manually by human experts, which is a time-consuming and error-prone process. Therefore, there is growing interest in automated neural architecture search (NAS) methods in bioinformatics. In this paper, we present a novel search space for NAS algorithms that operate on genome data, thus creating extensions for existing NAS algorithms for sequence data that we name Genome-DARTS, Genome-P-DARTS, Genome-BONAS, Genome-SH, and Genome-RS. Moreover, we introduce two novel NAS algorithms, CWP-DARTS and EDP-DART, that build on and extend the idea of P-DARTS. We evaluate the presented methods and compare them to manually designed neural architectures on a widely used genome sequence machine learning task to show that NAS methods can be adapted well for bioinformatics sequence datasets where they outperform human designers.**

***Index Terms*—Deep Learning, Neural Architecture Search, Genomics**

I. Introduction

Deep learning (DL) algorithms have shown promising performance when used in biological research and have been successfully used to tackle computational biology problems [1], [2]. Genomics is one of the largest contributors to large amounts of data in this field because of recent developments in sequencing technology [3]. DL is especially well-suited to tackle learning tasks within genomics, as the data—strings of nucleotides represented by the letters `A`, `C`, `G`, and `T`—are composed of simple individual units that work together to form abstract, high-level features that are translationally invariant. This setting closely resembles natural language processing (NLP) with some crucial differences [4].

When applied to genomic DNA and RNA sequences, DL has shed new light on molecular regulatory patterns and helped uncover potential therapeutic targets for various human diseases [5]–[7]. Predicting the function of non-coding DNA regions is a particularly interesting task, as 98% of the human genome is non-coding and 93% of disease-associated variants lie in these regions [7]. However, most of the DL architectures in this field are manually designed by human experts, a time-consuming process that requires extensive knowledge and experience. A great deal of practical expertise was accumulated in other fields where DL has been used in the last decade, resulting in canonical architectures that are most commonly used in specific prediction tasks, such as ResNet [8] in computer vision, Transformers [9] in NLP, and U-Nets [10] in biomedical image segmentation. Although there has been some research on what kind of models work

This work was funded in part by the German Federal Ministry of Education and Research (BMBF) under GenomeNet Grant No. 031L0199B.

well for genomics [11]–[13], the overall amount of work done in this area is still relatively small. Genome sequence analysis can therefore profit particularly well from methods that automatically optimize DL architectures.

Neural Architecture Search (NAS) algorithms automatically search for the most appropriate DL architecture for a given prediction task, which typically achieve higher performance than architectures handcrafted by human experts [14]. This is challenging, however, since the search space encompasses billions of different architectures whose performance is computationally expensive to obtain [15]. NAS methods, therefore, use various techniques to avoid exhaustive search and to find well-performing architectures with less computational cost. NAS approaches can be defined by three fundamental properties [14]: The *search space*, which defines a parameterized set of neural architectures to be considered, the *search strategy*, which determines how the NAS algorithm decides which elements of the search space to evaluate, and the *performance estimation strategy*, which is used to gauge the performance of proposed network architectures, preferably without fully training each of them from scratch every time.

Out of the three NAS algorithm components listed above, the *search space* is most closely tied to the analyzed data modality. Liu, Simonyan and Yang [16], for example, define two different search spaces, one for convolutional neural networks (CNNs) for images, and one for recurrent neural networks (RNNs) for NLP. However, methods commonly used for DNA sequences tend to contain both CNN and RNN layers [11], suggesting that a joint search space containing both should be used.

Furthermore, since many NAS methods are optimized for computer vision tasks, they need to pay attention to memory efficiency, leading to methods that optimize smaller "proxy" models instead of a final model [16], [17] . Genome sequence data found in common tasks is less memory intensive than image data, however, making such proxy models unnecessary and enabling other kinds of optimizations (see Section III-B).

When choosing a NAS method for a task in genome sequence analysis, a researcher is confronted with two important challenges: first, most NAS algorithms, in particular their search space, are not appropriate for genome sequences, and second, there is no benchmark that shows if and how well a given NAS algorithm works on such data. In this work, we tackle these problems and therefore list our contributions as follows:

- We select a diverse set of popular NAS algorithms—random search, Successive Halving [18], DARTS [16], P-DARTS [19], BONAS [15]—and adapt them to the domain of genome sequence data (Section III-A)
- We provide a natural extension of P-DARTS which we term CWP-DARTS (Continuous Weight Sharing P-DARTS). It avoids training network weights from scratch in each of its optimization stages by keeping model size constant (Sections III-B). This is particularly well-suited for genomics, where memory efficiency is not as relevant as in other settings.
- We propose another extension of P-DARTS, EDP-DARTS (Edge Discarding P-DARTS), which takes the idea behind P-DARTS a step further by progressively discarding not only operations, but also edges (Section III-C).
- We perform an extensive benchmark study on the well-established DeepSEA [5] task encompassing 900 GPU days, where we compare the different NAS methods against each other and against established expert-designed models that are commonly used.

II. Background and Related Works

A. Deep Learning on Genome Sequence Data

DNA molecules, consisting of sequences of Adenine (A), Guanine (G), Cytosine (C), and Thymine (T) nucleotides, are part of all known cellular life on earth. The totality of all DNA sequences within a cell makes up its *genome*, and the field of genomics is concerned with understanding the function and meaning of these sequences. Various machine learning methods have been used for genomics research [1], [2], [5]. Deep learning is particularly fitting for genomics data, because of its ability to handle unstructured data efficiently without the need for preprocessing [20]. The translation-invariant nature of genome sequences makes them well suited for convolutional [21] and recurrent [22] neural networks.

An early application of convolutional neural networks for genome sequence data was the DeepSEA framework [5], which predicted biochemical properties of parts of the human DNA. Their model does multi-task prediction for 919 binary features specifying transcription factor binding, DNase I sensitivity, and histone-mark profiles. Their dataset has become a reference dataset frequently used as a benchmark by subsequent genome sequence models; in this work, we refer to it as the *DeepSEA dataset*. Later, [6] proposed *DanQ*, which is composed of convolutional and recurrent neural networks followed by a fully connected layer and softmax for the DeepSEA task, while [7] showed its *NCNet* architecture, composed of deeper convolutional network with residual connections besides a recurrent layer, can outperform other networks on the DeepSEA dataset.

B. Neural Architecture Search

Neural Architecture Search (NAS) was introduced by [23], who treat the generation of neural architectures as a reinforcement learning problem with the successive configuration of layers as actions and ultimate network performance as rewards for an RNN controller network. While in their method, the controller network is used to generate the entire architecture, [24] innovate by configuring individual network *cells* instead of the entire network: They generate a *normal* cell, which is a network of convolutional layers and operations that ultimately leaves the size of the input constant, and a *reduction* cell, which is generally a different network of layers where the cell input operations have a stride of two, therefore reducing the size of the input. These two cell types are then applied

consecutively and repeatedly, with different numbers of repeats for different datasets. This not only reduces the search space relative to the ultimate network size, but also makes it possible to optimize an architecture on a relatively simple dataset with few repeated cells and fast evaluation time, and scale the result up for a more complex dataset.

Many NAS methods (e.g. [15], [16], [19]) use a search space that is heavily leaning on the one used in [24], which they termed the *NASNet* search space. For this search space, the possible network configurations of both the normal and reduction cells are represented as directed acyclic graphs (DAGs). The DAG for a given cell configuration consists of a predefined number V of *vertices* or *nodes* $x^{(i)}, i \in \{1, \dots, V\}$ that are connected by (directed) edges $(i, j) \in \{1, \dots, V\}^2$. Each vertex stands for a latent representation of the network's input; they are divided into *input* nodes that have their values set from incoming data, *intermediate* nodes, and *output* nodes. Each edge is associated with an operation $o^{(i,j)}(\cdot) \in \mathcal{O}$. The value assigned to $x^{(i)}$ for intermediate nodes is the sum of the operations applied along these edges: $x^{(i)} = \sum_{h \in \text{pred}(i)} o^{(h,i)}\left(x^{(h)}\right)$, where $\text{pred}(i)$ are the predecessors of i in the DAG. While the predecessors of intermediate nodes are learned directly by the NAS methods, the predecessors of the output nodes depend on the rest of the graph; sometimes they are the set of all intermediate nodes, sometimes only the set of all intermediate nodes that otherwise have no outgoing edge.

The original NASNet search space was designed for CNNs and was therefore mostly used for image data. For CNN architecture search, the DAG has two input nodes, and all intermediate nodes have exactly two predecessors. The operations in $\mathcal{O}$ are various convolution and pooling operations, as well as the identity and, for some NAS methods, the *zero* operation. In all methods treated in this work, the final network architecture is formed by sequentially connecting multiple cells, where one input node of cell n is assigned the output of cell $n-1$, and the other input node is assigned the output of cell $n-2$, as illustrated in Fig. 1. The value of the output node is set to the concatenation along the filter dimension of its predecessors in the graph.

While only some methods (e.g. [16]) also use a DAG search space for RNNs, their search space has a slightly different setup than the NASNet search space, first developed in this form by [17], which we will therefore call the *ENAS RNN* search space[1]. The DAG for RNNs has one input node, which is set to the $\tanh$-activated sum of a linear transformation of (i) the input data of the current time step, and (ii) the cell output state of the previous timestep. All intermediate nodes have exactly one predecessor, and the operations in $\mathcal{O}$ are linear transformations followed by various activation functions. The value of the output node is set to the average of its predecessors. Methods using the ENAS RNN search space usually employ variational dropout [25].

[1]While [24] also perform NAS for RNNs, they do not use a DAG for it and have a different search space.

In this work, we combine and adapt the search spaces described above to include networks that are specifically well-suited for genome sequence data. Similarly to many other NAS methods, we use a fixed *macro-architecture* of blocks of cells, where the architectures of the cells themselves are being optimized. However, unlike all other cell-based NAS methods that we are aware of, we combine both CNN and RNN cells in a single architecture, since these hybrid models have so far proven most successful on genome data [11]. The resulting search space is the product space of the individual cell search spaces. The overall macro-architecture, as well as exemplary cell architectures, are shown in Fig. 1. A large variety of methods exist that can use a DAG-based search space and this adaption can be used with many of these. In the following, we shortly describe the NAS methods that we adapted:

Random Search (RS) is the random sampling of configurations that are then fully evaluated on a given dataset. It was shown to be relatively efficient for the related field of hyperparameter optimization [26] due to the low effective dimensionality of this problem. It can easily be used with any given search space.

Successive Halving (SH) [27], also called Sequential Halving [28], was developed for hyperparameter optimization of iterative machine learning methods, which it treats like a non-stochastic multi-armed bandit problem. In this setting, an *arm* corresponds to a hyperparameter or neural architecture configuration, and *pulling* that arm corresponds to training an iterative model for some iterations and evaluating it on a validation set. By sequentially continuing to train a model for more iterations, more information about the configuration in the limit of model convergence is gained. SH is part of the popular Hyperband [29] method.

DARTS (Differentiable ARchiTecture Search, [16]) is a one-shot NAS method that uses a continuous relaxation of the NASNet / ENAS RNN search spaces to be able to perform gradient descent optimization on it. Instead of training models within these discrete search spaces and evaluating them, it trains a one-shot-model that is parameterized by continuous architecture parameters $\boldsymbol{\alpha} = \left\{\alpha^{(\mathbf{i},\mathbf{j})}\right\}$, in addition to conventional neural network parameters $\mathbf{w}$. $\alpha^{(\mathbf{i},\mathbf{j})}$ is a vector of length $|\mathcal{O}|$ and determines, for edge (i, j), the relative weight of each operation in $\mathcal{O}$ being performed on this edge. In this one-shot model, every edge (i, j) for $i < j$ is present in the DAG and performs the operation $\bar{o}^{(i,j)}(\cdot) = \text{softmax}(\alpha^{(\mathbf{i},\mathbf{j})}) \cdot \mathbf{o}(\cdot)$, with $\mathbf{o}(\cdot)$ the vector of all operations in $\mathcal{O}$. DARTS tackles the bilevel optimization problem where the architecture parameters $\boldsymbol{\alpha}$ are optimized for the outer loss $\mathcal{L}_{\text{val}}(\mathbf{w}^*(\boldsymbol{\alpha}), \boldsymbol{\alpha})$ on validation data, given that the model parameters $\mathbf{w}^*(\boldsymbol{\alpha})$ minimize the inner (training) loss $\mathcal{L}_{\text{train}}(\mathbf{w}, \boldsymbol{\alpha})$ on training data. After the DARTS optimization is terminated, a result architecture is produced by dropping all but the top two (for the NASNet search space) or top one (for the ENAS RNN search space) incoming edges for each intermediate node, based on the largest $\text{softmax}(\alpha^{(\mathbf{i},\mathbf{j})})$-values (ignoring the coefficient for the *zero*-operation). For each of the remaining

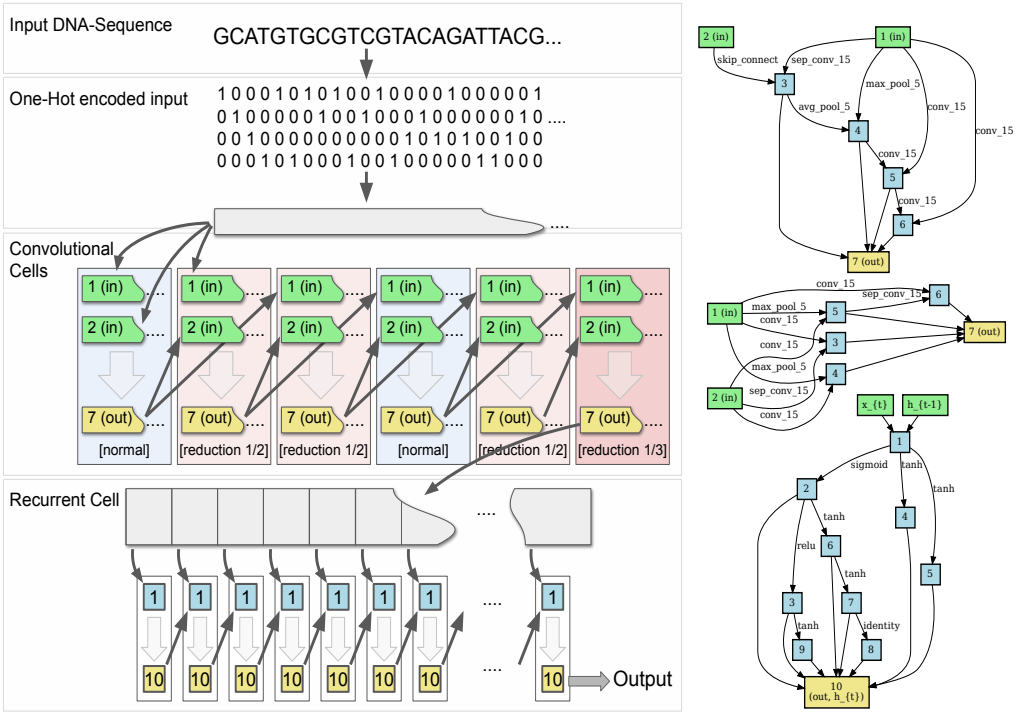


Fig. 1. Neural Architectures. **Left**: Macro-Architecture, i.e. the way in which cells are connected. The input sequence of length n is one-hot encoded as a $4 \times n$ matrix and fed through an initial convolution layer. The data is then successively fed through convolutional cells, which get their input from the two preceding cells. Here, the large transparent arrows represent data processing according to the cell architectures. Reduction cells are used whenever a stride length of 2 or 3 is used to reduce the sequence length. The output of the last convolutional cell is fed to the recurrent cell as a sequence.
Right: Overall best performing cells (learned by Genome-DARTS algorithm). From top to bottom: normal cell, reduction cell, recurrent (RNN) cell. For the normal and reduction cell, nodes 1 and 2 are input nodes and node 7, the output node, is set to the concatenated value of all intermediate nodes. In the recurrent cell, node 1 gets its input from both the input *x_{t}* as well as the output of the previous time step, *h_{t-1}*. The value of the output node 10 is set to the mean of all intermediate nodes.

edges, the operation (excluding *zero*) with the largest $\alpha^{(\mathrm{i},\mathrm{j})}$ is then used.

The one-shot model used in DARTS needs to contain the weights (and their gradients) of all possible operations $\mathcal{O}$ for all edges $(i, j), i < j$ and therefore needs large amounts of memory. It is therefore common to perform architecture search with a relatively small number of cells and possibly even on an easier "proxy"-task that is different from the target task for which the network is ultimately used. The performance of the final model is evaluated with the resulting cell architectures, but with a larger number of cells and, if applicable, on the target task.

P-DARTS (Progressive DARTS, [19]) is an extension of DARTS that tackles what is described as the *optimization gap*: The large difference between the model during optimization (fewer cells, possibly different data, weights optimized in context of the one-shot model) and evaluation (more cells, actual evaluation task, weights trained on resulting model architecture). P-DARTS therefore divides optimization into k *stages*, where the initial stage is equivalent to DARTS optimization, but subsequent stages approximate the final model architecture more closely. This is done by by dropping some of the operations for each edge $\mathcal{O}_k^{(i,j)}$ so that $\left|\mathcal{O}_k^{(i,j)}\right| = O_k$ with monotonically decreasing schedule on number of operations O_k. In later stages, fewer operations are considered, and the decreased need of GPU memory for an individual cell makes it possible to increase the number of cells after each optimization stage. However, since this changes the overall topology of the model, it becomes necessary to re-initialize model-weights $\mathbf{w}$ after each stage. A second innovation introduced in P-DARTS is what [19] call *search space regularization*, which aims to reduce the number of identity-operations in the final model. This is because of the observation that identity operations are chosen disproportionately since they have unnaturally large gradients at the beginning of the optimization. P-DARTS uses a dropout mechanism for this operation with a dropout rate that slowly decays during optimization. It furthermore introduces a hyperparameter M that determines the maximum number

of edges with identity operations in the final model; if the number of identity edges in the naively constructed model exceeds M, they set the $\boldsymbol{\alpha}$-components corresponding to identity connections to 0 for all but the largest M occurrences.

BONAS (Bayesian Optimization NAS, [15]) is a NAS method that, like DARTS, is a one-shot-model based approach, where multiple candidate architectures are trained simultaneously. However, instead of gradient based methods, it relies on Bayesian Optimization [30] for generating these candidates. Instead of using a Gaussian Process as surrogate model, as is often done, BONAS uses a graph-convolutional network [31] to create a graph-embedding that represents the DAG and assignment of operations $o^{(i,j)}$ of a given neural architecture. BONAS performs Bayesian linear regression on that embedding; the predicted mean and variance are then used with the UCB acquisition function to select multiple candidate solutions. These candidates are combined into a single one-shot model and trained together. The trained weights are then used to evaluate the performance of the candidate networks for the optimization.

C. NAS for Genome Data

There have been a few recent attempts to apply existing NAS methods on genome sequence data. [32] introduced BioNAS where they modified the NAS of [23] for protein binding prediction. They consider convolution, pooling, and fully-connected layers as search space. AMBER [33] used the Efficient Neural Architecture Search (ENAS, [17]) algorithm on genome sequences and evaluated it on the DeepSEA tasks. The AMBER search space includes convolution, dilated, max-pooling, and ReLU activation. AMBIENT [34] builds on top of that and uses dataset-characterizing meta-features for faster convergence. AMBIENT's search space includes convolution and as well as recurrent LSTM layers. Unlike our method, however, they do not optimize the architecture of the recurrent network itself.

Similarly to these methods, we study NAS algorithms that operate on genome data. Unlike these works, we provide a comprehensive study on the search space and explore a NAS algorithm with both convolutional and recurrent search space. Specifically, our work differs in that (i) we have adapted a variety of different NAS methods (DARTS, P-DARTS, BONAS, SH, and RS; see above) for genomic sequence data, that (ii) Our search space includes both the convolutional and the recurrent architecture, and that (iii) we introduce two novel extensions of P-DARTS named CWP-DARTS and EDP-DARTS, which we also adapt for biological sequences.

III. Methods

A. GenomeNAS Search Space

The GenomeNAS search space is created by combining both the NASNet search space for CNNs and the ENAS search space for RNNs. The NASNet search space here consists of the configurations of two disjoint DAGs that describe each the "normal" and a "reduction" convolutional cell; the ENAS RNN search space is the configuration space of the DAG describing the recurrent cell. The operations used for the convolutional cells differ necessarily from the operations used in other work focusing on image data, since they are applied to one-dimensional data, and are chosen to have potentially relatively large receptive field sizes because DNA data may have relatively long-range interactions [35].

The operations $\mathcal{O}_{\text{cnn}}$ used for the convolutional cells are:

- average pooling (size 5)
- max pooling (size 5)
- convolution (size 15)
- depth-wise separable convolution (size 9 and size 15)
- dilated depth-wise separable convolution (size 9 and size 15, both dilation 2)
- identity (i.e. skip connection)
- *zero*-operation

. The non-dilated depth-wise separable convolutions are applied twice to the hidden state as in [24], and as in [16] each convolution operation is preceded by ReLU activation and followed by batch normalization. The operations $\mathcal{O}_{\text{rnn}}$ used for the recurrent cell are:

- tanh
- sigmoid
- ReLU
- identity (i.e. skip connection)
- *zero*-operation

. Note that for all cells, the *zero*-operation is only used by DARTS-based methods during model search and is never part of a final network. We follow [16] in that the value of the output node of each cell is the concatenation (CNN) or average (RNN) of *all* intermediate nodes within that cell.

As in [24], the reduction convolutional cells are used to decrease the input size, while the normal convolutional cells keep the input size constant. However, since input lengths for genome sequence tasks can be thousands of nucleotides, it is beneficial to have more aggressive input size reduction, thereby increasing the effective receptive field size of later convolutional operations. We therefore typically use more reduction cells than used in [24], and also allow reduction cells to have a step size greater than 2.

The overall network built from the individual cells consists of normal and reduction convolutional cells stacked as in [24] that are followed by a single recurrent cell. Both the convolutional and the recurrent network are then optimized jointly.

The GenomeNAS search space is used together with the methods described in Section II-B to form the following methods: **Genome-RS** (performing random search over the GenomeNAS space), **Genome-SH** (performing successive halving), **Genome-DARTS** (based on [16]), **Genome-P-DARTS** (based on [19]), and **Genome-BONAS** (based on [15]). Because neural networks for genome sequence data do not tend to be as deep as for image data, we limit the number of convolutional cells in the final model. The consequence of this is that, for Genome-DARTS, the one-shot model used during architecture

search can have the same size as the model used on the ultimate evaluation task while still fitting in GPU memory, reducing the optimization gap. This is also true for Genome-P-DARTS, which, unlike the original P-DARTS, does not (need to) change the number of cells between optimization stages.

B. CWP-DARTS

The original P-DARTS implementation [19] uses a different number of cells for each optimization stage, which makes it necessary to re-initialize the network weights $\mathbf{w}$ after each stage. This means that the architecture search will, at the beginning of each stage, proceed for a while with a one-shot model that is relatively distant from an optimal parameterization $\mathbf{w}^*$, leading to wasted optimization steps.

We, therefore, propose the *CWP-DARTS* (Continuous Weight sharing Progressive DARTS) method, which does not change the number of cells between optimization stages, and which therefore makes it possible to continue using the network weights when starting a new optimization stage. The other innovations of P-DARTS, namely the continuous reduction of the search space over operations $\mathcal{O}_k^{(i,j)}$ for each stage k, as well as the search space regularization, can be used as before. If the final model evaluation task involves a model with more cells than used during one-shot model optimization, then this method slightly increases the optimization gap compared to P-DARTS. It is traded for more efficient optimization within the optimization stages.

In the context of genome data using the GenomeNAS search space, we refer to this method as **Genome-CWP-DARTS**.

C. EDP-DARTS

The original P-DARTS method [19] closes the optimization gap to the final model by (among others) reducing the search space over operations $\mathcal{O}_k^{(i,j)}$ for each stage k, thereby "progressively" reducing the search space towards the final result and at the same time making the one-shot model more similar to the ultimate single result architecture. However, even with this approach, the one-shot model is still a DAG where most intermediate nodes have more incoming edges than they will in the final model since the edges are pruned and each intermediate node retains only one (RNN) or two (CNN) edges at the end.

We, therefore, propose *EDP-DARTS* (Edge Discarding P-DARTS), which builds on the idea of P-DARTS and expands it by also pruning edges between optimization stages. This is done by specifying a schedule E_k for the maximum number of edges that each intermediate node may have at most at the end of optimization stage k. After each stage, all edges that are not among the best E_k edges based on their largest softmax$(\alpha^{(\mathbf{i},\mathbf{j})})$-value (where the component of the *zero* operation is ignored) are dropped. This mirrors the method of selection for edges to use at the end of the optimization. This way, the idea behind P-DARTS is taken a step further and the optimization stages narrow the search space down to the final model in an even more principled way. In the context of genome data using the GenomeNAS search space, we term this method **Genome-EDP-DARTS**.

IV. Experiments

Dataset and Methods We performed all of our experiments on the DeepSEA dataset, which is a popular and widely used non-coding DNA sequence benchmark. The input is represented by a one-hot encoded 1000-bp DNA sequence which is used to predict 919 binary chromatin features in a multi-label prediction setting. The data is used as provided by [5][2] under a CC Attribution 3.0 license, who also provide splits into training, validation and test data. In all our experiments, we limit the size of one epoch to 3000 samples which are randomly drawn without replacement from the full dataset. Different samples from the DeepSEA data are drawn in each epoch, for each experimental replicate and for each method. The DeepSEA architecture [5], DanQ [6], and NCNet [7] are popular deep learning architectures that we investigate as human-designed baselines and compare them to the architectures chosen by NAS algorithms. Experiments were conducted each on single Nvidia A100 40GB accelerator cards on an Nvidia DGX A100 Server.

Experimental Design We closely follow the NAS procedure suggested by [16] in our experiments. It consists of three defined steps: architecture *search*, architecture *selection*, and architecture *evaluation*. For architecture search, the aim is to learn the optimal architecture by running the NAS algorithms for a pre-defined number of epochs. After the search phase is completed, we obtain the final architecture. Each final architecture is then trained for a specific number of replications on the test set and average results are reported. The overall experiments proceed as follows:
(i) *Preliminary hyperparameter optimization*: The learning rate and dropout hyperparameters of the RNN part were optimized in a preliminary step; see Appendix B for details.
(ii) *Search phase*: Each NAS algorithm is run four times, and four final architectures are obtained.
(iii) *Selection phase*: Each of these four final architectures is trained for 50 epochs and the validation performance of each architecture is evaluated. The architecture that achieves the highest performance on the validation data is chosen as the final architecture of the NAS algorithm.
(iv) *Evaluation phase*: The performance of each NAS algorithm is evaluated by training the selected architecture from scratch for 50 epochs. We report the performance on the test set.

We perform all runs with batch size of 100 as used by DeepSEA, DanQ, and NCNet. For most hyperparameter settings of the Genome-X NAS methods we use the settings provided by the original authors, see Appendix A for more details on chosen hyperparameters.

V. Results and Discussion

Selection phase There was about a 10% difference in median F1 score of the architecture found by the random

[2]http://deepsea.princeton.edu/media/code/deepsea_train_bundle.v0.9.tar.gz

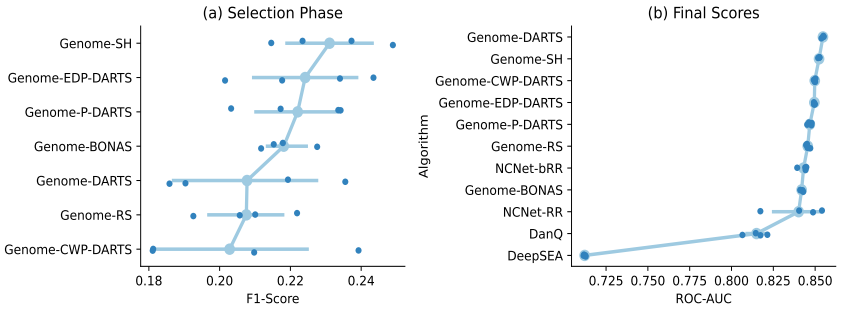


Fig. 2. Performance of algorithms in the two phases of the evaluation. (a) selection phase: validation set performance of models, which is used to select models for the evaluation phase. Architecture search was performed four different times, each dot represents the performance of a separate NAS run. For each method, the architecture with the highest performance was chosen to be evaluated in the evaluation phase. (b) evaluation phase: The model architecture selected in the selection phase was trained from scratch four separate times and evaluated on the test set. Each dot represents the performance of the selected architecture for each NAS method, trained with different weight initializations.

TABLE I
COMPARISON OF PERFORMANCE VALUES OBTAINED WITH VARIOUS METHODS. PR-AUC IS THE PRECISION-RECALL-AUC [36] AS RECOMMENDED BY [6], ROC-AUC THE AREA UNDER THE RECEIVER OPERATOR CHARACTERISTIC ([37], MORE IS BETTER); VALUES ARE AVERAGED OVER ALL 919 LABELS. TRAINING WAS DONE WITH SUBSAMPLED EPOCHS, SO VALUES MAY NOT BE COMPARABLE WITH OTHER VALUES IN THE LITERATURE.

	Method	Params. (M)	PR-AUC	ROC-AUC	Train Time (GPU-Days)
Baselines	DeepSEA	52.84	6.44	71.22	-
	DanQ	46.93	17.27	81.5	-
	NCNet-RR	57.58	22.15	84.01	-
	NCNet-bRR	47.69	22.05	84.31	-
Ours	Genome-RS	27.01	21.90	84.56	10.22
	Genome-SH	26.86	23.44	85.23	9.91
	Genome-DARTS	29.22	**23.90**	**85.47**	10.21
	Genome-P-DARTS	27.08	22.24	84.68	11.61
	Genome-BONAS	26.25	21.20	84.19	24.03
	Genome-CWP-DARTS	26.54	22.97	84.98	8.62
	Genome-EDP-DARTS	26.73	22.87	84.95	10.45

baseline and the best-performing method, which was Genome-SH, closely followed by Genome-DARTS (Fig. 2a). In certain cases the variability among different architecture search runs of the same method was considerable. Different runs of Genome-CWP-DARTS resulted in both worst- and third best-performing architectures, with a gap of 25% in F1 score. This algorithm was also the worst in terms of median performance, which was even lower than a purely random search. On the contrary, the best and worst architectures found by Genome-BONAS were separated by only about 8%. Despite producing the best of the worst architectures, Genome-BONAS could only increase its performance by 0.02 points.

Evaluation phase Except for Genome-BONAS, all Genome-NAS consistently outperformed the baseline algorithms, as well as random search and successive halving (Table I and Fig. 2b). Random search achieved an average PR-AUC [36] score of 21.90 and an average ROC-AUC score [37] of 84.56, which was comparable to the results from NCNet-RR [7] and NCNet-bRR [7] and better than the results of the DeepSEA [5] and the DanQ algorithm [6]. This suggests an advantage of our novel search space, which combines a convolutional and a recurrent DAG. The best result was achieved by Genome-DARTS, with Genome-SH within 0.5 percentage points in term of PR-AUC. The best architecture is shown in Fig. 1. With an average PR-AUC score of 21.20 and an average ROC-AUC score of 84.19, the final architecture of Genome-BONAS is the worst-performing architecture. Our Genome-EDP-DARTS algorithm performed better than the original Genome-P-DARTS, which suggests that we achieved a further decrease of the optimization gap between search and evaluation [19].

Runtime The runtime of all GenomeNAS algorithms was between eight and 10 GPU-days, except for Genome-BONAS which required more than 24 GPU-days on average. In spite of the superior performance, the final networks found by the GenomeNAS family of algorithms were considerably smaller compared to the baseline models, respectively 26-29 and 46-57 million parameters. However, such networks required on average 156 minutes per epoch to train, compared to 7, 13, 34, and 52 minutes for DeepSEA [5], DanQ [6], NCNet-bRR and NCNet-RR, and converged after about 40 training epochs.

Discussion Neural architecture search performs very well

on genome data, outperforming prior hand-crafted models [6], [7] while also being significantly smaller. Successive Halving performs surprisingly strong considering its simplicity. It is also noticeable that P-DARTS, which was specifically designed to outperform DARTS, does not work better in this setting. This is likely due to the fact that one of its main advantages, the shrinking of the optimization gap, is diminished in this setting where the size of the network does not change between architecture search and evaluation. It is also likely that, since the more complicated NAS methods have a variety of hyperparameters on their own, they have so far been tuned for more popular deep learning tasks such as computer vision. This likely explains some part of the relatively weak performance of the more complex algorithms and also creates hope that these algorithms could be tweaked to work even better in this field.

Limitations Our work has shown that NAS methods can easily be adapted to the field of genome sequence data, where it outperforms human-designed baseline models. However, we only conducted a benchmark experiment on a single dataset that exemplifies the data and tasks in this field. It is likely that for other genome sequence tasks that predict other features, the results would be slightly different.

Broader impact statement NAS algorithms promise to provide a better design and a better performance for deep learning applications since they help to minimize the time and cost involved in model design. Although on the surface, NAS methods are expensive, they are likely more systematic and therefore more efficient than human experimentation for architecture design. We consider deep learning on genomic data a promising field that has the potential to bring new insights for biomedical research and therefore ultimately benefit society. Neural networks optimized for genomic data modalities could e.g. benefit outbreak detection due to the optimization of taxonomic assignments [38] and could contribute to the design of novel therapeutica due to the modeling of protein folding [39].

VI. Conclusion

In this paper, we have adapted a diverse set of popular NAS algorithms (random search, Successive Halving, DARTS, P-DARTS, BONAS) to the field of genomic sequence analysis. Moreover, we have extended the P-DARTS algorithm in two novel ways, by introducing continuous weight sharing (CWP-DARTS), and edge discarding (EDP-DARTS). The new algorithms can generate accurate combined CNN and RNN architectures that are capable of modeling genomic sequences.

It can be summarized that our uniquely designed search space works very well, as all GenomeNAS algorithms showed strong performance on the DeepSEA task and most outperformed current state-of-the-art baseline models as well as randomly sampled models. We conclude that our unique combination of DAG search spaces for both CNN and RNN cells has great potential for further research.

Appendix A
Appendix: Details about Hyperparameter Settings

Most hyperparameters were set as in the original publications of the respective methods. We use momentum SGD to ensure that our optimizer is similar to the original DARTS; the CNN weights are optimized with momentum set to 0.9 and weight decay set to 3×10^{-4}; the weights of the recurrent part of the network are optimized without momentum and with weight decay set to 5×10^{-7}.

- **Genome-RS**: To get an optimization runtime similar to Genome-DARTS, 20 architectures are sampled randomly and trained for 7 epochs before being evaluated.
- **Genome-SH**: An initial sample of 25 architectures is sampled. Halving happens every 3 epochs by discarding the worse half of models by performance.
- **Genome-DARTS**: The $\boldsymbol{\alpha}$ parameters are optimized as in DARTS with Adam, with initial learning rate 3×10^{-3}, weight decay 1×10^{-3}, momentum parameters $\beta = (0.9, 0.999)$. Weight clipping of size 0.25 is applied for stability. As in DARTS, the architecture search runs for 50 epochs.
- **Genome-P-DARTS**: As in P-DARTS, the initial dropout probability of skip-connects for search space regularization is set to 0.1 in the first stage, 0.2 in the second stage and 0.3 in the third stage. The one-shot model is trained for 25 epochs in each stage, where in the first 10 epochs, only network weights are trained. Network and architecture weights are optimized jointly in the last 15 epochs of each stage. As in P-DARTS, $\boldsymbol{\alpha}$ are optimized with Adam with learning rate 0.0006, weight decay 0.001 and momentum $\beta = (0.5, 0.999)$.
- **Genome-CWP-DARTS**: Unlike Genome-P-DARTS, only the first stage is trained for 25 epochs with the first 10 epochs network parameter optimization only; the other two stages optimize only 15 epochs, optimizing network and architecture weights jointly. This reduces the overall number of epochs from 75 to 55.
- **Genome-EDP-DARTS** is optimized as Genome-P-DARTS. The maximum number of input edges kept for all vertices is 4 after the first stage, 3 after the second stage, and 2 after the third (final) stage for the CNN networks and 5 after the first stage, 3 after the second stage, and 1 after the third (final) stage for the RNN network.
- **Genome-BONAS** starts with 60 randomly sampled architectures trained for 60 epochs. Then, two BO iterations are executed where 1000 randomly sampled architectures are evaluated by the surrogate model and the top 60 networks by UCB are proposed for evaluation, again for 60 epochs.

During the model search, the batch size is set to 64, and epochs are limited to random subset of the entire data of 2000 steps (i.e. 64×2000 samples). The final training of architectures is done with 3000 steps and batch size 100.

APPENDIX B
APPENDIX: PRELIMINARY HYPERPARAMETER OPTIMIZATION

DARTS and ENAS run experiments on image classification and language modeling tasks [16], [17]. For image classification using CNNs, an initial learning rate of 0.025 is used, while the RNNs used for language modeling have an initial learning rate of 20. Due to high difference between the learning rates, we decided to use different learning rates for the convolutional and recurrent parts of our search space.

Initial experiments showed low sensitivity to the learning rate of the convolutional part and high sensitivity to the learning rate of the recurrent part. We therefore decided to run a preliminary grid search to optimize the learning rate and variational dropout rate of the recurrent network. We evaluated the Genome-DARTS method for recurrent learning rate values of 2, 8, and 12, and dropout rates (input / "embedding" dropout, recurrent dropout) of (0,0), (0.1,0.05), and (0.3,0.1). The best-performing settings that were chosen were learning rate 8, input dropout 0.1, and recurrent dropout 0.05.

We performed a grid search to find the best parameters for the learning rate of the RNN layers and variational dropout rate pairs for cell input and hidden nodes. The searched space for the learning rate is 2, 8 an 12; and the variational dropout rate of cell input and hidden nodes are (0,0), (0.1,0.05) and (0.3,0.1). The best-performing settings were 8, 0.1, and 0.05 respectively and we used these hyperparameters for the rest of the experiments.

REFERENCES

[1] M. AlQuraishi, "Alphafold at casp13," *Bioinformatics*, vol. 35, no. 22, pp. 4862–4865, 2019.

[2] C. Cao, F. Liu, H. Tan, D. Song, W. Shu, W. Li, Y. Zhou, X. Bo, and Z. Xie, "Deep learning and its applications in biomedicine," *Genomics, proteomics & bioinformatics*, vol. 16, no. 1, pp. 17–32, 2018.

[3] B. Segerman, "The most frequently used sequencing technologies and assembly methods in different time segments of the bacterial surveillance and refseq genome databases," *Frontiers in cellular and infection microbiology*, p. 571, 2020.

[4] A. Wahab, H. Tayara, Z. Xuan, and K. T. Chong, "Dna sequences performs as natural language processing by exploiting deep learning algorithm for the identification of n4-methylcytosine," *Scientific reports*, vol. 11, no. 1, pp. 1–9, 2021.

[5] J. Zhou and O. G. Troyanskaya, "Predicting effects of noncoding variants with deep learning–based sequence model," *Nature methods*, vol. 12, no. 10, pp. 931–934, 2015.

[6] D. Quang and X. Xie, "Danq a hybrid convolutional and recurrent deep neural network for quantifying the function of dna sequences," *Nucleic acids research*, vol. 44, no. 11, p. 107, 2016.

[7] H. Zhang, C.-L. Hung, M. Liu, X. Hu, and Y.-Y. Lin, "Ncnet: Deep learning network models for predicting function of non-coding dna," *Frontiers in genetics*, vol. 10, p. 432, 2019.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[10] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[11] A. Trabelsi, M. Chaabane, and A. Ben-Hur, "Comprehensive evaluation of deep learning architectures for prediction of dna/rna sequence binding specificities," *Bioinformatics*, vol. 35, no. 14, pp. i269–i277, 2019.

[12] H. R. Hassanzadeh and M. D. Wang, "Deeperbind: Enhancing prediction of sequence specificities of dna binding proteins," in *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 2016, pp. 178–183.

[13] F. Runge, D. Stoll, S. Falkner, and F. Hutter, "Learning to design rna," *arXiv preprint arXiv:1812.11951*, 2018.

[14] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 1997–2017, 2019.

[15] H. Shi, R. Pi, H. Xu, Z. Li, J. Kwok, and T. Zhang, "Bridging the gap between sample-based and one-shot neural architecture search with bonas," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1808–1819, 2020.

[16] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[17] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *International conference on machine learning*. PMLR, 2018, pp. 4095–4104.

[18] R. Schmucker, M. Donini, M. B. Zafar, D. Salinas, and C. Archambeau, "Multi-objective asynchronous successive halving," *arXiv preprint arXiv:2106.12639*, 2021.

[19] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive darts: Bridging the optimization gap for nas in the wild," 2020.

[20] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[21] L. Torada, L. Lorenzon, A. Beddis, U. Isildak, L. Pattini, S. Mathieson, and M. Fumagalli, "Imagene: a convolutional neural network to quantify natural selection from genomic data," *BMC bioinformatics*, vol. 20, no. 9, pp. 1–12, 2019.

[22] Z. Shen, W. Bao, and D.-S. Huang, "Recurrent neural network for predicting transcription factor binding sites," *Scientific reports*, vol. 8, no. 1, pp. 1–10, 2018.

[23] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *5th International Conference on Learning Representations (ICLR), Conference Track Proceedings*, 2017.

[24] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[25] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks," *Advances in neural information processing systems*, vol. 29, 2016.

[26] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization." *Journal of machine learning research*, vol. 13, no. 2, 2012.

[27] K. Jamieson and A. Talwalkar, "Non-stochastic best arm identification and hyperparameter optimization," in *Artificial intelligence and statistics*. PMLR, 2016, pp. 240–248.

[28] Z. Karnin, T. Koren, and O. Somekh, "Almost optimal exploration in multi-armed bandits," in *International Conference on Machine Learning*. PMLR, 2013, pp. 1238–1246.

[29] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6765–6816, 2017.

[30] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, vol. 25, 2012.

[31] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *International Conference on Learning Representation, ICLR*, 2016.

[32] Z. Zhang, L. Zhou, L. Gou, and Y. N. Wu, "Neural architecture search for joint optimization of predictive power and biological knowledge," *arXiv preprint arXiv:1909.00337*, 2019.

[33] Z. Zhang, C. Y. Park, C. L. Theesfeld, and O. G. Troyanskaya, "An automated framework for efficiently designing deep convolutional neural networks in genomics," *Nature Machine Intelligence*, vol. 3, no. 5, pp. 392–400, 2021.

[34] Z. Zhang, E. M. Cofer, and O. G. Troyanskaya, "Ambient: accelerated convolutional neural network architecture search for regulatory genomics," *bioRxiv*, 2021.

[35] L. Koumakis, "Deep learning models in genomics; are we there yet?" *Computational and Structural Biotechnology Journal*, vol. 18, pp. 1466–1473, 2020.

[36] K. Boyd, K. H. Eng, and C. D. Page, "Area under the precision-recall curve: Point estimates and confidence intervals," in *Machine Learning and Knowledge Discovery in Databases*, H. Blockeel, K. Kersting, S. Nijssen, and F. Železný, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 451–466.

[37] K. H. Zou, A. J. O'Malley, and L. Mauri, "Receiver-operating characteristic analysis for evaluating diagnostic tests and predictive models," *Circulation*, vol. 115, no. 5, pp. 654–657, 2007.

[38] J. N. Nissen, J. Johansen, R. L. Allesøe, C. K. Sønderby, J. J. A. Armenteros, C. H. Grønbech, L. J. Jensen, H. B. Nielsen, T. N. Petersen, O. Winther *et al.*, "Improved metagenome binning and assembly using deep variational autoencoders," *Nature biotechnology*, vol. 39, no. 5, pp. 555–560, 2021.

[39] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko *et al.*, "Highly accurate protein structure prediction with alphafold," *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.

6 Contributions to Uncertainty Quantification and Calibration in Genomics

6.1 Uncertainty Quantification for Deep Learning Models Predicting the Regulatory Activity of DNA Sequences

Contributing Article

Gündüz HA, Giri S, Binder M, Bischl B, Rezaei M (2023). "Uncertainty Quantification for Deep Learning Models Predicting the Regulatory Activity of DNA Sequences." In *2023 International Conference on Machine Learning and Applications (ICMLA)*, pp. 566–573. doi: 10.1109/ICMLA58977.2023.00084

Note: Due to copyright requirements of the publisher (IEEE), the accepted version of the paper is available in this thesis with added authors, acknowledgements, and a footnote stating the equal contributions of the co-first authors. Thus, the version in the thesis is not the published version.

Declaration of Contributions

Hüseyin Anil Gündüz and Sheetal Giri share the first authorship of this paper and their contributions are overall equal. Hüseyin Anil Gündüz's significant contributions can be described as follows: Hüseyin Anil Gündüz is a main contributor to the project. The paper was mainly written by Hüseyin Anil Gündüz. Hüseyin Anil Gündüz co-supervised the project and contributed significantly to the design and evaluation of the experiments and the presentation and interpretation of the results. Hüseyin Anil Gündüz also presented the paper at the ICMLA 2023 main conference as a full paper and at the ISMB/ECCB 2023 as an abstract.

Contribution of the coauthors: Sheetal Giri contributed to the design, code development, running, and evaluation of the experiments as a main contributor. Sheetal Giri also wrote some parts of the paper. Mina Rezaei was the main supervisor of the project. Martin Binder provided support with running computational jobs on a batch processing system.

All authors helped with the editing of the paper.

Note: The project was started as Sheetal Giri's master thesis, which she completed. The thesis was supervised by Mina Rezaei and Hüseyin Anil Gündüz, who provided supervision and assistance throughout the entire process.

Copyright License © 2023 IEEE. Reprinted, with permission, from Hüseyin Anil Gündüz, Sheetal Giri, Martin Binder, Bernd Bischl, and Mina Rezaei, Uncertainty Quantification for Deep Learning Models Predicting the Regulatory Activity of DNA Sequences, In 2023 International Conference on Machine Learning and Applications (ICMLA), December 2023.

In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of LMU Munich's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink. If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

Uncertainty Quantification for Deep Learning Models Predicting the Regulatory Activity of DNA Sequences

Hüseyin Anil Gündüz*
LMU Munich
Munich Center for Machine Learning
Munich, Germany
anil.guenduez@stat.uni-muenchen.de

Sheetal Giri*
TU Munich
Munich, Germany
sheetal.giri@tum.de

Martin Binder
LMU Munich
Munich Center for Machine Learning
Munich, Germany
martin.binder@stat.uni-muenchen.de

Bernd Bischl
LMU Munich
Munich Center for Machine Learning
Munich, Germany
bernd.bischl@stat.uni-muenchen.de

Mina Rezaei
LMU Munich
Munich Center for Machine Learning
Munich, Germany
mina.rezaei@stat.uni-muenchen.de

***Abstract*—The field of computational biology has been enhanced by deep learning models, which hold great promise for revolutionizing domains such as protein folding and drug discovery. Recent studies have underscored the tremendous potential of these models, particularly in the realm of gene regulation and the more profound understanding of the non-coding regions of the genome. On the other hand, this raises significant concerns about the reliability and efficacy of such models, which have their own biases by design, along with those learned from the data. Uncertainty quantification allows us to measure where the system is confident and know when it can be trusted. In this paper, we study several uncertainty quantification methods with respect to a multi-target regression task, specifically predicting regulatory activity profiles using DNA sequence data. Using the Basenji model, we investigate how such methods can improve in-domain generalization, out-of-distribution detection, and provide coverage guarantees on prediction intervals.**

***Index Terms*—Uncertainty Quantification, Regularity Activity of Genome Sequences, Multi-Target Regression**

I. Introduction

Deep learning (DL) models have the potential to process vast amounts of data and provide new insights, but their black-box nature can make it difficult to understand and trust their predictions. To address this issue, two approaches have emerged: explainable AI and uncertainty quantification (UQ). This study focuses on the latter.

In this paper, we study state-of-the-art methods to evaluate the effectiveness of UQ methods for sequence genomes in two different tasks. We use Basenji [1] as a DL model for genome prediction of gene regulation. The Basenji model is based on a convolutional neural network (CNN) architecture and is trained on large-scale genomic data to learn the patterns and relationships between genetic variations and gene regulation. We report our experiments based on Basenji due to its code and data availability, reproducibility, and popularity in the community. However, the techniques introduced in this paper can be applied to other multi-target regression tasks in genomics. Our main findings and contributions can be summarized as follows:

- We study a variety of approaches to estimate the uncertainty of a DL model for a multi-target regression problem in biology, specifically for predicting the regulatory activity profiles of genome sequences.
- We propose to use the negative binomial loss to allow the Basenji architecture to learn parameters to evaluate the variance of the predictions and thus quantify uncertainty.
- We study the effectiveness of the UQ methods for in-domain generalization and benchmark the performance improvement when using deep ensembles with our proposed loss.
- We study the effect of using deep ensembles for an out-of-distribution task and show that the deep ensemble not only outperforms a single model in terms of prediction performance but also is more uncertain in its predictions. This indicates a better ability to detect that the tested data differs from the trained data.
- We observe good coverage of the prediction intervals on the test set using conformalized quantile regression.
- We observe a moderate correlation between the absolute change in uncertainty and experimentally validated log fold changes in gene expression for a single nucleotide mutation in a subset of the data.

* Hüseyin Anil Gündüz and Sheetal Giri contributed equally to this work.

This work was funded in part by the German Federal Ministry of Education and Research (BMBF) under GenomeNet Grant No. 031L0199B.

II. METHODS

A. The Basenji Model And Datasets

1) Basenji: We use the Basenji model, a highly performant, standard CNN architecture for predicting regulatory activity profiles from DNA sequences [1]. In the Basenji model, a single DNA sequence of 131,072 base pairs is taken as input as one-hot encoded vectors representing the four nucleotides: A, C, G, and T. The model transforms this input to a representation of 896 $\times$ 1024 by multiple layers of convolution, pooling, and cropping. A final width-one convolution with 1643 filters for mouse and 5313 filters for human is applied, followed by a softplus ($f(x) = log(1 + e^x)$) activation function to ensure positive predictions. Thus, the average count/expression values across 128 base pairs for each genomic track and for each of the central 896 windows of the sequence is predicted for 1643 genomic tracks for mouse and 5313 genomic tracks for human. The Basenji model is trained to minimize a Poisson log-likelihood loss.

2) Datasets: The dataset was used as published by Kelley [2], with their pre-processing and train-validation-test split. The dataset consists of 38.2k human and 33.5k mouse genome sequences that are 131,072 base pairs long. The targets consist of expression values from 5,313 human and 1,643 mouse genomic tracks. Each track is an average across 896 windows, each spanning 128 base pairs. These genomic tracks are specific to different types of cells (for example, skin cells, stem cells, liver cells) that belong to different stages of development, from the embryo to adults at various ages. The tracks capture different types of biochemical activity [3], [4].

B. Evaluating Quality of Measured Uncertainty

In healthcare-related applications, it is essential to evaluate the reliability of the probabilistic forecasts for practical use. This requires a deeper understanding of three important concepts: calibration, sharpness, and OOD behavior.

1) Measuring Calibration and Sharpness: A forecaster is said to be well-calibrated if its predictions match the frequencies of the true observations. For example, if 100 different input samples are classified as "A" with a probability of 40% (each), then the classifier is well calibrated if about 40 of these samples actually belong to "A". Sharpness, on the other hand, is a property of probability distributions that describes the length of the prediction interval. A shorter prediction interval indicates that the model has a higher degree of certainty, which is a desirable property in practical applications. Calibration and sharpness can be quantified by using scoring rules [5], and the log-likelihood of a probability distribution, $\log p_\theta(y \mid \mathbf{x})$, is a proper scoring rule [6].

2) Out-of-Distribution Detection: In addition to generalizing well to new examples from the training data distribution, it is important for models to be able to identify data that are significantly different from what they have seen. A model's ability to detect OOD data can be measured by examining the confidence of its predictions. For instance, if a model trained for digit classification in images assigns high confidence to a prediction for an image of a cat, this indicates that the model is not able to detect OOD data.

C. Uncertainty Quantification methods with Neural Networks

1) Deep Ensemble: The Deep Ensemble method [6], is a method that quantifies uncertainty by combining the results from multiple models trained with random initialization. These models are neural networks trained to model the probabilistic predictive distribution by learning to predict the mean and variance of this distribution as outputs. The training objective is to minimize the Negative Log-Likelihood (NLL) of the distribution, as NLL is a proper scoring rule that can be considered a measure of predictive uncertainty.

The Basenji model is trained by minimizing a Poisson loss in the original setup [1], which implies that the data are modeled using a Poisson distribution. This distribution assumes that the mean and variance are equal. While the mean is used for the prediction and the model can be optimized with this purpose, the uncertainty in the prediction cannot be reliably quantified, as the variance is bound to the prior assumption of being equal to the mean. To address this problem, we adopted the Poisson loss to a Negative Binomial (NB) loss. We modify deep ensemble to predict the parameters of the NB loss, which are the mean μ and the overdispersion parameter α^{-1}. Here, μ is used for the prediction of the model, similarly to the Poisson loss. The other learned parameter α^{-1} is used for the uncertainty quantification, as the variance in NB loss is defined as $\sigma^2 = \mu + \alpha\mu^2$.

For our purposes, we adopt the loss following the NB2 formulation of the negative binomial distribution. We reformulate the log-likelihood of this distribution given by [7] in terms of μ and α^{-1}, to make it numerically feasible for training and the final NLL criterion is given by:

$$\begin{aligned} L =& \log\Gamma(\alpha^{-1}) + \log\Gamma(y+1) - \log\Gamma(\alpha^{-1}+y) \\ &+(y+\alpha^{-1})\log(\alpha^{-1}+\mu) - \alpha^{-1}\log(\alpha^{-1}) - y\log(\mu). \end{aligned} \tag{1}$$

The neural network is optimized to minimize the above NLL criterion the predicted mean μ and the predicted alpha inverse, α, for any given input.

During the test time, the predictions of the deep ensemble are computed by feeding the input x into N_d different models $\{\theta_1, \theta_2, ..., \theta_{N_d}\}$, which are trained independently with random initializations on the same data. The average mean $\mu_*(\mathbf{x})$ and average variance $\sigma_*^2(\mathbf{x})$ across all models are obtained by combining results from individual models as follows [6]:

$$\mu_*(\mathbf{x}) = N_d^{-1} \sum_n \mu_{\theta_n}(\mathbf{x}) \tag{2}$$

$$\sigma_*^2(\mathbf{x}) = N_d^{-1} \sum_n \left(\sigma_{\theta_n}^2(\mathbf{x}) + \mu_{\theta_n}^2(\mathbf{x})\right) - \mu_*^2(\mathbf{x}) \tag{3}$$

The average mean $\mu_*(\mathbf{x})$ is then the prediction of the deep ensemble and the average variance $\sigma_*^2(\mathbf{x})$ is the variance of the prediction as a measure of uncertainty. Note that the original Basenji model is not suitable for predicting the variance by using the deep ensemble since it is trained with Poisson loss, which assumes that mean and variance are equal.

2) MC Dropout: MC Dropout by Gal et al. [8] approximates uncertainty in the framework of Bayesian Neural Networks. Instead of treating weights as single values or point estimates, we learn a probability distribution over all possible weights that could explain our data. In the Bayesian framework, we start by assuming a prior distribution $p(\mathbf{W})$ over the weights that could have generated the data. The goal is to learn the posterior distribution over these weights based on the observed data. Given inputs $X = \{x_1, x_2..., x_n\}$ and observations $Y = \{y_1, y_2..., y_n\}$, such that $X \in \mathbb{R}^{NxD}$ and $Y \in \mathbb{R}^{NxQ}$, the posterior distribution is given by:

$$p(\mathbf{W} \mid \mathbf{X}, \mathbf{Y}) \propto p(\mathbf{Y} \mid \mathbf{X}, \mathbf{W}) p(\mathbf{W}) \tag{4}$$

Inferring the posterior using the Bayes rule is computationally intractable for neural network-based models and therefore, has to be approximated. MC dropout can be used to learn the approximate posterior. It is based on theoretical proof that a deep neural network trained with dropout before every weight layer and minimizing a Euclidean loss (for regression) or softmax loss (for classification) is equivalent to approximate inference in a deep Gaussian Process. Thus, even though Basenji originally minimizes a Poisson loss, MSE loss is used with this method to stay true to the theory accompanying it.

The mean and variance of the approximated posterior for any given input can be obtained by Monte Carlo Estimation. This is done by computing N_m samples from the posterior and averaging them. In practice, this computing of samples is done by having dropout enabled during the test time (that is, the dropout is still applied in the test time, similar to the training time) and doing N_m stochastic forward pass on the model for any given input. As mentioned, the predictions computed by N_m forward passes are averaged.

3) Conformalized Quantile Regression: In regression modeling, it is beneficial to determine the prediction interval, i.e., the upper and lower bounds within which the true prediction is likely to fall. Conformal prediction methods offer theoretically valid coverage guarantees for the prediction intervals generated by the algorithm [9]. The level of coverage controlled by the user-defined significance level α, where α = 0.1 implies that the model can have at most 10% miscoverage and consequently that ground truth value will be in the prediction interval 90% of the time. Therefore, the goal of this method is not to improve the uncertainty but to determine the interval in which the ground truth value will lie with respect to the user-defined level of confidence. Unlike traditional methods, conformal prediction does not make any strong assumptions about the data distribution, making it a more flexible approach.

This method trains two models, $\hat{t}_{0.05}$ and $\hat{t}_{0.95}$, using quantile loss instead of Poisson loss to regress to a quantile. The quantile loss is given by the following function:

$$L_\gamma\left(\hat{t}_\gamma, y\right) = \left(y - \hat{t}_\gamma\right)\gamma \mathbb{1}\left\{y > \hat{t}_\gamma\right\} + \left(\hat{t}_\gamma - y\right)(1-\gamma)\mathbb{1}\left\{y \leq \hat{t}_\gamma\right\} \tag{5}$$

This loss ensures that model $\hat{t}_{0.05}$ learns to make predictions that fall below the 5% quantile and $\hat{t}_{0.95}$ above the 95% quantile. Once fitted, the models can be used to generate the 90% prediction interval, as given by $[\hat{t}_{0.05}(x), \hat{t}_{0.95}(x)]$, for any given input x.

TABLE I
AVERAGE PEARSON CORRELATION (r) AND RMSE METRICS ON THE TEST SET ACROSS MODELS ON MOUSE AND HUMAN DATASETS. OUR MODIFIED DEEP ENSEMBLE NOT ONLY ALLOWS TO LEARN A VARIATION OF THE PREDICTION BUT ALSO IMPROVES PERFORMANCE METRICS

Dataset	**Mouse**		**Human**	
Method	***r***	**RMSE**	***r***	**RMSE**
Single Model - Poisson Loss	0.7036	1.8414	0.6198	1.8474
Single Model - NB Loss	0.6986	1.8760	0.6179	1.8606
Single Model - MSE Loss	0.6687	1.9375	0.6080	1.8656
Deep Ensemble (N_d = 5)	**0.7129**	**1.8395**	**0.6320**	**1.8164**
MC Dropout (N_m = 5)	0.6669	1.9216	0.5915	1.8922
MC Dropout (N_m = 50)	0.6699	1.9140	0.5945	1.8842

To have significant level of α= 0.1 for the conformal prediction, this prediction interval is further refined by calibrating it on the validation set as follows:

1) Calculate the set of scores $S = \{S_1, S_2, ..., S_n\}$ for all n input-output pairs (x_i, y_i) in the calibration set,

$$S_i = max(y_i - \hat{t}_{0.95}(x_i), \hat{t}_{0.05}(x_i) - y_i). \tag{6}$$

2) Calculate the $\left\lceil \frac{(n+1)(1-\alpha)}{n} \right\rceil^{th}$ quantile for the set of scores $S = \{S_1, S_2, ..., S_n\}$, which is denoted by $\hat{q}$.

During the test time, the valid prediction interval for any input x is given by

$$[\hat{t}_{0.05}(x) - \hat{q}, \hat{t}_{0.95}(x) + \hat{q}]. \tag{7}$$

III. RESULTS

A. Can UQ Methods Perform Better than Existing Methods for In-Distribution Tasks?

The prediction performance of the deterministic models and models with UQ methods was evaluated on both the mouse and human datasets. The results are shown in Table I. The deterministic model was trained with a Poisson loss, and the Basenji model has trained with a Negative Binomial (NB) loss. The Basenji model with the NB loss was also evaluated for the deep ensemble method, while the Mean Squared Error (MSE) loss was used for MC Dropout. Both the Poisson loss and the NB loss were comparable in performance for the deterministic model, with the NB loss being an extension of the Poisson loss that accounts for overdispersion. All models trained with MSE loss had relatively high root mean squared error (RMSE) values and lower Pearson correlation (r) values. This could be because the MSE loss assumes that the data follow a Gaussian distribution, which is not appropriate for this type of data.

Deep ensemble predictions were obtained by averaging predictions across five randomly initialized models trained with NB loss. For both mouse and human datasets, it was observed that deep ensemble with our proposed loss improves the performance compared to all the other models in Pearson correlation (r) and RMSE metrics, including the original Basenji architecture. This could be because ensembles are known to perform better in general in ML [10]. In addition, deep ensemble with our proposed loss can learn the standard

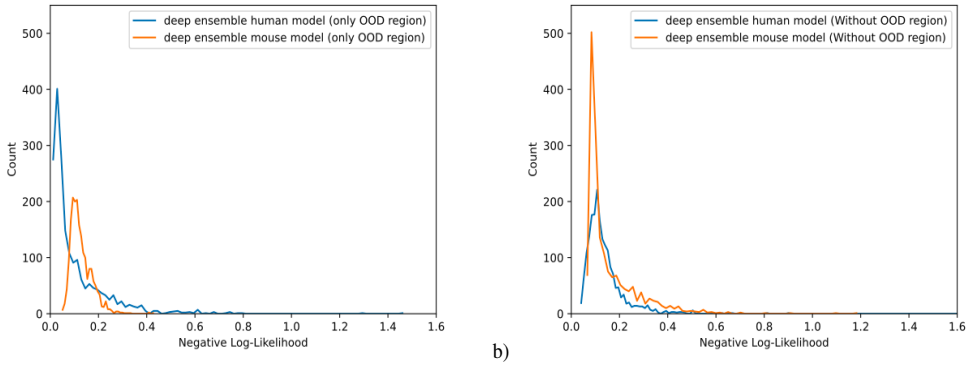


Fig. 1. Histogram of predictive uncertainty for mouse and human model on human test sequences. (a) The mouse model has higher predictive uncertainty compared to the human model because of the domain shift in the input distribution due to the Alu elements in the human sequence (OOD regions). (b) When the Alu elements are removed, the uncertainty distributions are more similar.

TABLE II
HUMAN TEST SET PERFORMANCE - AVERAGE PEARSON CORRELATION (r) AND PREDICTIVE UNCERTAINTY (NLL) FOR HUMAN SEQUENCES WHEN MODELS ARE TRAINED ON THE MOUSE VS HUMAN DATA. MODELS ARE EVALUATED FOR REGIONS CONTAINING ONLY ALU ELEMENTS (OOD REGION) AND REGIONS WITHOUT ALU ELEMENTS (EXC. OOD REGION)

Test Set Samples		**OOD region**		**Exc. OOD region**	
Method	**Train Data**	***r***	**NLL**	***r***	**NLL**
Deep Ensemble	Human	0.7628	0.1035	0.5567	0.1379
Deep Ensemble	Mouse	**0.4146**	0.1275	**0.5176**	0.1596
Single Model	Mouse	0.2979	0.1258	0.4837	0.1772

deviation of the prediction for each data sample, enabling quantification of uncertainty.

B. Deep Ensemble with our proposed loss both outperforms a single model and better recognizes OOD data

Due to the largely conserved Transcription Factor binding preferences between mice and humans for the same cell types, a model trained for predicting regulatory activity on mouse DNA should be able to generalize well for human DNA sequences [2]. Cochran et al. [11] evaluated this by evaluating how a model trained to predict binding behavior on mouse DNA performs on human DNA input. They observed degraded prediction performance for regions in the sequence containing repeated elements called Alus which are exclusive to primates and cover 10% of the human genome [12]. We refer to this region as out-of-distribution(OOD) region for the mouse model, since the model has not seen the Alu elements during its training.

We tested the performance of the model trained on mouse data to predict shared genomic tracks between mouse and human. Specifically, the CAGE expression values for common Myeloid Progenitor cells and Mesenchymal stem cells are estimated and the prediction performance and uncertainty for human sequence input are evaluated.

The NLL measure is used to assess the model's uncertainty, with higher NLL values implying higher uncertainty. Comparing the deep ensemble trained on the mouse data with the single model trained on the same data, Table II shows that the deep ensemble has slightly higher uncertainty in the OOD region, although it has considerably better performance in terms of the Pearson correlation coefficient (r). Thus, deep ensemble does not only improve the performance in both OOD and non-OOD region but is also more uncertain in its prediction, when tested on the OOD region. This indicates a better ability to detect that the tested data is different from the trained data. For the non-OOD region, the deep ensemble not only outperforms a single model in terms of r, but also has considerably smaller uncertainty. This indicates that the deep ensemble model both outperforms the single model and is better at distinguishing that this region is not OOD.

Table II also shows that for the OOD regions containing only Alu elements, a model trained on mouse data has a notably lower Pearson correlation between the true observation and prediction compared to a model trained on human data. The differences between Pearson correlation values are much smaller for the sequence without Alu elements. Thus, our selection of OOD region is justified.

Fig. 1 shows NLL values for 2017 sequences in the human test set. The model trained on mouse data is more uncertain about its predictions for OOD regions containing Alu elements in the human test set. However, when Alu elements were removed, the count distributions of NLL values are closer, suggesting that the mouse model can perform well for human sequences if OOD regions are removed.

C. Prediction intervals obtained using Conformal Quantile Regression provide good coverage on test set

The Basenji model for mouse DNA was evaluated using Conformal Quantile Regression to determine 90% confidence intervals for predictions. The results showed that the intervals had an average coverage rate of 89.63% on the test set, meaning the predicted interval accurately captured the true

expression value for that percentage of the test set. The coverage rates for each sequence in the test set are displayed in a histogram (Fig. 2), calculated as the proportion of values out of a total of 896 bins and 1643 genomics tracks for which the predicted interval contained the true value.

Fig. 3 illustrates the usefulness of prediction intervals to practitioners, as they can determine the level of confidence in the model. The length of the interval (the distance between the upper and lower bounds) provides further insight into uncertainty of the model, with shorter intervals indicating greater confidence.

D. Experimentally observed variant effect correlates with the change in model uncertainty

In-silico mutation is an important application of sequence-to-profile models, providing an inexpensive way to study the effect of mutations in one or more nucleotides on regulatory activity [13]. If the model is of high quality, the difference in uncertainty between its predictions on the original sequence and the mutated sequence can reflect the degree of change in gene expression. A model that is confident in its predictions should not show a significant change in uncertainty values when presented with two variants of a sequence with known similar expression values. However, the model's predictions should be less certain when expression levels change drastically.

This concept was tested using the saturation mutagenesis data [14], which comprises experiments that record gene expression changes for all possible variants of a single nucleotide in 15 locus regions within the human genome. The data includes reference nucleotides (REF) and three possible alterations (ALT) for each variant, with expression changes measured as log fold change (log FC) calculated as

$$\log \text{FC} = \log_2 \text{Expression}_{ALT} - \log_2 \text{Expression}_{REF} . \quad (8)$$

To perform in-silico mutation using Basenji, cell lines from the experiments had to be mapped to corresponding genomic tracks, which the human model is trained to predict. This was done by matching cell lines with CAGE/DNase genomic tracks based on their ENCODE description, following a similar experiment in Karollus et al. [15].

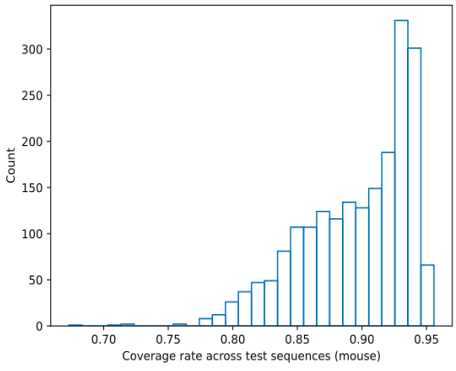


Fig. 2. Histogram for coverage on test set sequences of mouse dataset.

The deep ensemble model is used to predict expression values for all mutations, assuming a NB distribution and predicting the mean and variance of genomic track values for a given sequence. The mean is the expression, while the variance is used as a measure of uncertainty, with the change in uncertainty measured as the absolute difference in the predicted variance between the sequence containing the altered mutation and the sequence with the reference nucleotide, i.e.,

$$\|VAR(ALT) - VAR(REF)\|. \quad (9)$$

The best prediction for Basenji is observed for variants in the LDLR locus measuring CAGE expression for the HepG2 cell line, with a moderate correlation of 0.65 (p-value = 1.19e-89) between the measured variant effect (log fold change) and the predicted variant effect (change in predicted expression). We focus on this locus to assess changes in uncertainty and evaluate the correlation between the change in uncertainty and the experimentally observed log fold change at two levels of this locus. First, changes are measured only for the 128 bp window/bin containing the mutated variant, resulting in a lower correlation of 0.34 (p-value = 3.02e-21). However, the correlation is much better when the prediction uncertainty is averaged over the whole sequence, as shown in Fig. 4, with a Pearson correlation of 0.67 (p-value = 7.04e-95).

IV. Discussion

This study explored UQ techniques for DL models that perform sequence-to-regulatory activity profile prediction. This work aims to provide insights for improving the performance of such models and enable practitioners to make informed decisions based on the reliability of the model's results.

To this end, we evaluated three UQ methods (MC Dropout, Deep Ensemble, and Conformalized Quantile Regression) using the Basenji model. Our results indicate that Deep Ensemble performed best in model performance on the test data, while Conformalized Quantile Regression generated prediction intervals with 90% coverage. In addition, we observed an increase in uncertainty for OOD inputs when evaluating Alu elements of the human sequence on the mouse model.

The current literature on gene expression prediction models incorporating UQ techniques is limited. While there have been efforts to estimate uncertainty using Bayesian methods for Gaussian Mixture Models, these techniques might not be directly transferable to DL models [16]. On the other hand, more research has been conducted on quantifying uncertainty in medical imaging models, but these studies primarily focus on classification tasks. The work by Laves et al. [17] provides the closest relevance to the profile prediction task, demonstrating the feasibility of attaining well-calibrated uncertainty values using MC Dropout for regression tasks using medical images.

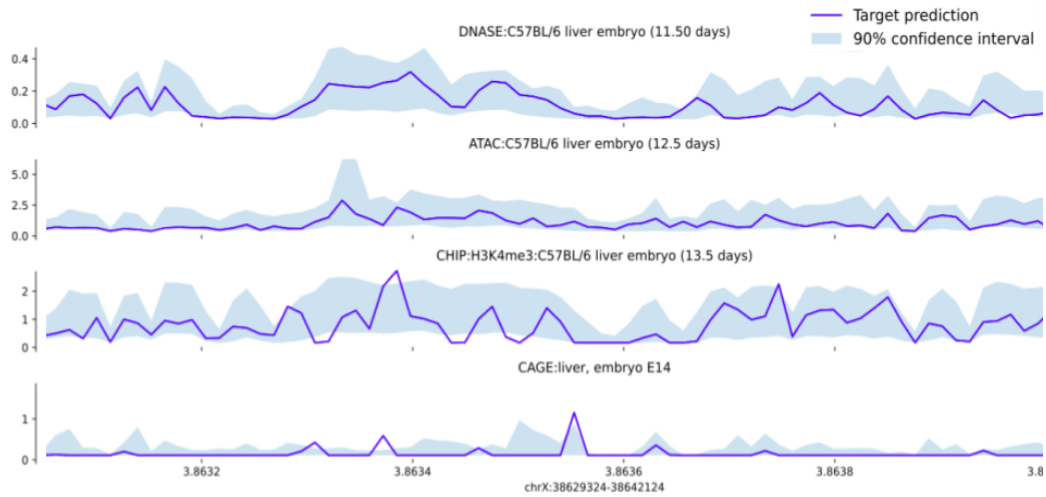


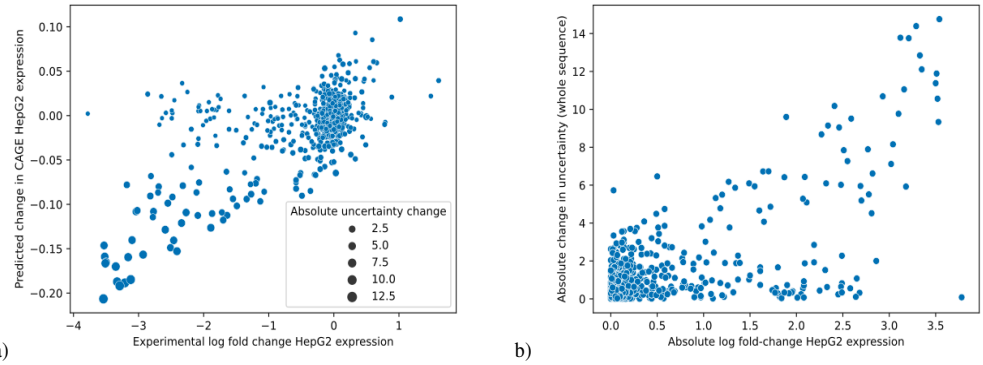
Fig. 3. Shown for a model trained and tested on mouse data, Conformal Quantile Regression generates 90% confidence intervals for genomic track predictions.

A. Limitations

The field of UQ in DL is largely focused on classification problems and single-output regression, leaving a gap in UQ for multi-target regression problems. The task of multi-target regression in genomic sequences involves predicting multiple expression/count values across nucleotides for multiple genomic tracks that capture different biochemical behavior. Treating this problem as multiple single-output regression problems ignores the relationship between tasks and their correlations. Further research should examine methods for this using the Conformal Prediction framework for multitarget regression [18] and using the Generalized Fiducial Inference framework for high dimensional multitask learning [19].

When using MC Dropout for multi-target regression in genomic sequences, MSE Loss is used to approximating a Gaussian process [8]. The Basenji model enforces positive target signals by preprocessing the data to clip negative values to zero [2]. This allows for modelling the data using a Poisson distribution and has shown to have the best performance as seen in Table I, for the human and mouse data. However, using the MSE loss for deterministic models and MC Dropout has relatively poor performance, highlighting the need for distribution-free methods such as conformal prediction [20].

Deep Ensemble is computationally expensive and infeasible for larger models like Enformer [21], the current state-of-the-art for the sequence to regulatory activity prediction. There is a need for UQ methods that can be applied without retraining larger models. Finally, uncertainty values can be useful for in-silico mutation analysis. The relationship between uncertainty



a) b)

Fig. 4. (a) A moderate correlation between the absolute change in uncertainty averaged across predictions and the absolute experimental log fold change, for variants in the LDLR locus measuring expression for the HepG2 cell line. (b) Predicted change in CAGE expression for HepG2 plotted against log fold change, the bigger dots indicate predictions for which the model is more uncertain.

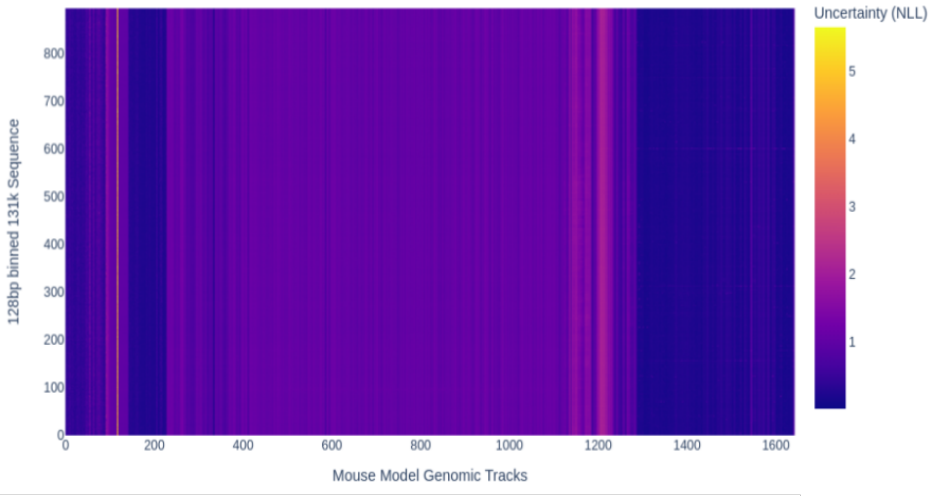


Fig. 5. Heatmap of Averaged Predictive Uncertainty (NLL) across all predictions made by Deep Ensemble on 1937 Mouse test set sequences.

values and log fold change in gene expression for single nucleotide mutations was studied using the dataset by Kircher et al. [14]. Although some correlation was observed in Fig. 4, one cannot be sure about the quality of the predictions because the cell lines used in the experiment do not exactly match the genomic tracks that Basenji is predicting for and the correlation alone cannot be trusted as log fold change is an average value that may be biased and noisy. Having base pair resolution expression values per mutation would have been more helpful in this case, as they would serve as ground truth predictions that could be used to evaluate uncertainty quality and improve the model.

B. Future Work

The development of accurate and interpretable models for genomic sequence analysis is an important area of research. Incorporating uncertainty into sequence-to-profile models offers a promising direction for improving their performance. Here, we propose two avenues for improvement.

First, the current multitask loss formulation for predicting individual target values can be enhanced by considering task correlations. The current design computes the loss independently for each task and sums them, treating all tasks equally. An alternative approach is the weighting of losses using task-specific model uncertainty [22].

Second, the uncertainty can be used to identify the regions with data problems. Fig. 5 shows an example of the predictive uncertainties of a deep ensemble model averaged over the test set. The genomic tracks with high uncertainty can be further investigated to verify data quality. To effectively analyze high-resolution predictions such as those made by models like Basenji, specialized visualization tools and metrics must be developed. In addition, interpretability tools such as importance scores [23] should be integrated to understand which input features contribute most to the uncertainty in the prediction.

In conclusion, incorporating uncertainty into sequence-to-profile models has immense potential for improving their accuracy and interpretability. Further work in this direction is important to advance the field of genomic sequence analysis.

ACKNOWLEDGMENTS

We thank Alexander Karollus and Laura Martens for helpful discussions.

REFERENCES

[1] D. R. Kelley, Y. A. Reshef, M. Bileschi, D. Belanger, C. Y. McLean, and J. Snoek, "Sequential regulatory activity prediction across chromosomes with convolutional neural networks," *Genome research*, vol. 28, no. 5, pp. 739–750, 2018.

[2] D. R. Kelley, "Cross-species regulatory sequence activity prediction," *PLoS computational biology*, vol. 16, no. 7, p. e1008050, 2020.

[3] A. P. Boyle, S. Davis, H. P. Shulha, P. Meltzer, E. H. Margulies, Z. Weng, T. S. Furey, and G. E. Crawford, "High-resolution mapping and characterization of open chromatin across the genome," *Cell*, vol. 132, no. 2, pp. 311–322, 2008.

[4] H. Takahashi, T. Lassmann, M. Murata, and P. Carninci, "5 end–centered expression profiling using cap-analysis gene expression and next-generation sequencing," *Nature protocols*, vol. 7, no. 3, pp. 542–561, 2012.

[5] T. Gneiting and A. E. Raftery, "Strictly proper scoring rules, prediction, and estimation," *Journal of the American statistical Association*, vol. 102, no. 477, pp. 359–378, 2007.

[6] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," *Advances in neural information processing systems*, vol. 30, 2017.

[7] A. C. Cameron and P. K. Trivedi, *Regression analysis of count data*. Cambridge university press, 2013, vol. 53.

[8] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*. PMLR, 2016, pp. 1050–1059.

[9] V. Vovk, A. Gammerman, and G. Shafer, "Conformal prediction," *Algorithmic learning in a random world*, pp. 17–51, 2005.

[10] T. G. Dietterich, "Ensemble methods in machine learning," in *International workshop on multiple classifier systems*. Springer, 2000, pp. 1–15.
[11] K. Cochran, D. Srivastava, A. Shrikumar, A. Balsubramani, R. C. Hardison, A. Kundaje, and S. Mahony, "Domain-adaptive neural networks improve cross-species prediction of transcription factor binding," *Genome research*, vol. 32, no. 3, pp. 512–523, 2022.
[12] M. A. Batzer and P. L. Deininger, "Alu repeats and human genomic diversity," *Nature reviews genetics*, vol. 3, no. 5, pp. 370–379, 2002.
[13] J. Schreiber and R. Singh, "Machine learning for profile prediction in genomics," *Current Opinion in Chemical Biology*, vol. 65, pp. 35–41, Dec. 2021.
[14] M. Kircher, C. Xiong, B. Martin, M. Schubach, F. Inoue, R. J. Bell, J. F. Costello, J. Shendure, and N. Ahituv, "Saturation mutagenesis of twenty disease-associated regulatory elements at single base-pair resolution," *Nature communications*, vol. 10, no. 1, pp. 1–15, 2019.
[15] A. Karollus, T. Mauermeier, and J. Gagneur, "Current sequence-based models capture gene expression determinants in promoters but mostly ignore distal enhancers," *bioRxiv*, 2022.
[16] B. Rajaratnam, D. Sparks, K. Khare, and L. Zhang, "Scalable bayesian shrinkage and uncertainty quantification for high-dimensional regression," *arXiv preprint arXiv:1509.03697*, 2015.
[17] M.-H. Laves, S. Ihler, J. F. Fast, L. A. Kahrs, and T. Ortmaier, "Well-calibrated regression uncertainty in medical imaging with deep learning," in *Medical Imaging with Deep Learning*. PMLR, 2020, pp. 393–412.
[18] S. Messoudi, S. Destercke, and S. Rousseau, "Conformal multi-target regression using neural networks," in *Conformal and Probabilistic Prediction and Applications*. PMLR, 2020, pp. 65–83.
[19] Z. Wei, "Some contributions to high-dimensional statistical machine learning," Ph.D. dissertation, UC Davis, 2016.
[20] A. N. Angelopoulos and S. Bates, "A gentle introduction to conformal prediction and distribution-free uncertainty quantification," *arXiv preprint arXiv:2107.07511*, 2021.
[21] Ž. Avsec, V. Agarwal, D. Visentin, J. R. Ledsam, A. Grabska-Barwinska, K. R. Taylor, Y. Assael, J. Jumper, P. Kohli, and D. R. Kelley, "Effective gene expression prediction from sequence by integrating long-range interactions," *Nature methods*, vol. 18, no. 10, pp. 1196–1203, 2021.
[22] A. Kendall, Y. Gal, and R. Cipolla, "Multi-task learning using uncertainty to weigh losses for scene geometry and semantics," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7482–7491.
[23] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," in *International conference on machine learning*. PMLR, 2017, pp. 3145–3153.

6.2 FiLM-Ensemble: Probabilistic Deep Learning via Feature-wise Linear Modulation

Contributing Article

Turkoglu MO, Becker A, Gündüz HA, Rezaei M, Bischl B, Daudt RC, D' Aronco S, Wegner J, Schindler K (2022). "FiLM-Ensemble: Probabilistic Deep Learning via Feature-wise Linear Modulation." In S Koyejo, S Mohamed, A Agarwal, D Belgrave, K Cho, A Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 22229–22242. Curran Associates, Inc. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/8bd31288ad8e9a31d519fdeede7ee47d-Paper-Conference.pdf

Declaration of Contributions

Hüseyin Anil Gündüz contributed to this paper as a co-author with the following significant contributions:
Hüseyin Anil Gündüz designed, developed the code, and ran all the experiments of the 6mA identification task, which is presented in Chapter 3.5 and Figure 3 of the main paper, and Subchapter A3 and Figure 2 in the Appendix. These experiments show that the proposed method can also be applied successfully on 1-dimensional sequential genomics data and therefore prove the effectiveness of the proposed method on 1-dimensional data. Additionally, the performed experiments by Hüseyin Anil Gündüz show that there is a tradeoff between improving the calibration of the model predictions and improving the model's performance, and this tradeoff can be managed by adjusting the initialization gain parameter of the batch normalization layers, which is a hyperparameter induced by the proposed method. Hüseyin Anil Gündüz also presented the paper at the NeurIPS 2023 main conference together with Mehmet Ozgur Turkoglu.

Contribution of the coauthors:
Mehmet Ozgur Turkoglu is the main contributor and author of the paper. Mehmet Ozgur Turkoglu proposed the method, developed the main parts of the code, and ran most of the experiments. Alexander Becker worked on parallelization of the algorithm and performed experiments for benchmarking memory and inference complexities. He also helped with the code implementation of the baselines. Mina Rezaei designed, ran, and analyzed the experiments on the retinal glaucoma detection task, presented in Chapters 3.4 and 3.6. Konrad Schindler was the main supervisor of the project. Konrad Schindler provided supervision and assistance throughout the entire process and helped with the writing of the paper significantly. All authors contributed to the writing and editing of the paper.

FiLM-Ensemble: Probabilistic Deep Learning via Feature-wise Linear Modulation

Mehmet Ozgur Turkoglu ETH Zurich
Alexander Becker ETH Zurich
Hüseyin Anil Gündüz LMU Munich
Mina Rezaei LMU Munich
Bernd Bischl LMU Munich
Rodrigo Caye Daudt ETH Zurich
Stefano D'Aronco ETH Zurich
Jan Dirk Wegner ETH Zurich & University of Zurich
Konrad Schindler ETH Zurich

Abstract

The ability to estimate epistemic uncertainty is often crucial when deploying machine learning in the real world, but modern methods often produce overconfident, uncalibrated uncertainty predictions. A common approach to quantify epistemic uncertainty, usable across a wide class of prediction models, is to train a *model ensemble*. In a naïve implementation, the ensemble approach has high computational cost and high memory demand. This challenges in particular modern deep learning, where even a single deep network is already demanding in terms of compute and memory, and has given rise to a number of attempts to emulate the model ensemble without actually instantiating separate ensemble members. We introduce FiLM-Ensemble, a deep, implicit ensemble method based on the concept of Feature-wise Linear Modulation (FiLM). That technique was originally developed for multi-task learning, with the aim of decoupling different tasks. We show that the idea can be extended to uncertainty quantification: by modulating the network activations of a single deep network with FiLM, one obtains a model ensemble with high diversity, and consequently well-calibrated estimates of epistemic uncertainty, with low computational overhead in comparison. Empirically, FiLM-Ensemble outperforms other implicit ensemble methods, and it comes very close to the upper bound of an explicit ensemble of networks (sometimes even beating it), at a fraction of the memory cost.

1 Introduction

A key component for reliable and trustworthy machine learning are algorithms that output not only accurate predictions of the target variables, but also well-calibrated estimates of their uncertainty [Gal and Ghahramani, 2016]. The overall uncertainty of a predictor is usually decomposed into two parts [Der Kiureghian and Ditlevsen, 2009]. *Aleatoric uncertainty* is inherent in the data, for instance due to class overlap or sensor noise. On the contrary, *epistemic uncertainty* characterises the uncertainty of the model weights, due to a lack of knowledge about parts of the input space that are insufficiently represented in the training set. Uncertainty caused by distributional shifts between the training and test data is sometimes conceived as a third source of uncertainty [Malinin and Gales, 2018], but in practice often modelled as part of epistemic uncertainty.

Measuring epistemic uncertainty for complex models such as deep neural networks is not trivial: by definition one cannot derive it from the training data, since it concerns the behavior of the model

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

in regions of the input space that are not represented in the training set. Different methods have been explored [Welling and Teh, 2011, Graves, 2011, Blundell et al., 2015, Lakshminarayanan et al., 2017a, Huang and Belongie, 2017, Gal and Ghahramani, 2016], however the de facto standard remain deep ensemble models [Lakshminarayanan et al., 2017a]. In its basic form, such a *deep ensemble* is simply a collection of independently trained networks that can be regarded as Monte Carlo samples from the model space. To obtain ensemble members with reasonably low correlation, one can exploit the stochastic nature of the optimisation procedure, with different (random) weight initialisation and different (random) batches during stochastic gradient descent. The expectation is that, once trained, the ensemble members will agree for inputs near the training samples, since the loss function favours similar outputs at those locations. Whereas they may disagree in unseen regions of the data space. Thus, the spread of their predictions is a measure of epistemic uncertainty. The ensemble idea is conceptually very simple, but nevertheless yields uncertainties that are well calibrated, i.e., they are in line with the actual prediction errors (in expectation).

The drawback of deep ensembles is their large computational cost. Both computation and memory consumption grow directly proportionally with the number of ensemble members, during training as well as during inference. This makes them impractical in hardware-constrained settings, and leads to a widening gap as models continue to grow in size, while at the same time applications like mobile robotics, augmented reality and smart sensor networks increase the need for mobile and embedded computing.

To improve efficiency, several researchers have explored ways to mimic deep ensembles without explicitly duplicating the underlying network. Possible strategies include the reuse and recombination of network modules [e.g., Wen et al., 2020, Havasi et al., 2021], injection of noise at inference time [e.g., Gal and Ghahramani, 2016], as well as hybrid variants [e.g., Durasov et al., 2021]. Empirically, these models do speed up training and/or inference, but still exhibit a significant performance gap compared to the naïve, explicit ensemble, both w.r.t. prediction quality and w.r.t. the calibration of the predicted (epistemic) uncertainties.

The lottery ticket hypothesis [Frankle and Carbin, 2018] and other network pruning studies, e.g., by Han et al. [2015], Lee et al. [2019], Mallya and Lazebnik [2018], underline that neural networks are heavily over-parameterized. Their parameters are used inefficiently, and they can be pruned significantly without large performance drops. In lifelong learning and multi-task learning it is essential to use the network efficiently, in order to limit computational overhead when introducing new tasks. There are recent works that achieve good performance in these tasks by introducing modulation (respectively, adaptation) strategies. In particular, Li et al. [2018], Takeda et al. [2021] propose an efficient lifelong learning / domain adaptation method for multi-task learning, utilizing single feature-wise linear modulation (FiLM).

Inspired by that line of work, we propose a new, efficient ensemble method, FiLM-Ensemble. Our method adapts feature-wise linear modulation as an alternative way to construct an ensemble for (epistemic) uncertainty estimation. FiLM-Ensemble greatly reduces the computational overhead compared to the naïve ensemble approach, while performing almost on par with it, sometimes even better. In a nutshell, FiLM-Ensemble can be described as an implicit model ensemble in which each individual member is defined via its own set of linear modulation parameters for the activations, whereas all other network parameters are shared among ensemble members – and therefore only need to be stored and trained once. Thanks to this design, our method requires only a very small number of additional parameters on top of the base network, e.g., converting a single ResNet-18 model to an ensemble of 16 models increases the parameter count by 1.3%, compared to an increase by 1500% when setting up a naïve ensemble, see Table 1. We further show that FiLM-Ensemble results in more diverse ensemble members compared to other efficient ensemble methods. For instance, on the Cifar-10 benchmark it achieves diversity scores $> 6.9\%$ and up to 9.2% (depending on ensemble size), against 6.8% for a naive ensemble, see Fig. 1. Our contributions can be summarized as follows.

- We propose FiLM-Ensemble, a novel parameter- and time-efficient deep ensemble method.
- FiLM-Ensemble is designed in such a way that it can be readily combined with many popular deep learning models, by simply replacing batch normalisation (batchnorm) layers with conditional batchnorm layers.
- We show that FiLM-Ensemble provides an excellent trade-off between accuracy and calibration performance, improving over existing ensemble methods.

2 FiLM-Ensemble

Perez et al. [2018] achieved the linear feature modulation by varying the affine parameters of the batch normalization layers $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$. For each batchnorm layer n in the network, the parameters are predicted according to some conditioning input z (for instance, different prediction tasks):

$$\boldsymbol{\gamma}_n = g_n(z) \qquad \boldsymbol{\beta}_n = h_n(z) \tag{1}$$

The size of $\boldsymbol{\gamma}_n$ and $\boldsymbol{\beta}_n$ is $\mathbb{R}^{D_n}$ where D_n is the feature dimension at layer n. These parameters are then used to linearly modulate the activations at the n-th layer $\mathbf{F}_n$:

$$\text{FiLM}(\mathbf{F}_n|\boldsymbol{\gamma}_n, \boldsymbol{\beta}_n) = \boldsymbol{\gamma}_n(z) \circ \mathbf{F}_n + \boldsymbol{\beta}_n(z) \;, \tag{2}$$

with $\circ$ being the Hadamard (element-wise) product taken w.r.t. the feature dimension.

In our scenario we aim to use the affine parameters to instantiate different ensemble members, i.e., the conditioning variable z is simply an index $m \in \{1, ..., M\}$ that identifies the ensemble member. The functions $g_n(z)$ and $h_n(z)$ degenerate to look-up tables, so we can dispose of them and simply learn M different sets of affine parameters for every batchnorm layer $\mathbf{F}_n$:

$$\text{FiLM}(\mathbf{F}_n|\boldsymbol{\gamma}_n^m, \boldsymbol{\beta}_n^m) = \boldsymbol{\gamma}_n^m \circ \mathbf{F}_n + \boldsymbol{\beta}_n^m \;. \tag{3}$$

To achieve ensemble members with diverse parameters, we resort to Xavier initialization, i.e., all $\boldsymbol{\gamma}_n^m$ and $\boldsymbol{\beta}_n^m$ are sampled from a uniform distribution bounded between:

$$\pm \frac{\sqrt{3}}{\sqrt{D_\text{n}}} \rho \;, \tag{4}$$

where D_n is the number of features (channels) in the n-th layer, and ρ is an initialization gain factor. In our setting ρ is a tunable hyperparameter that allows one to control the trade-off between predictive accuracy and calibration of the model, see Section 3.5. In general, increasing ρ leads to more diverse ensemble members and thus favours calibration. Note that as $\rho \to 0$, all ensemble members start from similar initial values for $\boldsymbol{\gamma}_n^m$ and $\boldsymbol{\beta}_n^m$ and the FiLM-Ensemble gradually collapses to a single model.

During training, we feed each input sample $\mathbf{x}$ to every ensemble member and obtain predictions $y_m = f_{\boldsymbol{\theta}, \boldsymbol{\gamma}^m, \boldsymbol{\beta}^m}(\mathbf{x})$, which depend both on the member-specific parameters $(\boldsymbol{\beta}^m, \boldsymbol{\gamma}^m)$ and on the shared parameters θ. All those parameters are optimized together to minimise the chosen loss function. Here we focus on classification and use a standard cross-entropy (CE) loss.

At inference time the final prediction $\hat{y}$ is obtained by averaging the M predictions of the ensemble:

$$\hat{y} = \frac{1}{M} \sum_{m=1}^{M} y_m \;. \tag{5}$$

2.1 Implementation Details

We implemented 1-dimensional and 2-dimensional FiLM-Ensemble layers, to be used in combination with 1-dimensional and 2-dimensional convolution operations, respectively. The forward passes through different ensemble members can easily be parallelized by replicating both the input tensor and the FiLM parameters $\boldsymbol{\gamma}, \boldsymbol{\beta}$ along the batch dimension and applying the affine transformation Eq. 3 (the same holds for the BatchEnsemble method). In this way all ensemble members run simultaneously on a single device, thus optimally utilizing modern tensor computing hardware without having to load several instances of the classification network into memory. We have implemented FiLM-Ensemble in PyTorch and release the source code.[1]

For all experiments, we optimize the model parameters with standard stochastic gradient descent, with momentum $\mu = 0.9$ and weight decay $\lambda = 0.0005$ for regularization. We train for 200 epochs with batch size 128. The learning rate is initially set to 0.1 and decays according to a cosine annealing schedule [Loshchilov and Hutter, 2017]. The initialisation gain is set to $\rho = 2$ for all experiments, except for genome sequences (see Table 3), where $\rho \in \{4, 8, 16, 32\}$. Please refer to the supplementary material for additional, dataset-specific details.

[1] `https://github.com/prs-eth/FILM-Ensemble`

3 Experiments

In the following, we first empirically examine the ability of FiLM-Ensemble to learn independent ensemble members, and compare it to deep ensembles. Then we analyze the predictive accuracy and uncertainty calibration of our method, as well as its computational cost and memory consumption. We compare its performance against several baseline methods and state-of-the-art alternatives on a variety of different datasets, and using different backbone architectures. All experimental results are averaged over three runs with different random seeds.

We evaluate our proposed method on a diverse set of classification tasks, including popular image classification benchmarks, image-based medical diagnosis, and genome sequence analysis. **CIFAR-10** and **CIFAR-100** [Krizhevsky, 2009] are widely used testbeds for image classification, and deep learning in general. They consist of clean images of objects from 10, respectively 100, different semantic classes. Each of the two datasets contains 60,000 images, of which 10,000 are reserved for testing. The semantic classes are uniformly distributed in the datasets and stratified across the train/test spit. **Retina Glaucoma Detection** [Diaz-Pinto et al., 2019] is a real-world clinical dataset that includes microscopic retina images from 956 patients with the neuropathic disease Glaucoma, and from 1401 subjects with normal (healthy) retinas. Each input sample is a single RGB image; we resize all images to 128×128. The image augmentation applies by combination of crop, horizontal flip, and color jitter. **REFUGE 2020** [Orlando et al., 2020] was a challenge at the MICCAI-2020 conference, aimed at retinal Glaucoma diagnosis. The dataset consists of 800 microscopic retina images with size 1411×1411 pixels, collected from different clinics. We use this dataset in conjunction with the models trained on the previous dataset [Diaz-Pinto et al., 2019], to evaluate the ability to detect out-of-distribution samples with the help of the predicted uncertainties (Table 4). **6mA Identification** [Li et al., 2021], is a 1-dimensional sequential genome dataset. It consists of DNA sequences of rice plants, along with binary labels that indicate whether the sequence is a N6-methyladenine (6mA) site. 6mA is an important DNA modification associated with several biological processes, such as regulating gene transcription, DNA replication and DNA repair [Campbell and Kleckner, 1990, Cheng et al., 2016, Pukkila et al., 1983]. Each sequence consists of 41 nucleotides. As there are 4 different types of nucleotides, each one-hot encoded sequence is represented by a 41×4 matrix. In total there are 269,500 training samples and 38,500 test samples.

We compare FiLM-Ensemble against *(i)* a single model without any ensembling, as the elementary baseline, *(ii)* a naïve deep ensemble [Lakshminarayanan et al., 2017b], and *(iii)* MC-Dropout [Gal and Ghahramani, 2016]. Furthermore, we compare with other state-of-the-art methods, including: *(iv)* Masksemble [Durasov et al., 2021] which can be seen as an extension of MC-Dropout, *(v)* MIMO ("multi-input multi-output") [Havasi et al., 2021] which defines a multi-head architecture where each head acts as one ensemble member, and *(vi)* BatchEnsemble [Wen et al., 2020] which creates an efficient ensemble by expanding the layer weights using low rank matrices. Arguably, this layer-wise modification of an underlying, common representation is the approach that comes closest to our work.

3.1 Diversity Analysis

Diversity is the key to constructing powerful ensembles: nothing is to be gained from highly correlated ensemble members that return similar outputs for (almost) any input [Zhang and Ma, 2012]. In order to analyze the diversity (respectively, the degree of independence) among members, we compute two distance metrics between the members' predictive distributions. Let f_i and f_j denote two different ensemble members for a classification task. We first measure the *disagreement score* $\mathcal{D}$, defined as the fraction of all s test samples for which the two members return different answers, averaged over all possible pairs of members:

$$\mathcal{D} = \frac{2}{M(M-1)} \sum_{i=1}^{M} \sum_{j=i+1}^{M} \sum_{k=1}^{s} \frac{1}{s} \big[f_i(\mathbf{x}_k) \neq f_j(\mathbf{x}_k) \big] \,, \tag{6}$$

with $[\cdot]$ being the Iverson bracket. Second, we also measure the Kullback–Leibler (KL) divergence between the two predictive distributions $p(f_i)$ and $p(f_j)$, again averaged over all pairs:

$$KL = \frac{2}{M(M-1)} \sum_{i=1}^{M} \sum_{j=i+1}^{M} \sum_{k=1}^{s} p(f_i(\mathbf{x}_k)) \log \frac{p(f_i(\mathbf{x}_k))}{p(f_j(\mathbf{x}_k))} \,. \tag{7}$$

Table 1: Memory and inference complexity comparison (CIFAR-10/100 datasets): Number of additional trainable parameters to have 16 ensemble members for different backbones. The inference time (mult-adds) shown corresponds to the mean GPU time (number of multiply-add operations) required to run a forward pass for a batch of size 1 with 16 ensemble members. The bottom section comprises methods whose forward and backward passes are implemented in parallel over ensemble members. Measurements are done on an NVIDIA GeForce GTX 1080 Ti.

Method	Parameters (↓)		Inference time (ms) (↓)		Mult-adds (B) (↓)	
Backbone	VGG-11	ResNet-18	VGG-11	ResNet-18	VGG-11	ResNet-18
MC-Dropout	0.0%	0.0%	16× 2.5	16×2.4	16×0.15	16×0.56
Deep Ensemble	1500%	1500%	16× 2.3	16× 1.8	16×0.15	16×0.56
Masksemble	0.0%	0.0%	20.8	6.6	2.45	8.89
MIMO	1.1%	0.9%	2.7	1.9	0.18	0.58
BatchEnsemble	1.4%	5.2%	2.8	5.2	2.44	8.89
FiLM-Ensemble	0.9%	1.3%	2.8	5.7	2.45	8.89

For both metrics, higher numbers correspond to higher diversity. See Fig. 1. We observe that the average diversity increases with the number of ensemble members. In both metrics, FiLM-Ensemble achieves higher scores than the naïve, explicit deep ensemble. Meaning that the predictions of FiLM-Ensemble are less correlated than those of an equivalent number of networks trained independently with different random seeds. Also, note that with the increasing number of members, improvements in diversity metrics for the naïve explicit deep ensemble are negligible.

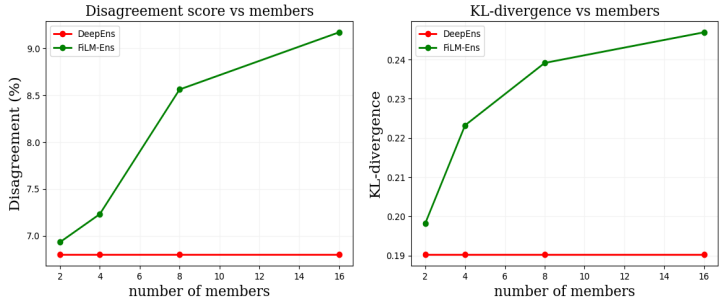


Figure 1: Diversity analysis: CIFAR-10/VGG-11 experiment. With increasing number of members, FiLM-Ensemble achieves more diverse representations. See Section 3.1.

3.2 Computational Cost

We go on to measure the efficiency of our proposed method, both in terms of parameter count and in terms of resources needed for a forward pass. Table 1 (left section) reports the numbers of additional numbers of parameters compared to a single network (i.e., without uncertainty calibration). Due to their design, MC-Dropout and Masksemble do not require any additional parameters. Among the remaining methods, FiLM-Ensemble has the lowest (VGG-11) or second-lowest (Resnet-18) overhead in terms of parameter count, while performing significantly better, as we will see below. In terms of inference complexity (Tab. 1, center and right), FiLM-Ensemble turns out to be very competitive. Only MIMO with a ResNet-18 backbone is significantly faster and lighter, however this comes at a considerable price in terms of accuracy and calibration, see below.

3.3 CIFAR-10 / CIFAR-100

We perform several experiments on widely used benchmarks in computer vision, CIFAR-10 and CIFAR-100. As performance metrics, we plot the test set accuracy and the expected calibration error

Table 2: Classification performance on CIFAR-100 with $M = 4$, using ResNet-18/34 as backbone. Best score for each metric in **bold**, second-best underlined.

Backbone	Resnet-18		Resnet-34	
Method	Acc (↑)	ECE (↓)	Acc (↑)	ECE (↓)
Single Network	78.0 ±0.4	0.046 ±0.001	79.3 ±0.2	0.089 ±0.006
Deep Ensemble	**81.6** ±0.3	0.041 ±0.002	**82.0** ±0.1	**0.044** ±0.002
MC-Dropout	75.5 ±0.6	0.064 ±0.003	72.2 ±0.2	0.079 ±0.004
MIMO	48.0 ±2.6	0.083 ±0.017	56.2 ±4.8	0.132 ±0.055
Masksemble	72.5 ±0.5	0.075 ±0.004	70.1 ±1.2	0.067 ±0.004
BatchEnsemble	77.7 ± 0.1	0.052 ± 0.002	78.3 ± 0.1	0.056 ± 0.002
FiLM-Ensemble	79.4 ±0.2	**0.038** ±0.000	80.2 ±0.1	0.045 ±0.001

(ECE) against the ensemble size M in Fig. 2, for all compared methods. In terms of accuracy (left subfigure), FiLM-Ensemble is outperformed only by the explicit deep ensemble, across all tested values of $M \in \{2, 4, 8, 16\}$. These two surpass all other methods by a clear margin. Surprisingly, we observe that BatchEnsemble generally exhibits a negative correlation between ensemble size and test set accuracy, with $M = 4$ performing best among all settings. MIMO shows very poor performance in this experimental setting in terms of accuracy (and also in ECE). We speculate that its shared backbone probably has a tendency to fragment into largely independent ensemble members of low channel depth, which lack the necessary capacity when using comparatively small networks like VGG-11 or Resnet-18. With regard to ECE (right subfigure), we see all methods improving with growing ensemble size M. FiLM-Ensemble achieves better calibration then the widely used deep ensemble and MC-dropout methods, with significant margins at large ensemble sizes. Masksemble and BatchEnsemble achieve very good calibration for sizes $M \in [4 \dots 16]$, but at the cost of significant drops in classification performance. An attractive feature of FiLM-Ensemble is that it offers a simple mechanism for trading off accuracy against calibration, by tuning the gain factor ρ. See Section 3.5.

In Tab. 2 we quantitatively compare all tested methods on the CIFAR-100 dataset, both with a standard Resnet-18 backbone and with a larger Resnet-34. We observe that with both architectures, and in terms of both predictive accuracy and calibration, FiLM-Ensemble always either ranks first, or it ranks second behind the inefficient, explicit deep ensemble. Also, note that when the capacity of the network is increased (Resnet-18 → Resnet-34), calibration performance improves for some of the implicit methods, but at the cost of reduced accuracy. Overall, the experiments with both variants of CIFAR and all three model architectures confirm that our method presents an excellent trade-off between predictive accuracy, uncertainty calibration and computational efficiency.

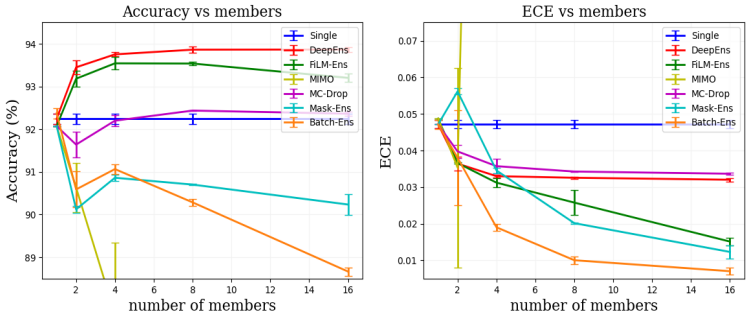


Figure 2: Accuracy and ECE for CIFAR-10, with varying ensemble sizes, using VGG-11 as backbone.

Table 3: Classification performance for retinal Glaucoma images. Best score for each metric in **bold**, second-best underlined.

Method	Accuracy (%) (↑)				ECE (↓)			
# member (M)	2	4	8	16	2	4	8	16
Single	84.4				0.084			
Deep Ensemble	85.6 ± 0.2	85.7 ± 0.3	86.0 ± 0.2	86.8 ± 0.4	0.041 ± 0.002	0.078 ± 0.002	0.091 ± 0.004	0.066 ± 0.003
FSSD Huang et al. [2020]	85.9 ± 0.1				0.047 ± 0.002			
SNGP Liu et al. [2020]	84.7 ± 0.2				0.064 ± 0.003			
pNML Bibas et al. [2021]	85.6 ± 0.1				0.061 ± 0.001			
MC-Dropout	67.0 ± 0.2	78.4 ± 0.5	80.0 ± 0.4	82.7 ± 0.4	**0.002** ± 0.001	0.046 ± 0.009	0.053 ± 0.011	0.051 ± 0.018
MIMO	72.4 ± 1.9	69.8 ± 2.3	68.9 ± 2.4	68.3 ± 2.4	0.049 ± 0.011	0.061 ± 0.013	0.082 ± 0.017	0.041 ± 0.019
Masksemble	82.7 ± 0.5	83.0 ± 0.5	80.2 ± 0.6	81.7 ± 1.1	0.064 ± 0.004	0.049 ± 0.007	$\underline{0.021} \pm 0.010$	0.062 ± 0.012
BatchEnsemble	84.5 ± 0.1	86.5 ± 0.1	86.8 ± 0.2	$\underline{87.1} \pm 0.2$	0.035 ± 0.003	0.063 ± 0.009	0.071 ± 0.002	0.066 ± 0.002
FiLM-Ensemble	86.3 ± 0.1	86.8 ± 0.2	86.9 ± 0.1	**87.8** ± 0.1	0.062 ± 0.001	0.074 ± 0.000	0.068 ± 0.002	0.055 ± 0.001

3.4 Retinal Glaucoma Detection

Glaucoma is currently the leading reason of irreversible blindness in the world. Detection of glaucomatous structural damages and changes is a challenging task in the field of ophthalmology. We evaluate our proposed FiLM-Ensemble, as well as the baselines described above, on the ask of diagnosing Glaucoma and quantifying the uncertainty associated with the prediction, see Table 3. The proposed FiLM-Ensemble achieves the best classification result across all ensemble sizes, and also the best overall result, with $M = 16$. Whereas there is no clear trend with respect to uncertainty calibration.

3.5 6mA Identification

With the 6mA identification task we show that FiLM-Ensemble can also be readily combined with existing models for 1-dimensional sequential genome data. We use the 6mA-rice-Lv dataset and a 1D-CNN architecture whose hyper-parameters have already been optimized for this dataset [Li et al., 2021]. FiLM-Ensemble improves the accuracy and the calibration of that model, see Fig.3. Our method performs on par with the explicit deep ensemble and better than a single instance of the model tuned for the specific task. More importantly, one can reach a significantly better calibration (lower ECE) by increasing the gain ρ, with only a minimal accuracy drop by < 0.25 percent points. Please refer to the Section **??** for more results.

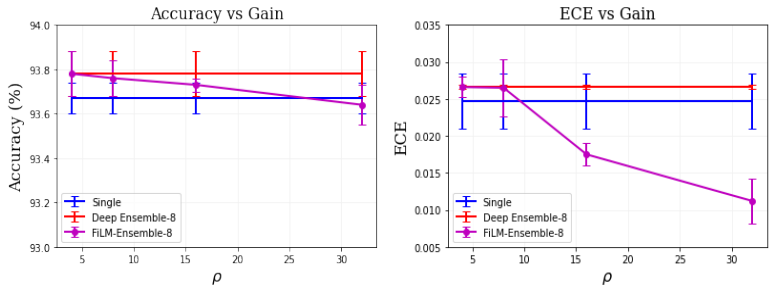


Figure 3: Performance of FiLM-Ensemble with varying gain ρ, c.f. Section 3.5.

3.6 Out-of-Distribution Detection

Domain shift often occurs in medical datasets; thus, detecting out-of-distribution (OOD) samples is an important task in clinical diagnosis. Model uncertainty can be used for OOD detection. For our experiment, we regard the retinal Glaucoma dataset [Diaz-Pinto et al., 2019] as the within-distribution samples and perform OOD detection with the REFUGE dataset [Orlando et al., 2020]. As performance metric, we report AUROC (Area Under the Receiver Operating Characteristic) scores, Table 4. FiLM-Ensemble can detect OOD test samples with significantly higher accuracy than other

Table 4: OOD detection for retinal Glaucoma images. Best score in **bold**, second-best underlined.

Method	AUROC (%) (↑)			
# member	2	4	8	16
Single	68.42			
Deep Ensemble	76.89 ± 0.1	77.91 ± 0.2	78.22 ± 0.1	78.06 ± 0.1
FSSD Huang et al. [2020]	76.42 ± 0.2			
SNGP Liu et al. [2020]	71.25 ± 0.6			
pNML Bibas et al. [2021]	76.68 ± 0.3			
MC-Dropout	68.03 ± 0.3	69.79 ± 0.2	77.94 ± 0.6	72.22 ± 0.2
MIMO	57.33 ± 1.4	59.49 ± 1.2	61.74 ± 2.1	60.52 ± 3.1
Masksemble	71.22 ± 0.5	70.83 ± 0.8	72.04 ± 1.1	70.95 ± 1.4
BatchEnsemble	74.38 ± 0.1	72.61 ± 0.3	75.44 ± 0.2	75.04 ± 1
FiLM-Ensemble	77.02 ± 0.1	77.92 ± 0.2	$\underline{79.43} \pm 0.1$	$\mathbf{79.85} \pm 0.2$

efficient ensemble methods, standard state-of-the-art OOD detection methods, and even the explicit deep ensemble. This suggests that for challenging test samples, which are not adequately represented in the training data, the uncertainties estimated with FiLM-Ensemble are better calibrated.

4 Related Work

4.1 Epistemic Uncertainty Quantification

A large corpus of related work addresses the estimation of epistemic (model) uncertainty in neural networks. At the heart of such modeling is often the concept of *marginalization instead of optimization*, i.e., integrating out a (possibly uncountably infinite) set of models weighted by their posterior probability, instead of committing to a point estimate of that distribution. A multitude of methods have been proposed to implement approximate Bayesian inference w.r.t. the model weights, given the training data and an appropriate prior, a process that is not analytically tractable in general [Kendall and Gal, 2017].

For instance, methods based on *variational inference* [Graves, 2011, Ranganath et al., 2014, Blundell et al., 2015] seek to learn an approximate posterior distribution which is a member of a simpler family of variational distributions. This variational distribution can often be learned using backpropagation [Blundell et al., 2015] and can then be sampled from, or sometimes even used for exact inference. Markov chain Monte Carlo sampling (MCMC) approaches [Neal, 1996, Welling and Teh, 2011, Chen et al., 2014] construct a Markov chain which has the *exact* posterior distribution as its stationary distribution, that can then be employed for sampling. However, in practise, these approaches often fail to sufficiently explore high-dimensional, multi-modal loss landscapes as they are common in deep learning [Gustafsson et al., 2020].

4.2 Ensembles and Sub-networks

In a method referred to as *deep ensembles* [Lakshminarayanan et al., 2017a], a set of M neural network models are randomly and independently initialized, and are subjected to stochastic mini-batch sampling during SGD training. The models generally converge to different minima in the parameter space, and can be considered samples from an approximate posterior [Wilson and Izmailov, 2020, Gustafsson et al., 2020, Izmailov et al., 2021]. Deep ensembles often achieve the best calibration and predictive accuracy [Ovadia et al., 2019, Gustafsson et al., 2020, Ashukha et al., 2020], but suffer from high computational complexity as they require training, storing, and running inference on several full instances of the network.

Many alternative methods have recently been proposed in an attempt to reduce either the computational cost or the storage cost of deep ensembles. Monte Carlo (MC) Dropout [Gal and Ghahramani, 2016] runs multiple forward passes on the dropout layers in order to obtain multiple predictions and obtain an uncertainty estimate. Although the method requires fewer computations compared to deep ensembles, it also leads to less accurate uncertainty estimates. Snapshot ensembles [Huang

et al., 2017] use a cyclic learning rate schedule in order to find multiple local minima, they then store multiple copies of the network to create the ensemble. While this approach reduces the computation cost during training it does not alleviate the storage cost.

BatchEnsembles [Wen et al., 2020] employ multiple low rank matrices, which can be stored efficiently, in order to modulate the parameters of a neural network and thus mimic an ensemble of network models. Masksembles [Durasov et al., 2021] aim to improve the performance of MC Dropout by carefully selecting the dropout masks, used to drop certain features, such that they lead to better uncertainty quantification. Another approach that uses multiple sub-networks is proposed in Havasi et al. [2021]. In this case additional, independent layers are added at the beginning and at the end of the network, in order to obtain multiple prediction with an single backbone model. Although such method seems to reduce the computational resources required at training and inference time, the gain is limited to larger backbones, such as Wide-ResNet (e.g., ResNet28-10), whereas for widely used standard architectures like VGG (e.g., -16), or ResNet (e.g., -34), the approach is not very effective, as shown in our experiments.

Although many methods have been proposed that aim at reducing the computational cost of deep ensembles, none of them appears to clearly outperform most others. In summary, the question how to effectively model uncertainty still remains open.

4.3 Feature-wise Linear Modulation

The idea of controlling the batch normalization parameters to modulate a network's function has been explored by many different authors to accomplish different tasks. Conditional batch normalization (CBN) was proposed by de Vries *et al.*, who achieved good performance in VQA experiments by using an MLP to estimate residuals for normalization parameters used in a pre-trained ResNet [De Vries et al., 2017b]. Perez et al. [2018] proposed FiLM also for solving VQA tasks. Strub et al. [2018] modify FiLM to produce normalization parameters in several stages instead of all at once. This formulation is better able to handle longer conditioning information, such as dialogues instead of questions, and achieved excellent results on the GuessWhat?! visual dialogue task [De Vries et al., 2017a].

Such modulation operations have also been used for image style transfer. It has been observed that the statistics of a feature map, which are associated with and can be controlled by the parameters produced by FiLM, are directly related to the style of an image [Huang and Belongie, 2017]. Dumoulin et al. [2017] succeeded in capturing the styles of artistic paintings, as well as combining extracted styles to create new ones, by using conditional instance normalization, which can be seen as a variation of FiLM. Ghiasi et al. [2017] expand on this work by jointly training a style prediction network and a style transfer network, which also operate based on conditional instance normalization. Finally, Brock et al. [2018] used FiLM for natural image synthesis using generative adversarial networks (GANs). They report that this allowed for a reduction in computation and memory costs, as well as a 37% increase in training speed.

Yang et al. [2018] have used FiLM to modulate the layers of a segmentation network to perform video object segmentation. This made it possible to avoid the fine-tuning process that was used by competing methods, which resulted in a 70$\times$ speed-up while achieving similar accuracy. Feature modulation has also been applied for various other tasks. Oreshkin et al. [2018] used FiLM for task-dependent metric scaling, which allowed them to achieve excellent results in few-shot classification. Vuorio et al. [2019] use FiLM for meta-learning via task-aware modulation. The authors note that FiLM outperforms attention-based modulation in this context, and is more stable. Finally, Vinyals et al. [2019] used FiLM in their AlphaStar neural architecture for multi-agent reinforcement learning. To our knowledge, FiLM has so far not been used for (implicit) model ensembling or uncertainty quantification.

5 Limitations & Future Work

The work presented in this paper gives rise to several questions that can be explored in future work. Using pre-trained models is standard procedure in deep learning applications. It would be useful to explore how FiLM-Ensemble performs when used in conjunction with pre-trained models. This may not be straightforward since FiLM-Ensemble (and also other implicit ensemble methods,

e.g., BatchEnsemble, MIMO) is based on variations between ensemble members, which may be reduced if all members are similarly initialized. Furthermore, self-attention-based models, e.g., Transformers [Vaswani et al., 2017] and Vision Transformers [ViTs, Dosovitskiy et al., 2021] have recently become very popular; therefore it is natural to adapt FiLM-Ensemble to work with such models. Note that layer normalization is standard in Transformers instead of batch normalization, which prevents the straightforward application of the presented method in this case. Also, a number of measures designed to enhance implicit ensembles are orthogonal to our approach and could potentially be combined with FiLM-Ensemble to further improve its performance and uncertainty calibration, while minimally increasing the computational costs. It appears straight-forward to add the selection of independent examples for each member during training [as in MIMO, Havasi et al., 2021], temperature scaling [Guo et al., 2017] or data augmentation strategies [such as, e.g., Ramé et al., 2021].

6 Conclusion

In this paper we present FiLM-Ensemble, a novel implicit deep ensembling method. We achieve high efficiency using a simple yet effective idea – feature-wise linear modulation – which has been shown to be effective in different domains, such as image style transfer, model-agnostic meta-learning, or multi-task learning. Our extensive evaluation shows that FiLM-Ensemble outperforms, or is on par with, state-of-the-art ensemble methods in many different experimental settings.

Broader Impact

Machine learning has recently witnessed a steep increase in model sizes and associated computational costs, and as a consequence a rapid growth in energy consumption. For instance, training state-of-the-art language models like GPT-3 would amount to at least 1400 MWh, or 4.6 million $. Therefore, recently the term *Green AI* has been introduced, referring to AI research that yields novel results while taking into account the computational cost, encouraging a reduction in resources spent. To this extent, we believe that the presented method can foster more efficient ensemble methods and be helpful towards a greener AI.

Acknowledgements We would like to thank Liyuan Zhu for his invaluable contributions to experiments. H.A.G. received funding from German Federal Ministry of Education and Research (BMBF, Grant “GenomeNet” 031L0199B). B. B. and M. R. were supported by the Bavarian Ministry of Economic Affairs, Regional Development and Energy through the Center for Analytics - Data - Applications (ADACenter) within the framework of BAYERN DIGITAL II (20-3410-2-9-8) and the German Federal Ministry of Education and Research and the Bavarian State Ministry for Science and the Arts.

References

A. Ashukha, A. Lyzhov, D. Molchanov, and D. Vetrov. Pitfalls of in-domain uncertainty estimation and ensembling in deep learning. In *International Conference on Learning Representations*, 2020.

K. Bibas, M. Feder, and T. Hassner. Single layer predictive normalized maximum likelihood for out-of-distribution detection. *Advances in Neural Information Processing Systems*, 34:1179–1191, 2021.

C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural network. In *International Conference on Machine Learning*, volume 37, 2015.

A. Brock, J. Donahue, and K. Simonyan. Large scale gan training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2018.

J. L. Campbell and N. Kleckner. E. coli oric and the dnaa gene promoter are sequestered from dam methyltransferase following the passage of the chromosomal replication fork. *Cell*, 62(5):967–979, 1990.

T. Chen, E. Fox, and C. Guestrin. Stochastic gradient Hamiltonian Monte Carlo. In *International Conference on Machine Learning*, volume 32, 2014.

L. Cheng, J. Sun, W. Xu, L. Dong, Y. Hu, and M. Zhou. Oahg: an integrated resource for annotating human genes with multi-level ontologies. *Scientific reports*, 6(1):1–9, 2016.

H. De Vries, F. Strub, S. Chandar, O. Pietquin, H. Larochelle, and A. Courville. Guesswhat?! visual object discovery through multi-modal dialogue. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017a.

H. De Vries, F. Strub, J. Mary, H. Larochelle, O. Pietquin, and A. C. Courville. Modulating early visual processing by language. In *Advances in Neural Information Processing Systems*, volume 30, 2017b.

A. Der Kiureghian and O. Ditlevsen. Aleatory or epistemic? does it matter? *Structural Safety*, 31(2):105–112, 2009.

A. Diaz-Pinto, A. Colomer, V. Naranjo, S. Morales, Y. Xu, and A. F. Frangi. Retinal image synthesis and semi-supervised learning for glaucoma assessment. *IEEE Transactions on Medical Imaging*, 38(9):2211–2218, 2019.

A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.

V. Dumoulin, J. Shlens, and M. Kudlur. A learned representation for artistic style. *ICLR*, 2017.

N. Durasov, T. Bagautdinov, P. Baque, and P. Fua. Masksembles for uncertainty estimation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13539–13548, 2021.

J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*, 2016.

G. Ghiasi, H. Lee, M. Kudlur, V. Dumoulin, and J. Shlens. Exploring the structure of a real-time, arbitrary neural artistic stylization network. *arXiv preprint arXiv:1705.06830*, 2017.

A. Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, volume 24, 2011.

C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning*, 2017.

F. K. Gustafsson, M. Danelljan, and T. B. Schon. Evaluating scalable bayesian deep learning methods for robust computer vision. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020.

S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

M. Havasi, R. Jenatton, S. Fort, J. Z. Liu, J. Snoek, B. Lakshminarayanan, A. M. Dai, and D. Tran. Training independent subnetworks for robust prediction. In *ICLR*, 2021.

G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger. Snapshot ensembles: Train 1, get m for free. In *International Conference on Learning Representations*, 2017.

H. Huang, Z. Li, L. Wang, S. Chen, B. Dong, and X. Zhou. Feature space singularity for out-of-distribution detection. *arXiv preprint arXiv:2011.14654*, 2020.

X. Huang and S. Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *IEEE International Conference on Computer Vision*, 2017.

P. Izmailov, S. Vikram, M. D. Hoffman, and A. G. G. Wilson. What are bayesian neural network posteriors really like? In *International Conference on Machine Learning*, volume 139, 2021.

A. Kendall and Y. Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *Advances in Neural Information Processing Systems*, 2017.

A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, volume 30, 2017a.

B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017b.

N. Lee, T. Ajanthan, and P. H. Torr. Snip: Single-shot network pruning based on connection sensitivity. *ICLR*, 2019.

Y. Li, N. Wang, J. Shi, X. Hou, and J. Liu. Adaptive batch normalization for practical domain adaptation. *Pattern Recognition*, 80:109–117, 2018.

Z. Li, H. Jiang, L. Kong, Y. Chen, K. Lang, X. Fan, L. Zhang, and C. Pian. Deep6ma: A deep learning framework for exploring similar patterns in dna n6-methyladenine sites across different species. *PLoS computational biology*, 17(2):e1008767, 2021.

J. Liu, Z. Lin, S. Padhy, D. Tran, T. Bedrax Weiss, and B. Lakshminarayanan. Simple and principled uncertainty estimation with deterministic deep learning via distance awareness. *Advances in Neural Information Processing Systems*, 33:7498–7512, 2020.

I. Loshchilov and F. Hutter. SGDR: stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

A. Malinin and M. Gales. Predictive uncertainty estimation via prior networks. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

A. Mallya and S. Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018.

R. M. Neal. *Bayesian Learning for Neural Networks*. Springer-Verlag, Berlin, Heidelberg, 1996.

B. Oreshkin, P. Rodríguez López, and A. Lacoste. Tadam: Task dependent adaptive metric for improved few-shot learning. *Advances in Neural Information Processing Systems*, 31, 2018.

J. I. Orlando, H. Fu, J. B. Breda, K. van Keer, D. R. Bathula, A. Diaz-Pinto, R. Fang, P.-A. Heng, J. Kim, J. Lee, et al. Refuge challenge: A unified framework for evaluating automated methods for glaucoma assessment from fundus photographs. *Medical Image Analysis*, 59:101570, 2020.

Y. Ovadia, E. Fertig, J. Ren, Z. Nado, D. Sculley, S. Nowozin, J. Dillon, B. Lakshminarayanan, and J. Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville. FiLM: Visual reasoning with a general conditioning layer. In *AAAI Conference on Artificial Intelligence*, volume 32, 2018.

P. J. Pukkila, J. Peterson, G. Herman, P. Modrich, and M. Meselson. Effects of high levels of dna adenine methylation on methyl-directed mismatch repair in escherichia coli. *Genetics*, 104(4):571–582, 1983.

A. Ramé, R. Sun, and M. Cord. Mixmo: Mixing multiple inputs for multiple outputs via deep subnetworks. In *IEEE/CVF International Conference on Computer Vision*, 2021.

R. Ranganath, S. Gerrish, and D. Blei. Black box variational inference. In *International Conference on Artificial Intelligence and Statistics*, 2014.

F. Strub, M. Seurin, E. Perez, H. De Vries, J. Mary, P. Preux, and A. C. Pietquin. Visual reasoning with multi-hop feature modulation. In *European Conference on Computer Vision*, 2018.

M. Takeda, G. Benitez, and K. Yanai. Training of multiple and mixed tasks with a single network using feature modulation. In *International Conference on Pattern Recognition*, pages 719–735. Springer, 2021.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575 (7782):350–354, 2019.

R. Vuorio, S.-H. Sun, H. Hu, and J. J. Lim. Multimodal model-agnostic meta-learning via task-aware modulation. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *International Conference on Machine Learning*, 2011.

Y. Wen, D. Tran, and J. Ba. Batchensemble: an alternative approach to efficient ensemble and lifelong learning. In *International Conference on Learning Representations*, 2020.

A. G. Wilson and P. Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

L. Yang, Y. Wang, X. Xiong, J. Yang, and A. K. Katsaggelos. Efficient video object segmentation via network modulation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

C. Zhang and Y. Ma. *Ensemble machine learning: methods and applications*. Springer, 2012.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No] **Purely methodological contribution that, when added to a learning system, does not alter its potential societal impacts. If anything, better uncertainty calibration reduces the risk of negative effects caused by undetected prediction errors.**
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No] **license information for the used datasets is public.**
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Training Details

A.1 CIFAR-10 / CIFAR-100

Please refer to Section 2.1.

A.2 Retinal Glaucoma Detection

We followed the same training procedure as for CIFAR-10/100, please refer to Section 2.1. Resnet-18 is used as a backbone. Models are trained with 70% of the dataset for 150 epochs and tested on the test set (30% of the dataset).

A.3 6mA Identification

A 1-dimensional CNN architcture is used, whose hyperparameters such as kernel size and the number of layers are optimized by Li et al. [2021]. The CNN consists of 5 convolutional blocks, where each block contains a 1-dimensional convolution, ReLU activation, batch normalization, and dropout with a rate of 0.5. The convolutional layers have a filter size of 256, kernel size of 10, a stride of 1, and the first convolutional layer have a padding of 5. On top of the last convolutional block, there is a linear layer for predicting the binary labels. Binary cross-entropy is used as a loss. All models are trained for 20 epochs. The initial learning rate of 0.01 is used, as in Li et al. [2021]. Cosine-annealing is used as a learning rate scheduler.

B Additional Results

B.1 EfficientNet as Backbone & More Calibration Metrics

We run more experiments using more modern architecture: EfficientNet Tan and Le [2019] whose proportion of the number of channels vs. the number of layers can vary drastically, compared to Resnets. In this experiment we use two extra calibration metrics: (i) the Brier score Brier et al. [1950] and (ii) the Static Calibration Error (SCE) Nixon et al. [2019]. SCE can be considered an extension of ECE but more accurately account for calibration by considering all classes, instead of just the one with the highest confidence. Table 1 shows that FiLM-Ensemble can also be effectively used in conjunction with EfficientNet architecture. In addition, other calibration metrics are also in favor of FilM-Ensemble.

Table 1: CIFAR-10/EfficientNet-B0 performance comparison. $M \in \{2, 4\}$. The best score for each metric is printed **bold**.

Method	Acc (↑)	ECE (↓)	SCE (↓)	Brier (↓)
Single	90.80	0.0496	0.0106	0.1470
MC-Dropout (2)	90.81	0.0499	0.0107	0.1478
MC-Dropout (4)	90.81	0.0497	0.0107	0.1474
Deep Ensemble (2)	92.67	0.0373	0.0080	0.1146
Deep Ensemble (4)	**93.30**	0.0307	0.0067	**0.1008**
Film-Ensemble (2)	91.62	0.0336	0.0073	0.1291
Film-Ensemble (4)	91.73	**0.0163**	**0.0044**	0.1222

B.2 Calibration-Accuracy Trade-off

As in Section 3.5, we show that one can reach a significantly better calibration (lower ECE) by increasing the gain ρ, with only a minimal accuracy drop. In this case, we use Resnet-34 with 2 ensemble members on Cifar-100 dataset. See Fig. 1. Also note Fig. 2 is an extension of Fig. 3 (of the main text) with various number of ensemble members.

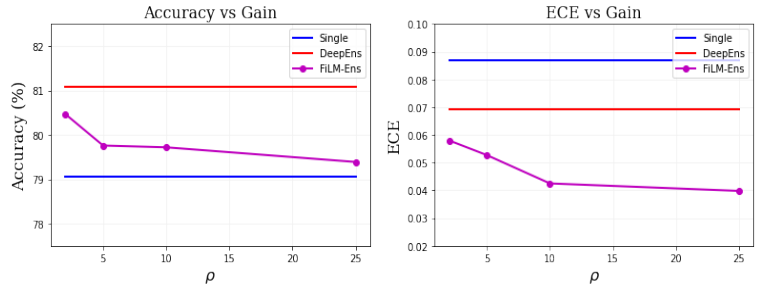


Figure 1: Performance of FiLM-Ensemble with varying gain ρ on Cifar-100 using Resnet-34 as backbone with $M = 2$, c.f. Section B.2.

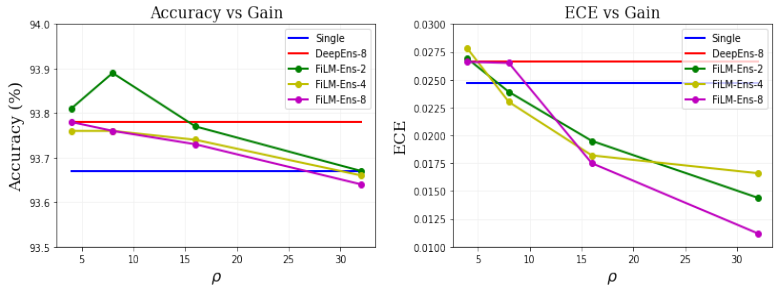


Figure 2: Performance of FiLM-Ensemble with varying gain ρ on 6mA-rice-Lv dataset, using CNN-based Deep6mA as backbone with $M \in \{2, 4, 8\}$, c.f. Section 3.5.

References

G. W. Brier et al. Verification of forecasts expressed in terms of probability. *Monthly weather review*, 78(1):1–3, 1950.

Z. Li, H. Jiang, L. Kong, Y. Chen, K. Lang, X. Fan, L. Zhang, and C. Pian. Deep6ma: A deep learning framework for exploring similar patterns in dna n6-methyladenine sites across different species. *PLoS computational biology*, 17(2):e1008767, 2021.

J. Nixon, M. W. Dusenberry, L. Zhang, G. Jerfel, and D. Tran. Measuring calibration in deep learning. In *CVPR Workshops*, volume 2, 2019.

M. Tan and Q. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

6.3 Diversified Ensemble of Independent Sub-Networks for Robust Self-Supervised Representation Learning

Contributing Article

Vahidi A, Wimmer L, Gündüz HA, Bischl B, Hüllermeier E, Rezaei M (2024). "Diversified Ensemble of Independent Sub-networks for Robust Self-supervised Representation Learning." In A Bifet, J Davis, T Krilavičius, M Kull, E Ntoutsi, I Žliobaitė (eds.), *Machine Learning and Knowledge Discovery in Databases. Research Track. ECML PKDD 2024*, pp. 38–55. Springer Nature Switzerland, Cham. doi:10.1007/978-3-031-70341-6_3

Acknowledgement: Reproduced with permission from Springer Nature.

Declaration of Contributions

Hüseyin Anil Gündüz contributed to this paper as a co-author with the following significant contributions:
Hüseyin Anil Gündüz designed, developed the code, and ran all the experiments of the T6SS identification task presented in the paper, which is presented in Figure 4 of the main paper, and Table 7 in the Appendix. These experiments show that the proposed method can also be applied to 1-dimensional sequential genomics data. Hüseyin Anil Gündüz also edited the paper.

Contribution of the coauthors:
Amirhossein Vahidi contributed to the design, code development, running, and evaluation of most experiments as the first author. Most of the paper is written by Amirhossein Vahidi. Laura Wimmer designed, developed the code, and ran the experiments of the natural language processing (NLP) task in the paper. These experiments show that the proposed method can also be applied to the NLP data. Mina Rezaei was the main supervisor of the paper. Mina Rezaei wrote some parts of the paper.

All authors helped with the editing of the paper.

Note: The project was started as Amirhossein Vahidi's master's thesis, which he also completed. The thesis was supervised by Mina Rezaei, who provided supervision and assistance throughout the entire process.

Diversified Ensemble of Independent Sub-networks for Robust Self-supervised Representation Learning

Amihossein Vahidi[1,2], Lisa Wimmer[1,2], Hüseyin Anil Gündüz[1,2], Bernd Bischl[1,2], Eyke Hüllermeier[2,3], and Mina Rezaei[1,2]( )

[1] Department of Statistics, LMU Munich, Munich, Germany
[2] Munich Center for Machine Learning, Munich, Germany
mina.rezaei@stat.uni-muenchen.de
[3] Institute of Informatics LMU Munich, Munich, Germany

Abstract. Ensembling a neural network is a widely recognized approach to enhance model performance, estimate uncertainty, and improve robustness in deep supervised learning. However, deep ensembles often come with high computational costs and memory demands. In addition, the efficiency of a deep ensemble is related to diversity among the ensemble members, which is challenging for large, over-parameterized deep neural networks. Moreover, ensemble learning has not yet seen such widespread adoption for unsupervised learning and it remains a challenging endeavor for self-supervised or unsupervised representation learning. Motivated by these challenges, we present a novel self-supervised training regime that leverages an ensemble of independent sub-networks, complemented by a new loss function designed to encourage diversity. Our method efficiently builds a sub-model ensemble with high diversity, leading to well-calibrated estimates of model uncertainty, all achieved with minimal computational overhead compared to traditional deep self-supervised ensembles. To evaluate the effectiveness of our approach, we conducted extensive experiments across various tasks, including in-distribution generalization, out-of-distribution detection, dataset corruption, and semi-supervised settings. The results demonstrate that our method significantly improves prediction reliability. Our approach not only achieves excellent accuracy but also enhances calibration, improving on important baseline performance across a wide range of self-supervised architectures in computer vision, natural language processing, and genomics data.

1 Introduction

Ensemble learning has become a potent strategy for enhancing model performance in deep learning [19,28]. This method involves combining the outputs of multiple independently trained neural networks, all using the same architecture

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/978-3-031-70341-6_3.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024
A. Bifet et al. (Eds.): ECML PKDD 2024, LNAI 14941, pp. 38–55, 2024.
https://doi.org/10.1007/978-3-031-70341-6_3

and same training dataset but differing in the randomized configurations of their initialization and/or training. Despite its remarkable effectiveness, training deep ensemble models poses several challenges: i) The high performance achieved by deep ensembles comes with a significant increase in computational costs. Running multiple neural networks independently demands more resources and time. ii) Maintaining diversity among ensemble members – a property often critical to success – becomes increasingly difficult for large, over-parameterized deep neural networks [37] in which the main source of diversity stems from random weight initialization. iii) Most of the existing literature focuses on deep ensembles for supervised models. Adapting these approaches to unsupervised and self-supervised models requires careful consideration and evaluation to ensure comparable performance.

In recent years, self-supervised learning methods have achieved cutting-edge performance across a wide range of tasks in natural language processing (NLP; [2,9]), computer vision [5,40], multimodal learning [36], and bioinformatics [18]. In contrast to supervised techniques, these models learn representations of the data without relying on costly human annotation. Despite remarkable progress in recent years, self-supervised models do not allow practitioners to inspect the model's confidence. This problem is non-trivial given the degree to which critical applications rely on self-supervised methods. As recently discussed by LeCun[1], representing predictive uncertainty is particularly difficult in self-supervised contrastive learning for computer vision. Therefore, quantifying the predictive uncertainty of self-supervised models is critical to more reliable downstream tasks. Here, we follow the definition of reliability as described by Plex [44], in which the ability of a model to work consistently across many tasks is assessed. In particular, [44] introduces three general desiderata of reliable machine learning systems: a model should generalize robustly to *new tasks*, as well as *new datasets*, and represent the associated *uncertainty* in a faithful manner.

In this paper, we introduce a novel, robust, and scalable framework for ensembling *self-supervised learning* while *preserving performance* with a negligible increase in computational cost and *encouraging diversity among the ensemble of sub-networks*.

Our contributions can be summarized as follows:

- We propose a novel, scalable ensemble component of self-supervised learning that is robust, efficient and enhances performance in various downstream tasks.
- We develop a complementary loss function to enforce diversity among the independent sub-networks.
- We perform extensive empirical analyses to highlight the benefits of our approach. We demonstrate that this inexpensive modification achieves very competitive (in most cases, better) predictive performance: 1) on in-distribution (IND) and out-of-distribution (OOD) tasks; 2) in semi-supervised settings; 3) learns a better predictive performance-uncertainty trade-off than compared

[1] https://ai.facebook.com/blog/self-supervised-learning-the-dark-matter-of-intelligence/.

baselines. (i.e., exhibits high predictive performance and low uncertainty on IND datasets as well as high predictive performance and high uncertainty on OOD datasets).

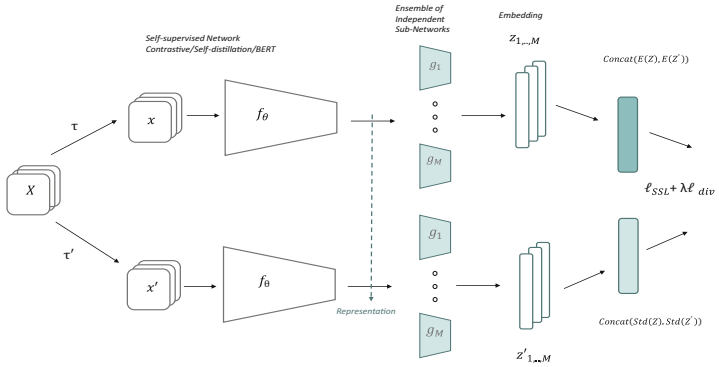


Fig. 1. Illustration of our proposed method. Given a batch $\boldsymbol{X}$ of input samples, two different views $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{x}}'$ are produced for each sample, which is then encoded into representations by the encoder network $f_{\boldsymbol{\theta}'}$. The representations are projected to the ensemble of independent sub-networks g_m, where each sub-network produces embedding vectors $\boldsymbol{z}$ and $\boldsymbol{z}'$. The mean value of these embeddings is passed to the self-supervised loss, while their standard deviation is used for the diversity loss. Finally, the total loss is computed by a combination of the two loss components.

2 Related Work

Self-supervised Learning. For most large-scale modeling problems, learning under full supervision is severely inhibited by the scarcity of annotated samples. Self-supervised learning techniques, which solve *pretext tasks* [9] to generate labels from (typically abundant) unlabeled data, have proven to be a powerful remedy to this bottleneck. The learned feature maps can serve as a starting point for *downstream* supervised tasks, such as classification, object detection, or sentiment analysis, with a substantially reduced need for labeled examples [25]. Alternatively, the downstream application may directly use the extracted representation for problems such as anomaly OOD detection. While there have been attempts to make pretraining more robust by preventing embedding collapse [40] or boosting performance in OOD detection [39, 45], the aspect of *uncertainty-awareness* has been studied to a lesser extent in the self-supervised context. Motivated by this, we present a simple way to make self-supervised learning robust during pretext-task learning.

Ensemble Learning. Deep ensembles [28] comprise a set of M neural networks that independently train on the same data using random initialization.

Deep ensembles often outperform other approaches in terms of calibration and predictive accuracy [38], but their naive application incurs high computational complexity, as training, memory, and inference cost multiply with the number of base learners. BatchEnsemble [46] introduces multiple low-rank matrices, with little training and storage demand, whose Hadamard products with a shared global weight matrix mimic an ensemble of models. Masksensemble [13] builds upon Monte Carlo dropout [15] and proposes a learnable (rather than random) selection of masks used to drop specific network neurons. MIMO [20] uses ensembles of sub-networks diverging only at the beginning and end of the parent architecture – thus sharing the vast majority of weights – to obtain multiple predictions with a single forward pass.

Diversity in Ensembles: Diversity is a crucial component for successful ensembles. [37] classify existing approaches for encouraging diversity among ensemble members into three groups: i) methods that force *diversity in gradients* with adaptive diversity in prediction [35], or using joint gradient phase and magnitude regularization (GPMR) between ensemble members [7], ii) methods focusing on *diversity in logits*, improving diversity with regularization and estimating the uncertainty of out-of-domain samples [30], iii) methods promoting *diversity in features* that increase diversity with adversarial loss [4] for conditional redundancy [37], information bottleneck [42], or $f1$-divergences [4]. Our method belongs to this last category, where our loss function encourages the diversity of feature maps.

3 Method

We propose a simple principle to 1) make self-supervised pretraining robust with an ensemble of diverse sub-networks, 2) improve predictive performance during pretraining of self-supervised deep learning, 3) while keeping an efficient training pipeline.

As depicted in Fig. 1, our proposed method can be readily applied to recent trends in self-supervised learning [3,5,10,17,18,26] and is based on a joint embedding architecture. In the following sections, we first describe our proposed ensemble model, followed by the diversity loss, and then a discussion on diversity, and computational cost.

3.1 Robust Self-supervised Learning via Independent Sub-networks

Setting. Given a randomly sampled mini-batch of data $\boldsymbol{X} = \{\boldsymbol{x}_k\}_{k=1}^{N} \subset \mathcal{X} \subseteq \mathbb{R}^p$, the transformer function derives two augmented views $\tilde{\boldsymbol{x}} = \tau(\boldsymbol{x}), \tilde{\boldsymbol{x}}' = \tau'(\boldsymbol{x})$ for each sample in $\boldsymbol{X}$. The augmented views are obtained by sampling τ, τ' from a distribution over suitable data augmentations, such as masking parts of sequences [1,10], partially masking image patches [21], or applying image augmentation techniques [5].

The two augmented views $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{x}}'$ are then fed to an encoder network $f_{\boldsymbol{\theta}}$ with trainable parameters $\boldsymbol{\theta} \subseteq \mathbb{R}^d$. The encoder (e.g., ResNet-50 [22], ViT [12])

maps the distorted samples to a set of corresponding features. We call the output of the encoder the *representation*. Afterward, the representation features are transformed by M independent sub-networks $\{g_{\phi_m}\}_{m=1}^{M}$ with trainable parameters $\boldsymbol{\phi}_m$ to improve the feature learning of the encoder network. The ensemble constructs from the representation M different q-dimensional *embedding* vectors $\{\boldsymbol{z}_m\}_{m=1}^{M}$, $\{\boldsymbol{z}'_m\}_{m=1}^{M}$, respectively, for $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{x}}'$. We modify the conventional self-supervised loss and replace the usual $\boldsymbol{z}_m$ by the mean value $\bar{\boldsymbol{z}} = (\boldsymbol{z}_1 + \ldots + \boldsymbol{z}_M)/M$, and similarly $\boldsymbol{z}'_m$ by $\bar{\boldsymbol{z}}'$. Averaging over the embeddings generated by the M sub-networks increases robustness, which in turn may help to improve predictive performance in downstream tasks.

Self-supervised Loss. In the case of contrastive learning [5], the self-supervised loss ℓ_{ssl} with temperature $t > 0$ and cosine similarity $\text{sim}(\cdot, \cdot)$ is computed as:

$$\ell_{\text{ssl}}\left(\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{x}}'_k\right) = -\log \frac{\exp(\text{sim}(\bar{\boldsymbol{z}}_k, \bar{\boldsymbol{z}}'_k)/t)}{\sum_{i=1}^{2N} \mathbb{I}_{[k \neq i]} \exp(\text{sim}(\bar{\boldsymbol{z}}_k, \bar{\boldsymbol{z}}_i)/t)}. \tag{1}$$

Diversity Loss. Since diversity is a key component of successful model ensembles [14], we design a new loss function for encouraging diversity during the training of the sub-networks. We define the diversity regularization term ℓ_{div} as a hinge loss over the difference of the standard deviation across the embedding vectors $\{\boldsymbol{z}_{k,m}\}_{m=1}^{M}$, $\{\boldsymbol{z}'_{k,m}\}_{m=1}^{M}$ to a minimum diversity of $\alpha > 0$. The standard deviation is the square root of the element-wise variance $\{\sigma^2_{k,o}\}_{o=1}^{q}$:

$$\sigma^2_{k,o} = \tfrac{1}{M-1} \textstyle\sum_{m=1}^{M} (z_{k,m,o} - \bar{z}_{k,o})^2 + \epsilon\,,$$

where we add a small scalar $\epsilon > 0$ to prevent numerical instabilities. The diversity regularization function is then given by:

$$\begin{aligned} \ell_{\text{div}}\left(\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{x}}'_k\right) = \textstyle\sum_{o=1}^{q} \max\left(0, \alpha - \sigma_{k,o}\right) \\ + \max(0, \alpha - \sigma'_{k,o})\,, \end{aligned} \tag{2}$$

where σ and σ' indicate standard deviation for the input sample and augmented views, respectively.

Total Loss. The objective of the diversity loss is to encourage disagreement among sub-networks by enforcing the element-wise standard deviations to be close to $\alpha > 0$ and to thus prevent the embeddings from collapsing to the same vector. Figure 2a underlines the importance of the diversity loss on the total sum of standard deviations between different sub-networks, which increases by adding this loss. The total loss is calculated by combining the self-supervised loss (Eq. 1) and the diversity loss (Eq. 2), where the degree of regularization is controlled by a tunable hyperparameter $\lambda \geq 0$:

$$\ell\left(\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{x}}'_k\right) = \ell_{\text{ssl}}\left(\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{x}}'_k\right) + \lambda \cdot \ell_{\text{div}}\left(\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{x}}'_k\right). \tag{3}$$

Finally, the total loss is aggregated over all the pairs in minibatch $\boldsymbol{X}$:

$$\mathcal{L}_{\text{total}} = \frac{1}{N}\sum_{k=1}^{N} \ell\left(\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{x}}'_k\right). \tag{4}$$

Gradients. Consider the output of the encoder $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = b$ and the output of the m-th linear sub-network $\boldsymbol{z}_m = g_{\phi_m}(b) = w_m \cdot b$. The weight w_m is updated by two components during backpropagation, the first of which depends on the self-supervised loss and is the same for the entire ensemble, while the second term depends on the diversity loss and is different for each sub-network. Given Eq. 2, we simplify the equation by vector-wise multiplication since the sub-networks are linear; furthermore, we omit the numerical stability term since it does not have an effect on the derivative. The element-wise standard deviation can be computed as follows:

$$\sigma_{k,o} = \left(\frac{1}{M-1}\sum_{m=1}^{M}(\boldsymbol{z}_{k,m,o} - \bar{\boldsymbol{z}}_{k,o})^2\right)^{\frac{1}{2}}. \tag{5}$$

Consider Eq. 2 for aggregating the element-wise standard deviations for one observation ($\boldsymbol{x}$) and assume $\sigma_k < \alpha$; otherwise, the diversity loss is zero. The derivative of the loss with respect to $\boldsymbol{z}_{k,\hat{m},o}$, $\hat{m} \in 1, \ldots, M$, is then given as follows:

$$\frac{\partial\left(\ell_{\text{div}}\right)}{\partial \boldsymbol{z}_{k,\hat{m},o}} = \frac{-A}{M-1} \cdot (\boldsymbol{z}_{k,\hat{m},o} - \bar{\boldsymbol{z}}_{k,o}), \tag{6}$$

where $A := \frac{1}{M-1}\sum_{m=1}^{M}(\boldsymbol{z}_{k,m,o} - \bar{\boldsymbol{z}}_{k,o})^2)$. The proof is provided in the appendix (see Theoretical Supplement).

In the optimization step of stochastic gradient descent (SGD), the weight of sub-network $\hat{m}$ is updated by:

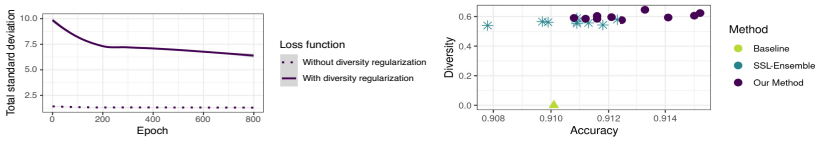
$$\eta \cdot \nabla_{w_{\hat{m},o}} \ell_{\text{div}} = -C \cdot (\boldsymbol{z}_{k,\hat{m},o} - \bar{\boldsymbol{z}}_{k,o}), \tag{7}$$

where $\eta > 0$ is the learning rate, and C is constant with respect to $w_{\hat{m},o}$, which depends on the learning rate, number of sub-networks, A, and b. The proof is provided in Appendix (see Theoretical Supplement).

Equation 7 shows the updating step in backpropagation. Hyperparameter α prevents $\boldsymbol{z}_{k,\hat{m},o}$ from collapsing to a single point. Hence, $w_{\hat{m},o}$ is updated in the opposite direction of $\bar{\boldsymbol{z}}_{k,o}$, so the diversity loss prevents weights in the sub-networks from converging to the same values.

3.2 Empirical Analysis of Diversity

Diversity of ensemble members is an important feature for powerful model ensembles and reflects the degree of independence among its members [34, 49]. We follow [14] to quantify the diversities among the ensemble of sub-networks. Specifically, we report the diversities in terms of *disagreement score* between the members' predictive distributions and a baseline. Diversity disagreement



(a) (b)

Fig. 2. (a) **Total standard deviation**: sum of all standard deviations between independent sub-networks during training. Training with diversity loss (Eq. 2) increases the standard deviation and improves the diversity between independent sub-networks. (b) **Diversity analysis**: prediction diversity disagreement vs. achieved accuracy on CIFAR-10. Our method is on par with the deep self-supervised ensemble in terms of both accuracy and diversity disagreement. Models in the top right corner are better.

is defined as *distance disagreement* divided by 1− *accuracy*, where the distance disagreement between two classification models h_i and h_j is calculated as $\frac{1}{N}\sum_{k=1}^{N}\left[h_i(\mathbf{x}_k) \neq h_j(\mathbf{x}_k)\right]$, with N denoting the number of samples. Figure 2b compares the diversity disagreement between our method with 10-sub-networks, a deep ensemble with 10 members, and the single-network baseline. The results clearly indicate that our proposed method achieves comparable results with deep self-supervised ensembles in terms of both accuracy and diversity disagreement.

3.3 Computational Cost and Efficiency Analysis

We analyze the efficiency of our proposed method in Table 1. SSL-Ensemble increases memory and computational requirements compared to the baseline by 200% and 900% for 3 and 10 members, respectively. The increase in the number of parameters is 32% and 143%, and the increase in computational requirement is $\sim 0-6\%$ for our method.

Table 1. Computational cost in 4 DGX-A100 40G GPUs (PyTorch) on CIFAR 10.

Method	Members	Parameters(M)	Memory/GPU	Time/800-ep.
Baseline (SSL)	1	28	9 G	3.6 (h)
SSL-Ensemble	3	3×28	3×9 G	3× 3.6 (h)
SSL-Ensemble	10	10×28	10×9 G	10×3.6 (h)
Our method	3	37	9.2 G	3.6 (h)
Our method	10	68.1	10 G	3.8 (h)

4 Experimental Setup

We perform several experiments with a variety of self-supervised methods to examine our hypothesis for robustness during both pretext-task learning and downstream tasks (fine-tuning).

Deep Self-supervised Network Architecture. Our proposed approach builds on two recent popular self-supervised models in computer vision: i) **SimCLR** [5] is a contrastive learning framework that learns representations by maximizing agreement on two different augmentations of the same image, employing a contrastive loss in the latent embedding space of a convolutional network architecture (e.g., ResNet-50 [22]), and ii) **DINO** [3] is a self-distillation framework in which a student vision transformer (ViT; [11]) learns to predict global features from local image patches supervised by the cross-entropy loss from a momentum teacher ViT's embeddings. Furthermore, we study the impact of our approach in NLP and modify **SCD** [26], which applies the bidirectional training of transformers to language modeling. Here, the objective is self-supervised contrastive divergence loss. Lastly, we examine our approach on **Self-GenomeNet** [18], a contrastive self-supervised learning algorithm for learning representations of genome sequences. More detailed descriptions of the employed configurations are provided in Appendix (see Implementation Details)

Deep Independent Sub-networks. We implement M independent sub-networks on top of the encoder, for which many possible architectures are conceivable. For our experiments on computer vision datasets, we consider an ensemble of sub-network architecture where each network includes a multi-layer perceptron (MLP) with two layers of 2048 and 128 neurons, respectively, with ReLU as a non-linearity and followed by batch normalization [24]. Each sub-network has its own independent set of weights and learning parameters. For the NLP dataset, the projector MLP contains three layers of 4096 neurons each, also using ReLU activation's as well as batch normalization. For the genomics dataset, our ensemble of sub-networks includes one fully connected layer with an embedding size of 256.

Optimization. For all experiments on image datasets based on DINO and SimCLR, we follow the suggested hyperparameters and configurations by the paper [3,5]. Implementation details for pretraining with DINO on the 1000-classes ImagetNet dataset without labels are as follows: coefficients ϵ, α, and λ are respectively set to $0.0001, 0.15$, and 2 in Eq. 2, 2, and 3. We provide more details in ablation studies (Sect. 6) on the number of sub-networks and the coefficients λ and α used in the loss function. The encoder network $f_{\boldsymbol{\theta}}$ is either a ResNet-50 [22] with 2048 output units when the baseline is SimCLR [5] or ViT-s [12] with 384 output units when the baseline is DINO [3]. The best prediction and calibration performance is achieved when the number of sub-networks is 5. We followed the training protocol and settings suggested by [3].

Datasets. We use the following datasets in our experiments: **CIFAR-10/100** [27] are subsets of the tiny images dataset. Both datasets include 50,000 images for training and 10,000 validation images of size 32×32 with 10 and 100 classes, respectively. **SVH** [32] is a digit classification benchmark dataset that

contains 600,000 32×32 RGB images of printed digits (from 0 to 9) cropped from pictures of house number plates. **ImageNet** [8], contains 1,000 classes, with 1.28 million training images and 50,000 validation images. For the NLP task, we train on a dataset of 1 million randomly sampled sentences from **Wikipedia articles** [23] and evaluate our models on 7 different semantic textual similarity datasets from the SentEval benchmark suite [6]: **MR** (movie reviews), **CR** (product reviews), **SUBJ** (subjectivity status), **MPQA** (opinion-polarity), **SST-2** (sentiment analysis), **TREC** (question-type classification), and **MRPC** (paraphrase detection). The **T6SS** effector protein dataset is a public real-world bacteria dataset (SecReT6, [29]) with actual label scarcity. The sequence length of the genome sample is 1000nt in all experiments.

Tasks. We examine and benchmark a model's performance on different tasks considering evaluation protocols by self-supervised learning [5] and Plex's benchmarking tasks [44]. Specifically, we evaluate our model on the basis of **uncertainty-aware IND generalization**, **OOD detection**, **semi-supervised learning**, **corrupted dataset evaluation** (see Sect. 5), and **transfer learning to other datasets and tasks** (see Appendix: Transfer to Other Tasks and Datasets)

Evaluation Metrics. We report prediction/calibration performance with the following metrics, where upward arrows indicate that higher values are desirable, *et vice versa*. **Top-1 accuracy** ↑: share of test observations for which the correct class is predicted. **AUROC** ↑: area under the ROC curve arising from different combinations of false-positive and false-negative rates (here: with positive and negative classes referring to being in and out of distribution, respectively) for a gradually increasing classification threshold. **Negative log-likelihood (NLL)** ↓: negative log-likelihood of test observations under the estimated parameters. **Expected calibration error (ECE)**; [31] ↓: mean absolute difference between accuracy and confidence (highest posterior probability among predicted classes) across equally-spaced confidence bins, weighted by relative number of samples per bin. **Thresholded adaptive calibration error (TACE)**; [33]) ↓: modified ECE with bins of equal sample size, rather than equal interval width, and omitting predictions with posterior probabilities falling below a certain threshold (here: 0.01) that often dominate the calibration in tasks with many classes.

Compared Methods. We compare our method to the following contenders. **Baseline:** self-supervised architectures (i.e., SimCLR, DINO, SCD, or Self-GenomeNet, depending on the task). **SSL-Ensemble**: deep ensemble comprising a multiple of the aforementioned baseline networks. **Monte Carlo (MC) dropout**: [15] baseline networks with dropout regularization applied during pretraining of baseline encoder. **BatchEnsemble**: baseline encoder with BatchEnsemble applied during pretraining.

5 Results and Discussion

In-Distribution Generalization. IND generalization (or *prediction calibration*) quantifies how well model confidence aligns with model accuracy. We

perform several experiments on small and large image datasets as well as the genomics sequence dataset to evaluate and compare the predictive performance of our proposed model in IND generalization. Here, the base encoder $f_{\boldsymbol{\theta}}$ is frozen after unsupervised pretraining, and the model is trained on a supervised linear classifier. The linear classifier is a fully connected layer followed by softmax, which is placed on top of $f_{\boldsymbol{\theta}}$ after removing the ensemble of sub-networks. High predictive scores and low uncertainty scores are desired. Figure 3 illustrates the predictive probability of correctness for our model on CIFAR-10, CIFAR-100, ImageNet, and T6SS datasets in terms of Top-1 accuracy, ECE, and NLL, respectively. Based on Fig. 3, our method achieves better calibration (ECE and NLL) than the deep ensemble of self-supervised models. The discrepancy in performance between our model and the deep ensemble can be explained by various factors, including differences in uncertainty modeling, complexity, and robustness. While the deep ensemble excels in top-1 accuracy, our model's superior ECE and NLL scores indicate better-calibrated and more reliable predictions, which are essential for safety-critical applications and decision-making under uncertainty.

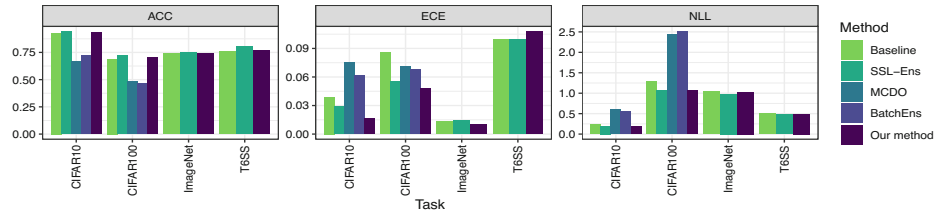


Fig. 3. IND generalization in terms of (a) **Top-1 Accuracy** (b) **ECE** (c) **NLL** averaged over in-distribution on test samples of *CIFAR-10/100, ImageNet, T6SS* datasets. Here, we compare our method with the ensemble of deep self-supervised networks (SSL-Ens), as well as the baseline.

Out-of-Distribution Detection. OOD detection shows how well a model can recognize test samples from the classes that are unseen during training [16]. We perform several experiments to compare the model generalization from IND to OOD datasets and to predict the uncertainty of the models on OOD datasets. Evaluation is performed directly after unsupervised pretraining without a fine-tuning step. Table 2 shows the AUROC on different OOD sets for our model, baseline, and deep self-supervised ensemble. Our approach improves overall compared to other methods.

Semi-supervised Evaluation. We explore and compare the performance of our proposed method in the low-data regime. Again, the encoder $f_{\boldsymbol{\theta}}$ is frozen after self-supervised pretraining, and the model is trained on a supervised linear classifier using 1% and 10% of the dataset. The linear classifier is a fully connected layer followed by softmax. Table 3 shows the result in terms of top-

Table 2. OOD detection. Results reported using AUROC show our method enhances the baseline up to 6%.

IND	OOD	Baseline	SSL-Ensemble	Our method
CIFAR-100	SVHN	84.22	84.95	**88.00**
	Uniform	91.65	90.53	**97.57**
	Gaussian	90.00	89.42	**94.10**
	CIFAR-10	74.71	74.80	**75.18**
CIFAR-10	SVHN	95.03	96.68	**97.07**
	Uniform	96.73	91.64	**99.05**
	Gaussian	96.39	93.24	**99.24**
	CIFAR-100	91.79	91.59	**91.87**

Table 3. Semi-supervised evaluation: Top-1 accuracy (ACC), ECE, and NLL for semi-supervised CIFAR-10/100 classification using 1% and 10% training examples.

Method	CIFAR-10 (1%)			CIFAR-10 (10%)			CIFAR-100 (1%)			CIFAR-100 (10%)		
	ACC	ECE	NLL	ACC	ECE	NLL	ACC	ECE	NLL	ACC	ECE	NLL
Baseline	89.1	0.075	0.364	91.1	0.039	0.274	56.2	0.097	2.01	59.5	0.086	1.79
SSL-Ensemble	90.1	0.056	0.334	92.2	0.050	0.257	59.7	0.081	1.86	62.6	0.053	1.48
Our method	90.4	0.018	0.296	92.6	0.016	0.249	59.3	0.060	1.71	62.4	0.042	1.56

1 accuracy, ECE, and NLL. The results indicate that our method outperforms other methods in the low-data regime – in terms of calibration.

Corrupted Dataset Evaluation. Another important component of model robustness is its ability to make accurate predictions when the test data distribution changes. Here, we evaluate model robustness under *covariate shift*. We employ a configuration similar to the one found in [44]. Figure 4 summarizes the improved performance across metrics of interest. The results confirm that our method outperforms the baseline and achieves comparable predictive performance as a deep self-supervised ensemble – both in terms of calibration (TACE) and AUROC.

Transfer to Other Tasks and Datasets. We further assess the generalization capacity of the learned representation on learning a new task in NLP. We train our model without any labels on a dataset of sentences from Wikipedia [23] and fine-tune the pretrained representation on seven different semantic textual similarity datasets from the SentEval benchmark suite [6]: **MR** (movie reviews), **CR** (product reviews), **SUBJ** (subjectivity status), **MPQA** (opinion-polarity), **SST-2** (sentiment analysis), **TREC** (question-type classification), and **MRPC** (paraphrase detection). Then, we evaluate the test set of each dataset. Figure 5 provides a comparison of the transfer learning performance of our self-supervised approach for different tasks. Our results in Fig. 5 indicate that our approach performs comparably to or better than the baseline method. We test the per-

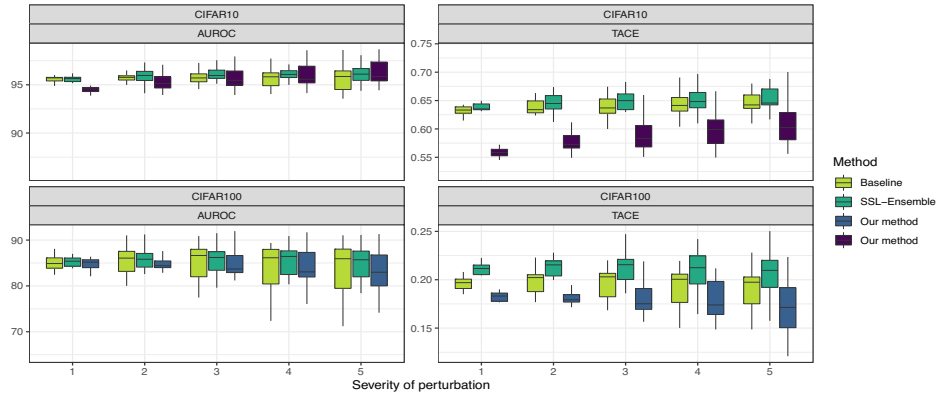


Fig. 4. Performance under **dataset corruption** (CIFAR-10/100 with five levels of increasing perturbation), evaluation in terms of AUROC and TACE for several types of corruption (vertical spread).

formance of the trained model on ImageNet [8] on CIFAR-10 [27] dataset where the model is trained for 100 epochs (Table 4).

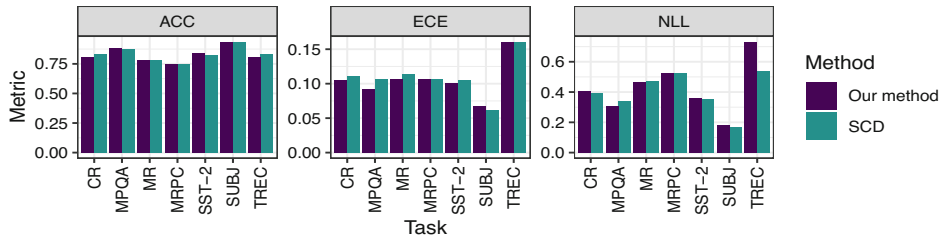


Fig. 5. Transfer to other dataset and tasks: Comparision of Sentence embedding performance on semantic textual similarity tasks.

Table 4. Transfer to other dataset: Expected calibration error averaged over uncertainty-aware evaluation on CIFAR-10 datasets.

Method	ACC (%) (↑)	ECE (↓)	NLL (↓)	TACE (↓)
Baseline	73.5	0.038	0.78	0.20
Our method	73.9	0.030	0.75	0.18

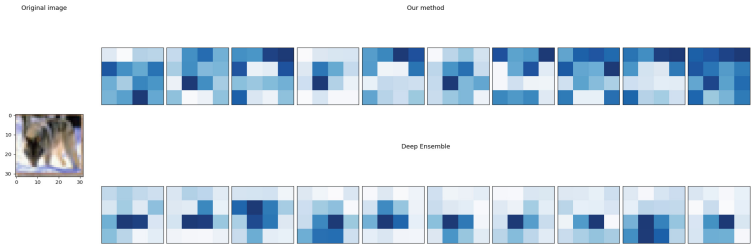


Fig. 6. We compare the feature diversity for different subnetworks and ensemble members. The top images are for different sub-networks, and the bottom images are for different ensemble members. We used Grad-CAM [41] for visualization.

6 Ablation Study

In order to build intuition around the behavior and the observed performance of the proposed method, we further investigate the following aspects of our approach in multiple ablation studies exploring: (1) the number M of sub-networks, (2) the role of each component of the proposed loss, and (3) analysis of diversity with visualization of the gradients of subnetworks. We also present more results on (4) the impact of our approach during pretraining vs. at the finetuning step, (5) the size of sub-networks, and (6) the impact of model parameters.

Number of Sub-networks. We train M individual deep neural networks on top of the representation layer. The networks receive the same inputs but are parameterized with different weights and biases. Here, we provide more details regarding our experiments on IND generalization by considering varying M. Figure 7a compares the performance in terms of top-1 accuracy, ECE, and NLL for CIFAR-10 and CIFAR-100. Based on the quantitative results depicted in Fig. 7a, the predictive performance improves in both datasets when increasing the number of sub-networks (M) until a certain point. For example, in the case of CIFAR-10, when $M = 3$, our performance is 91.9%; increasing M to 10 levels top-1 accuracy up to 92.6%, while the ECE and NLL decrease from 0.026 and 0.249 to 0.023 and 0.222, respectively. These findings underline that training our sub-networks with a suitable number of heads can lead to a better representation of the data and better calibration. Recently [43, 47] provided a theoretical statement as well as experimental results that projection heads help with faster convergence.

Analysis of Loss. The total loss (Eq. 3) is calculated by the combination of self-supervised loss (Eq. 1) and diversity loss (Eq. 2), where the mean value of the embeddings across the ensemble of sub-networks is fed to the self-supervised loss, and the corresponding standard deviation is used for the diversity loss. First, we note that the use of our diversity regularizer indeed improves calibration and provides better uncertainty prediction. The results in Fig. 3 show the impact of our loss function in relation to the baseline. Figure 3 compares the predictive probability of the correctness of DINO (baseline) and our model on

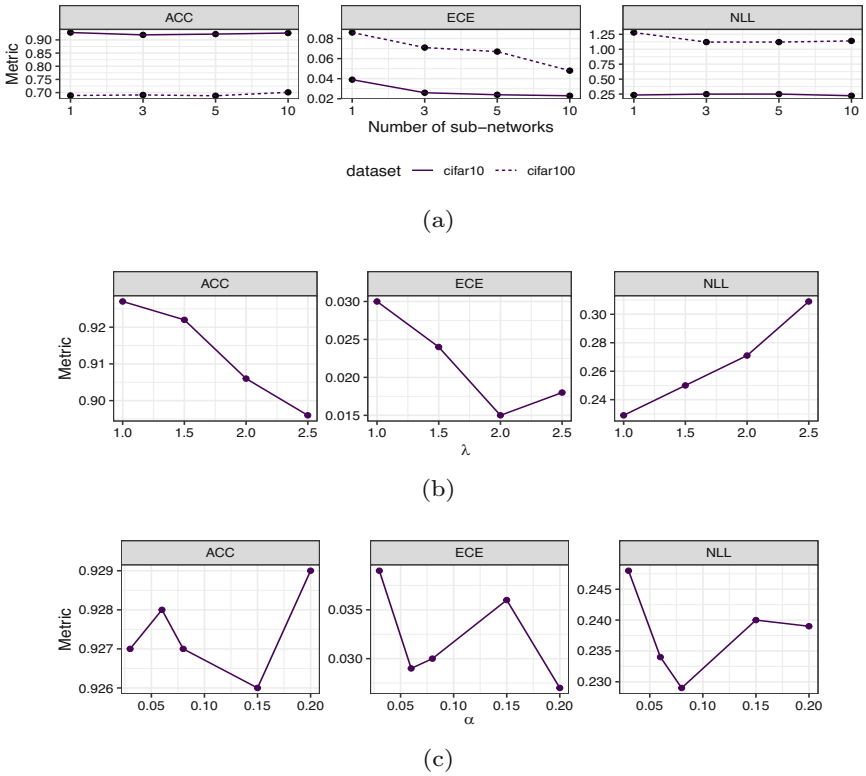


Fig. 7. Ablation study on number of M sub-networks (a), hyperparameters of our proposed loss (b) λ and (c) α.

ImageNet. Second, we explore different hyperparameter configurations to find the optimal values for α and λ in Fig. 7b, 7c. Note that, in practice, α and λ must be optimized jointly. The best top-1 accuracy in our case is achieved when α and λ are set to 0.08 and 1.5, respectively, on the CIFAR-10 dataset.

Analysis of Diversity. In addition to quantitative results for diversity analysis provided in Fig. 2b, we visualize the activation map for the last convolution layer in the encoder for each ensemble member and each subnetwork to motivate the effect of subnetworks on the encoder. As illustrated in Fig. 6, different subnetworks have more feature diversity compared to the deep ensemble as we expected.

Efficient Ensemble of Sub-networks at Pretraining vs. Finetuning We performed additional experiments to study the efficiency of proposed loss and independent sub-networks (InSub) i) during pretraining, ii) during finetuning, and iii) during both pretraining and finetuning. As shown in Table 5, pretraining with an ensemble of sub-networks is beneficial, and additional fine-tuning with multiple heads can further improve performance.

Table 5. Pretraining vs. Finetuning: Expected calibration error averaged over uncertainty-aware evaluation on CIFAR-10 datasets. InSub refers to training with our proposed Independent Subnetworks

Method	ACC (%) (↑)	ECE (↓)	NLL (↓)	TACE (↓)
Baseline	92.5	0.039	0.238	0.133
Pretrain-InSub	92.6	0.032	0.226	0.131
Finetune-InSub	92.6	0.021	0.222	0.103
Pretrain-InSub + Finetune-InSub	92.8	0.023	0.227	0.115

Table 6. Sub-Network Size: Expected calibration error averaged over uncertainty-aware evaluation on CIFAR-10 datasets.

Method	ACC (%) (↑)	ECE (↓)	NLL (↓)
Our method with 5 sub-network (100%)	92.9	0.019	0.221
With 25 percent of sub-network size	92.3	0.026	0.231
With 50 percent of sub-network size	92.6	0.021	0.226
With 75 percent of sub-network size	92.6	0.019	0.221

Table 7. Large variant encoder: Expected calibration error averaged over uncertainty-aware evaluation on CIFAR-10 datasets.

Method	ACC (%) (↑)	ECE (↓)	NLL (↓)	Number of parameters (M)
Our method with ResNet50 as a encoder with 5 sub-networks	92.9	0.019	0.221	45.79
Baseline with ResNet101 as a encoder	93.2	0.027	0.202	46.95

Analysis of Size of Sub-networks. We perform several experiments to study the different sizes of sub-network. As shown in Table 6, the dimension of projection heads does not change the top-1 accuracy. Recent self-supervised models such as SimCLR [5], BarlowTwins [48] also reach the same results with different projection head sizes.

Impact of Model Parameters. Our project aims to improve the predictive uncertainty of the baseline without losing predictive performance by mimicking the ensembles of self-supervised models with much lower computational costs. According to the results shown in Table 7, a bigger encoder can potentially improve the predictive performance, but it does not necessarily improve the predictive uncertainty of the results. We used ResNet101 as a baseline with more parameters in the encoder. To have a fair comparison, we compare it with our model with five heads. Our model performs better in ECE and NLL and has comparable accuracy. Also, we used ResNet34 as a baseline with fewer parameters in the encoder with twenty heads and compared it with baseline ResNet50 with one head. According to results obtained in Table 8, our model performs better in terms of ECE and NLL and has on-par accuracy.

Table 8. Different encoder (medium size): Expected calibration error averaged over uncertainty-aware evaluation on CIFAR-10 datasets.

Method	ACC (%) (↑)	ECE (↓)	NLL (↓)	Number of parameters (M)
Our method with ResNet34 as a encoder with 20 sub-networks	92.5	0.016	0.23	27.84
Baseline with ResNet50 as a encoder	92.8	0.039	0.233	27.89

7 Conclusion

In this paper, we presented a novel diversified ensemble of self-supervised framework. We achieved high predictive performance and good calibration using a simple yet effective idea – an ensemble of independent sub-networks. We introduced a new loss function to encourage diversity among different sub-networks. Our method is able to produce well-calibrated estimates of model uncertainty at low computational overhead over a single model while performing on par with deep self-supervised ensembles. It is straightforward to add our method to many existing self-supervised learning frameworks during pretraining. Our extensive experimental results show that our proposed method outperforms, or is on par with, an ensemble of self-supervised baseline methods in many different experimental settings.

Acknowledgments. L.W. is supported by the DAAD program Konrad Zuse Schools of Excellence in Artificial Intelligence, sponsored by the German Federal Ministry of Education and Research.

References

1. Baevski, A., Hsu, W., Xu, Q., Babu, A., Gu, J., Auli, M.: data2vec: a general framework for self-supervised learning in speech, vision and language. In: ICML (2022)
2. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. NeurIPS **33**, 1877–1901 (2020)
3. Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P., Joulin, A.: Emerging properties in self-supervised vision transformers. In: Proceedings of the IEEE/CVF ICCV, pp. 9650–9660 (2021)
4. Chen, C., Sun, X., Hua, Y., Dong, J., Xv, H.: Learning deep relations to promote saliency detection. In: Proceedings of the AAAI, pp. 10510–10517 (2020)
5. Chen, T., Kornblith, S., Norouzi, M., Hinton, G.: A simple framework for contrastive learning of visual representations. In: ICML, pp. 1597–1607 (2020)
6. Conneau, A., Kiela, D.: Senteval: An evaluation toolkit for universal sentence representations. In: Calzolari, N., et al. (eds.) Proceedings of the Eleventh International Conference on Language Resources and Evaluation, LREC 2018, Miyazaki, Japan, May 7-12, 2018. European Language Resources Association (ELRA) (2018), http://www.lrec-conf.org/proceedings/lrec2018/summaries/757.html
7. Dabouei, A., Soleymani, S., Taherkhani, F., Dawson, J., Nasrabadi, N.M.: Exploiting joint robustness to adversarial perturbations. In: Proceedings of the IEEE/CVF CVPR, pp. 1122–1131 (2020)

8. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE CVPR, pp. 248–255. IEEE (2009)
9. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. ACL (2018)
10. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). pp. 4171–4186. Association for Computational Linguistics, Minneapolis, Minnesota, June 2019. https://doi.org/10.18653/v1/N19-1423, https://aclanthology.org/N19-1423
11. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N.: An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In: Proceedings of the 9th ICLR (2021)
12. Dosovitskiy, A., et al.: An image is worth 16x16 words: Transformers for image recognition at scale (2021)
13. Durasov, N., Bagautdinov, T., Baque, P., Fua, P.: Masksembles for uncertainty estimation. In: Proceedings of the IEEE/CVF CVPR, pp. 13539–13548 (2021)
14. Fort, S., Hu, H., Lakshminarayanan, B.: Deep ensembles: A loss landscape perspective. arXiv preprint arXiv:1912.02757 (2019)
15. Gal, Y., Ghahramani, Z.: Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In: ICML pp. 1050–1059. PMLR (2016)
16. Geng, C., Huang, S.j., Chen, S.: Recent advances in open set recognition: a survey. IEEE TPAMI **43**(10), 3614–3631 (2020)
17. Grill, J.B., Strub, F., Altché, F., Tallec, C., Richemond, P., Avila Pires, B., Guo, Z., Gheshlaghi Azar, M.: Bootstrap your own latent-a new approach to self-supervised learning. NeurIPS **33**, 21271–21284 (2020)
18. Gündüz, H.A., et al.: A self-supervised deep learning method for data-efficient training in genomics. Commun. Biol. **6**(1), 928 (2023)
19. Hansen, L.K., Salamon, P.: Neural network ensembles. IEEE Trans. Pattern Anal. Mach. Intell. **12**(10), 993–1001 (1990)
20. Havasi, M., et al.: Training independent subnetworks for robust prediction. In: ICLR (2021)
21. He, K., Chen, X., Xie, S., Li, Y., Dollár, P., Girshick, R.: Masked autoencoders are scalable vision learners. In: Proceedings of the IEEE/CVF CVPR, pp. 16000–16009 (2022)
22. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE CVPR, pp. 770–778 (2016)
23. Huggingface: wiki1m_for_simcse.txt (2021). https://huggingface.co/datasets/princeton-nlp/datasets-for-simcse/blob/main/wiki1m_for_simcse.txt
24. Ioffe, S.: Batch Renormalization: Towards Reducing Minibatch Dependence in Batch-Normalized Models. In: NeurIPS, 2017 (2017)
25. Jaiswal, A., Babu, A.R., Zadeh, M.Z., Banerjee, D., Makedon, F.: A Survey on Contrastive Self-Supervised Learning. Technologies **9**(1) (2020)
26. Klein, T., Nabi, M.: Scd: self-contrastive decorrelation for sentence embeddings. Proceedings of the 60th ACL (2022)
27. Krizhevsky, A.: Learning multiple layers of features from tiny images. University of Toronto, Tech. rep. (2009)
28. Lakshminarayanan, B., Pritzel, A., Blundell, C.: Simple and scalable predictive uncertainty estimation using deep ensembles. In: NeurIPS, vol. 30 (2017)

29. Li, J., Yao, Y., Xu, H.H., Hao, L., Deng, Z., Rajakumar, K., Ou, H.Y.: Secret6: a web-based resource for type vi secretion systems found in bacteria. Environ. Microbiol. **17**(7), 2196–2202 (2015)
30. Liang, S., Li, Y., Srikant, R.: Enhancing the reliability of out-of-distribution image detection in neural networks. ICLR (2018)
31. Naeini, M.P., Cooper, G.F., Hauskrecht, M.: Obtaining well calibrated probabilities using bayesian binning. In: Proceedings of AAAI'15. AAAI Press (2015)
32. Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.Y.: Reading digits in natural images with unsupervised feature learning (2011)
33. Nixon, J., et al.: Measuring Calibration in Deep Learning (2019)
34. Ortega, L.A., Cabañas, R., Masegosa, A.: Diversity and generalization in neural network ensembles. In: International Conference on Artificial Intelligence and Statistics, pp. 11720–11743. PMLR (2022)
35. Pang, T., Xu, K., Du, C., Chen, N., Zhu, J.: Improving adversarial robustness via promoting ensemble diversity. In: ICML, pp. 4970–4979. PMLR (2019)
36. Radford, A., et al.: Learning transferable visual models from natural language supervision. In: ICML, pp. 8748–8763. PMLR (2021)
37. Ramé, A., Cord, M.: DICE: diversity in deep ensembles via conditional redundancy adversarial estimation. In: 9th ICLR 2021, Virtual Event, Austria, 3–7 May, 2021. OpenReview.net (2021)
38. Rezaei, M., Näppi, J., Bischl, B., Yoshida, H.: Deep mutual gans: representation learning from multiple experts. In: Medical Imaging 2022: Imaging Informatics for Healthcare, Research, and Applications, vol. 12037, pp. 191–197. SPIE (2022)
39. Rezaei, M., Näppi, J.J., Bischl, B., Yoshida, H.: Bayesian uncertainty estimation for detection of long-tail and unseen conditions in abdominal images. In: Medical Imaging 2022: Computer-Aided Diagnosis, vol. 12033, pp. 270–276. SPIE (2022)
40. Rezaei, M., Soleymani, F., Bischl, B., Azizi, S.: Deep bregman divergence for self-supervised representations learning. Computer Vision and Image Understanding, p. 103801 (2023)
41. Selvaraju, R.R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., Batra, D.: Grad-cam: visual explanations from deep networks via gradient-based localization. In: ICCV (2017)
42. Sinha, S., Bharadhwaj, H., Goyal, A., Larochelle, H., Garg, A., Shkurti, F.: Dibs: diversity inducing information bottleneck in model ensembles. In: Proceedings of the AAAI, pp. 9666–9674 (2021)
43. Tian, Y., Chen, X., Ganguli, S.: Understanding self-supervised learning dynamics without contrastive pairs. In: ICML (2021)
44. Tran, D., et al.: Plex: Towards reliability using pretrained large model extensions. arXiv preprint arXiv:2207.07411 (2022)
45. Vahidi, A., Schosser, S., Wimmer, L., Li, Y., Bischl, B., Hüllermeier, E., Rezaei, M.: Probabilistic self-supervised representation learning via scoring rules minimization. In: The Twelfth International Conference on Learning Representations (2023)
46. Wen, Y., Tran, D., Ba, J.: Batchensemble: an alternative approach to efficient ensemble and lifelong learning. In: ICLR (2020)
47. Wen, Z., Li, Y.: The mechanism of prediction head in non-contrastive self-supervised learning. NeurIPS (2022)
48. Zbontar, J., Jing, L., Misra, I., LeCun, Y., Deny, S.: Barlow twins: self-supervised learning via redundancy reduction. In: ICML, pp. 12310–12320. PMLR (2021)
49. Zhang, C., Ma, Y.: Ensemble Machine Learning: Methods and Applications. Springer Publishing Company, Incorporated (2012)

8 Implementation Details

8.1 Computation Cost Analysis

Figure 8 illustrates relative computation cost – as compared to the baseline – in terms of the number of parameters, computation time, and memory required between our model and a deep self-supervised ensemble.

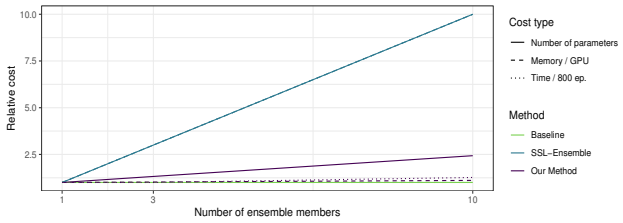


Fig. 8: The test time cost (purple, dotted) and memory cost (purple, dashed) of our model w.r.t. the ensemble size. The figures are relative to the cost incurred by a single model (green). The inference time cost and memory cost of a deep self-supervised ensemble are plotted in blue.

8.2 Computational Cost Analysis

As we mentioned in Section Method the increase in the number of parameters is 32% and 143%, and the increase in computational requirement is negligible and 6% for our method compared to the baseline when there exist 3 and 10 ensemble members, respectively. We would like to explain the reason as follows:

While the encoder networks used in the baseline methods (and our method) contain many convolutional layers, the additional parameters introduced by our method are in the projection head, and they are a few linear layers. Although these additional linear layers increase the number of parameters to some extent, the computational burden introduced by them is much more limited compared to the convolutional layers that exist in both baselines and our method. That is because convolutional layers typically contain fewer parameters compared to fully connected layers due to parameter-sharing but have a much higher computational burden since their outputs are evaluated over the whole image. A toy example to understand this would be the comparison of the two settings below:

1. Consider a 100×100 image fed into a convolutional filter with (kernel size= 3×3, stride= 1×1, padding= "same", bias= False). The convolutional filter has 9 parameters but needs to do $100 \times 100 \times 9 = 90000$ multiplications to evaluate its output.

2. Consider a vector of 1000 that is fed into a fully connected layer (without bias) to produce 1 output value. The fully connected layer contains 1000 parameters and the number of multiplications needed to evaluate its output is also 1000.

Comparing these two settings, the convolutional layer needs ∼100 times more computational burden (convolutional: 90000 vs dense: 1000 multiplications) to evaluate its outcome, although it has ∼100 times fewer parameters (convolutional: 9 vs dense: 1000) compared to the fully connected layer.

Similarly, the increase in memory requirements is low for our method compared to the SSL-Ensemble, but the increase in computational requirements is much lower and even negligable.

8.3 Data Augmentation for Computer Vision Datasets

We define a random transformation function $\boldsymbol{T}$ that applies a combination of crop, horizontal flip, color jitter, and grayscale. Similar to [5], we perform crops with a random size from 0.2 to 1.0 of the original area and a random aspect ratio from 3/4 to 4/3 of the original aspect ratio. We also apply horizontal mirroring with a probability of 0.5. Then, we apply grayscale with a probability of 0.2 as well as color jittering with a probability of 0.8 and a configuration of $(0.4, 0.4, 0.4, 0.1)$. However, for ImageNet, we define augmentation based on the original DINO from their official repository. In all experiments, at the testing phase, we apply only resize and center crop.

8.4 Hyperparameters for Self-supervised Network Architectures

SimCLR [5]: we use ResNet-50 as a backbone, a loss temperature of 0.07, batch size 512, and a cosine-annealing learning rate scheduler. The embedding size is 2048, and we train for 800 epochs during pretraining. **DINO** [3]: we use ViT-small as a backbone, patch size 16, batch size 1024, and a cosine-annealing learning rate scheduler. The embedding size is 384/1536, and we train for 100 epochs during pretraining.

9 Additional Results

9.1 Robustness of representation: IND- Generalization

Tables 9, 10, and 11 present results for the predictive performance and calibration of our model on CIFAR-10, CIFAR-100, and ImageNet respectively. Based on Table 9, our method achieves better calibration than the deep ensemble of self-supervised networks, MC-Dropout, and BatchEnsemble, with significant margins at large ensemble sizes. In order to have multiple batches for

BatchEnsemble, we decreased the initial batch size because of memory, so we ended up with a smaller batch size to which the self-supervised model (i.e., SimCLR) is sensitive. Also, each time we have more positive samples than the original.

In the case of dropouts, we again face the same problem with positive and negative samples. Dropouts also count as data regularization, but when applied randomly to all data in contrastive learning, it degrades the idea of positive and negative. For example, in NLP, dropouts are used to produce different augmentations.

Table 9: **IND Generalization**: Top-1 accuracy, ECE and NLL averaged over in-distribution on test samples of the **CIFAR-10** dataset over three random seeds. The best score for each metric is shown in **bold**, and the second-best is underlined.

Method	Top-1 Acc (%) (↑)			ECE (↓)			NLL (↓)		
# member (M)	3	5	10	3	5	10	3	5	10
Baseline	92.8 ± 0.4			0.039 ± 0.002			0.233 ± 0.011		
SSL-Ensemble	92.8 ± 0.1	93.0 ± 0.2	$\mathbf{94.2 \pm 0.3}$	0.043 ± 0.02	0.033 ± 0.01	$\underline{0.029 \pm 0.02}$	0.221 ± 0.011	0.226 ± 0.009	$\underline{0.199 \pm 0.004}$
MC Dropout	65.7 ± 0.2	66.3 ± 0.2	66.4 ± 0.2	0.083 ± 0.014	0.077 ± 0.009	0.075 ± 0.005	0.66 ± 0.012	0.637 ± 0.002	0.593 ± 0.006
BatchEnsemble	69.1 ± 04	72.1 ± 0.3	71.9 ± 0.2	0.064 ± 0.011	0.061 ± 0.008	0.062 ± 0.005	$0.613 \pm xx$	0.58 ± 0.007	0.551 ± 0.004
Our method	92.6 ± 0.2	92.9 ± 0.1	$\underline{93.6 \pm 0.1}$	0.021 ± 0.004	0.019 ± 0.002	$\mathbf{0.016 \pm 0.001}$	0.241 ± 0.010	0.221 ± 0.005	$\mathbf{0.193 \pm 0.003}$

Table 10: **IND Generalization**: Top-1 accuracy, ECE and NLL averaged over in-distribution on test samples of the **CIFAR-100** dataset over three random seeds. The best score for each metric is shown in **bold**, and the second-best is underlined.

Method	Top-1 Acc (%) (↑)			ECE (↓)			NLL (↓)		
# member (M)	3	5	10	3	5	10	3	5	10
Baseline	68.9 ± 0.3			0.086 ± 0.014			1.28 ± 0.05		
SSL-Ensemble	70.6 ± 0.12	71.4 ± 0.5	**72.0 ± 0.2**	0.12 ± 0.01	0.122 ± 0.01	0.119 ± 0.04	1.09 ± 0.01	1.12 ± 0.01	$\underline{1.06 \pm 0.02}$
MC Dropout	46.3 ± 0.1	45.2 ± 0.4	48.2 ± 0.1	0.077 ± 0.012	0.081 ± 0.002	0.071 ± 0.002	2.66 ± 0.11	2.37 ± 0.02	2.43 ± 0.06
BatchEnsemble	44.1 ± 03	45.2 ± 0.2	46.1 ± 0.1	0.073 ± 0.01	0.071 ± 0.08	$\underline{0.068 \pm 0.001}$	2.43 ± 0.03	2.64 ± 0.007	2.51 ± 0.004
Our method	67.7 ± 0.1	68.8 ± 0.1	$\underline{70.1 \pm 0.0}$	0.067 ± 0.001	0.063 ± 0.001	**0.048 ± 0.000**	0.114 ± 0.005	0.116 ± 0.0002	**1.06 ± 0.001**

Table 11: **IND Generalization**: Top-1 accuracy, ECE and NLL averaged over in-distribution on test samples of the **ImageNet** dataset over three random seeds. The best score for each metric is shown in **bold**, and the second-best is underlined.

Method	Top-1 Acc (%) (↑)	ECE (↓)	NLL (↓)
Baseline	73.8 ± 0.3	0.013 ± 0.015	1.05 ± 0.01
SSL-Ensemble	**75.1 ± 0.1**	0.014 ± 0.000	**0.98 ± 0.01**
Our method	74.0 ± 0.0	**0.010 ± 0.000**	1.03 ± 0.01

Table 12: **IND Generalization**: Top-1 accuracy, ECE and NLL averaged over in-distribution on test samples of the **T6SS Identification** dataset over three random seeds. The best score for each metric is shown in **bold**, and the second-best is underlined.

Method	Top-1 Acc (%) (↑)	ECE (↓)	NLL (↓)
Baseline	75.9 ± 2.0	0.100 ± 0.006	0.502 ± 0.020
SSL-Ensemble	**80.2 ± 0.7**	**0.099 ± 0.014**	**0.471 ± 0.011**
Our method	76.7 ± 2.3	0.108 ± 0.006	0.492 ± 0.024

We also performed experiments on a dataset of 1-dimensional genomic sequences – the T6SS identification of effector proteins– to demonstrate that uncertainty-aware subnetworks can also be readily combined with existing models for 1-dimensional datasets and models. Based on Table 12, our method improves the accuracy and the calibration compared to the baseline.

10 Additional Ablation Analysis

Efficient ensemble of sub-networks at pretraining vs. finetuning We performed additional experiments to study the efficiency of proposed loss and independent sub-networks (InSub) i) during pretraining, ii) during finetuning, and iii) during both pretraining and finetuning. As shown in Table 13, pretraining with an ensemble of sub-networks is beneficial, and additional fine-tuning with multiple heads can further improve performance.

10.1 Analysis of Size of Sub-Networks

We perform several experiments to study the different sizes of sub-network. As shown in Table 14, the dimension of projection heads does not change the top-1 accuracy. Recent self-supervised models such as SimCLR [5], BarlowTwins [48] also reach the same results with different projection head sizes.

Table 13: **Pretraining vs. Finetuning**: Expected calibration error averaged over uncertainty-aware evaluation on CIFAR-10 datasets. InSub refers to training with our proposed Independent Subnetworks

Method	ACC (%) (↑)	ECE (↓)	NLL (↓)	TACE (↓)
Baseline	92.5	0.039	0.238	0.133
Pretrain-InSub	92.6	0.032	0.226	0.131
Finetune-InSub	92.6	0.021	0.222	0.103
Pretrain-InSub + Finetune-InSub	92.8	0.023	0.227	0.115

Table 14: **Sub-Network Size**: Expected calibration error averaged over uncertainty-aware evaluation on CIFAR-10 datasets.

Method	ACC (%) (↑)	ECE (↓)	NLL (↓)
Our method with 5 sub-network (100%)	92.9	0.019	0.221
With 25 percent of sub-network size	92.3	0.026	0.231
With 50 percent of sub-network size	92.6	0.021	0.226
With 75 percent of sub-network size	92.6	0.019	0.221

Table 15: **Large variant encoder**: Expected calibration error averaged over uncertainty-aware evaluation on CIFAR-10 datasets.

Method	ACC (%) (↑)	ECE (↓)	NLL (↓)	Number of parameters (M)
Our method with ResNet50 as a encoder with 5 sub-networks	92.9	0.019	0.221	45.79
Baseline with ResNet101 as a encoder	93.2	0.027	0.202	46.95

Table 16: **Different encoder (medium size)**: Expected calibration error averaged over uncertainty-aware evaluation on CIFAR-10 datasets.

Method	ACC (%) (↑)	ECE (↓)	NLL (↓)	Number of parameters (M)
Our method with ResNet34 as a encoder with 20 sub-networks	92.5	0.016	0.23	27.84
Baseline with ResNet50 as a encoder	92.8	0.039	0.233	27.89

10.2 Impact of Model Parameters

Our project aims to improve the predictive uncertainty of the baseline without losing predictive performance by mimicking the ensembles of self-supervised models with much lower computational costs. According to the results shown in Table 15, a bigger encoder can potentially improve the predictive performance, but it does not necessarily improve the predictive uncertainty of the results. We used ResNet101 as a baseline with more parameters in the encoder. To have a fair comparison, we compare it with our model with five heads. Our model performs better in ECE and NLL and has comparable accuracy.

Also, we used ResNet34 as a baseline with fewer parameters in the encoder with twenty heads and compared it with baseline ResNet50 with one head. According to results obtained in Table 16, our model performs better in terms of ECE and NLL and has on-par accuracy.

10.3 Source Code

Please find the source code in the supplementary material.

11 Theoretical Supplement

11.1 Proof for Eq. 6

$$
\begin{aligned}
\frac{\partial\left(\ell_{div}\right)}{\partial \boldsymbol{z}_{k,\hat{m},o}} &= \underbrace{\frac{-1}{2}\left(\frac{1}{M-1}\sum_{m=1}^{M}(\boldsymbol{z}_{k,m,o}-\bar{\boldsymbol{z}}_{k,o})^2\right)}_{A}\cdot\frac{\partial\left(\frac{1}{M-1}\sum_{m=1}^{M}(\boldsymbol{z}_{k,m,o}-\bar{\boldsymbol{z}}_{k,o})^2\right)}{\partial \boldsymbol{z}_{k,\hat{m},o}} = \\
\frac{-A}{2}\cdot\frac{1}{M-1}\left[2\cdot\left[(\boldsymbol{z}_{k,\hat{m},o}-\bar{\boldsymbol{z}}_{k,o})\cdot\left(\frac{\partial\ \boldsymbol{z}_{k,\hat{m},o}}{\partial \boldsymbol{z}_{k,\hat{m},o}}-\frac{\partial\bar{\boldsymbol{z}}_{k,o}}{\partial \boldsymbol{z}_{k,\hat{m},o}}\right)+\sum_{m=1}^{M}\mathbb{I}_{[m\neq\hat{m}]}(\boldsymbol{z}_{k,m,o}-\bar{\boldsymbol{z}}_{k,o})\cdot\left(\frac{\partial \boldsymbol{z}_{k,m,o}}{\partial \boldsymbol{z}_{k,\hat{m},o}}-\frac{\partial\bar{\boldsymbol{z}}_{k,o}}{\partial \boldsymbol{z}_{k,\hat{m},o}}\right)\right]\right] = \\
\frac{-A}{2}\cdot\frac{1}{M-1}\left[2\cdot\left[(\boldsymbol{z}_{k,\hat{m},o}-\bar{\boldsymbol{z}}_{k,o})\cdot\left(1-\frac{1}{M}\right)+\sum_{m=1}^{M}\mathbb{I}_{[m\neq\hat{m}]}(\boldsymbol{z}_{k,m,o}-\bar{\boldsymbol{z}}_{k,o})\cdot\left(\frac{-1}{M}\right)\right]\right] = \\
\frac{-A}{M-1}\cdot\left[(\boldsymbol{z}_{k,\hat{m},o}-\bar{\boldsymbol{z}}_{k,o})\cdot\frac{M-1}{M}+\left(\left(\frac{\boldsymbol{z}_{k,\hat{m},o}}{M}\right)-\bar{\boldsymbol{z}}_{k,o}+\frac{M-1}{M}\cdot\bar{\boldsymbol{z}}_{k,o}\right)\right] = \\
\frac{-A}{M-1}\cdot(\boldsymbol{z}_{k,\hat{m},o}-\bar{\boldsymbol{z}}_{k,o})
\end{aligned}
\tag{8}
$$

11.2 Proof for Eq. 7

$$
\eta\cdot\nabla_{w_{\hat{m},o}}\ell_{div}=\eta\cdot\frac{\partial\ell_{div}}{\partial \boldsymbol{z}_{k,\hat{m},o}}\cdot\frac{\partial \boldsymbol{z}_{k,\hat{m},o}}{\partial w_{\hat{m},o}}=\eta\cdot\frac{-A}{M-1}\cdot(\boldsymbol{z}_{k,\hat{m},o}-\bar{\boldsymbol{z}}_{k,o})\cdot b
$$

7 Contributions to Software Development for Genomics

7.1 GenomeNet: A Platform for Deep Learning on (Meta)genomic Sequences

Contributing Article

Mreches R, To XY, Gündüz HA, Moosbauer J, Klawitter S, Deng ZL, Robertson G, Rezaei M, Asgari E, Franzosa EA, Huttenhower C, Bischl B, McHardy AC, Binder M, Münch PC (2024). "GenomeNet: A platform for deep learning on (meta)genomic sequences."

Declaration of Contributions

Hüseyin Anil Gündüz contributed to this paper as a co-author with the following significant contributions:
Hüseyin Anil Gündüz made a substantial contribution regarding code development to the deepG software library, a crucial aspect of this paper. Hüseyin Anil Gündüz also contributed to the inclusion and implementation of major parts of his other works to the deepG library, such as Self-GenomeNet and GenomeNet-Architect, which are methods focusing on self-supervised learning and architecture optimization for sequential genomics data.

Contribution of the coauthors:
Rene Mreches and Xiao-Yin To share the first authorship of this paper with overall equal contributions. Rene Mreches developed most of the deepG library and developed the GenomeNet structure. Xiao-Yin To developed the functions for training a self-supervised deep learning model as well as for siamese/twin neural networks. She further preprocessed the data, implemented the code, and ran the models for the BacDive database predictions. Philipp C. Münch and Martin Binder share senior authorship of the paper as the main supervisors of the project.

All authors contributed to the editing and proofreading of the paper.

GenomeNet: A platform for deep learning on (meta)genomic sequences

René Mreches[1,2,#], Xiao-Yin To[1,2,3,4,#], Hüseyin Anil Gündüz[3,4], Julia Moosbauer[3,4], Sandra Klawitter[1,2], Zhi-Luo Deng[1,2], Gary Robertson[1,2], Mina Rezaei[3,4], Ehsaneddin Asgari[1,2], Eric A. Franzosa[5], Curtis Huttenhower[5], Bernd Bischl[3,4], Alice C. McHardy[1,2,6,7], Martin Binder[3,4,*] and Philipp C. Münch[1,2,5,6,7,*]

[1] Department for Computational Biology of Infection Research, Helmholtz Center for Infection Research, 38124 Braunschweig, Germany
[2] Braunschweig Integrated Centre of Systems Biology (BRICS),Technische Universität Braunschweig, Braunschweig, Germany
[3] Department of Statistics, LMU Munich, Germany
[4] Munich Center for Machine Learning, Munich, Germany
[5] Department of Biostatistics, Harvard School of Public Health, Boston, Massachusetts, USA
[6] Cluster of Excellence RESIST (EXC 2155), Hannover Medical School, 30625 Hannover, Germany
[7] German Centre for Infection Research (DZIF)
* These authors share senior authorship
# These authors share first-authorship

Abstract

We introduce GenomeNet, an online platform designed to accelerate genomics research through the creation, use and sharing of deep learning models for bioinformatics tasks. Each model is accompanied by comprehensive model and dataset cards that detail training procedure, performance metrics, and potential applications. To complement this platform, we created *deepG*, a software library focused on the creation of deep learning models optimized for nucleotide sequence data, covering tasks such as labeling, classification, and regression. Unlike other deep learning software libraries that are primarily tailored for human genomics, deepG employs a variety of data augmentation techniques to facilitate the training of robust and accurate models. These models are capable of handling a diverse range of data types, from short reads and contigs to complete genomes and metagenomic assemblies. Overall, this platform aims to advance genomics research by making model development, deployment, and reproducibility more accessible, while fostering community-wide collaboration.

Introduction

Bioinformatics has long depended on specialized tools for processing and interpreting various types of biological data. One of the most intricate and information-rich types of data comes in the form of genomic sequences. These could range from full genomes to smaller sequence fragments such as contigs and reads, sourced from an array of organisms including humans, animals, plants, bacteria, and viruses. In the traditional bioinformatics pipeline, one often has to navigate a series of complex preprocessing steps before arriving at analyzable data. For example, in dealing with genomic sequences, tasks could include but are not limited to the identification and annotation of Open Reading Frames (ORFs), gene prediction, sequence alignment, and comparison with reference genomes. Additional steps may involve variant calling

to detect mutations or Single Nucleotide Polymorphisms (SNPs), and even comparative genomics to relate sequences across species.

These multi-step processes necessitate a vast landscape of software tools, each designed with specific input and output formats, as well as targeted functionalities. This poses a practical challenge for bioinformaticians and researchers, as they must carefully curate datasets and align tools to ensure compatibility and meaningful results. The situation becomes even more complex when considering that different tools often utilize unique algorithms, requiring users to understand the underlying methodologies to prevent errors or misinterpretation of data. Moreover, these preprocessing methods may inherently reduce the available information content of the raw sequences. For instance, one common technique involves converting longer genomic sequences into k-mer profiles—short, overlapping substrings—to simplify data and speed up computation. While this can be effective, it often omits valuable information. A case in point is the potential loss of structural relationships between distant genomic features, which might be essential for understanding phenomena like chromatin interactions, long-range regulatory effects, or three-dimensional protein configurations.

In recent years, deep learning (DL) has emerged as a transformative approach in the field of bioinformatics, providing the foundation for new toolsets designed to tackle complex genomic problems. Libraries such as Selene (Chen et al. 2019) and Janggu (Kopp et al. 2020) exemplify this trend, albeit with a primary focus on human genomics. These libraries have been particularly instrumental in predicting transcriptional regulation events, such as transcription factor binding sites. However, their applicability can be limited when it comes to other types of genomes, especially those of microbial origin. This is due in part to differing data distributions among genomes of various species, as well as the specific data augmentation and subsampling methods employed by these tools, which may not be universally suitable for all genomics applications.

In response to these challenges, we present *deepG*, a versatile deep learning platform tailored for genomics research across a spectrum of organisms. Unlike existing tools, deepG is designed with a focus on adaptability and robustness to different data distributions, thereby enabling the platform to be used for a variety of open problems in genomics. This includes but is not limited to the taxonomic classification of bacteria and viruses, functional genomics, and even phenotypic characterization. To account for the inherent diversity and complexity of genomic data, deepG incorporates novel data augmentation and subsampling techniques that are sensitive to the specificities of different organisms and genomic features.

In response to the growing complexity and diversity of deep learning models in bioinformatics, we propose the standardized use of "model cards," brief documents that specify a model's capabilities, limitations, and optimal application scenarios. Similarly, we introduce dataset cards to detail the characteristics and appropriate usage of the training data. Our online platform, genomenet.de, serves as a repository for both models and their corresponding cards, aiming to simplify and demystify the user experience. Additionally, genomenet.de offers an inference service, allowing researchers to upload FASTA files for immediate analysis. By combining

accessible model documentation with practical utility, we aim to promote transparent and effective use of machine learning in the field of genomics.

Results

From library to application: The GenomeNet ecosystem

Central to our ecosystem is *deepG*, an R-based software library developed for training DL models on (meta) genomical datasets. The library is built with a focus on flexibility, allowing researchers to easily adapt its features to different types of genomic data across various organisms. It incorporates data augmentation and subsampling techniques tailored for genomics, aimed at making the models robust to different data distributions (**Fig. 1A**). Additionally, *deepG* streamlines the development, training, and application of deep learning models, significantly reducing the computational overhead traditionally associated with genomics research.

Complementing *deepG*, we introduce *GenomeNet*, a platform featuring a curated repository of pre-trained DL models compatible with deepG library. For this, we introduce two foundational elements aimed at enhancing transparency, reproducibility, and utility in genomics research: "Model Cards" and "Dataset Cards." Model Cards furnish comprehensive details about each model, from architecture and performance metrics to the rationale behind the choice of those metrics (**Fig. 1B,** Supplementary Table 1). Dataset Cards, on the other hand, offer an overview of the training data used for each DL model. These cards encapsulate vital information such as the data source, preprocessing steps, class distributions, and any limitations like taxonomic biases that may affect model generalizability (**Fig. 1B,** Supplementary Table 2). For those interested in contributing to the platform, we offer options for community-submitted models, which undergo rigorous validation before being added to the platform, further enriching its capabilities.

On the end-user front, GenomeNet serves as a powerful resource, permitting researchers to search the repository for models matching their specific needs. Following model selection, users can upload their dataset and employ our online inference service for generating predictions (**Fig. 1C**). The detailed information available in Model and Dataset Cards aids in this model selection process, ensuring that end-users can make informed, responsible choices in their research endeavors. Further, a feature of the platform is its queuing system. Given that genomic inference tasks can be computationally intensive and vary in execution time, the queuing system ensures equitable distribution of computational resources. This is especially crucial during periods of high traffic, facilitating a smooth and uninterrupted user experience. Crucially, all the models discussed in this manuscript, including those predicting sporulation capability, identifying CRISPR cassettes, and imputing missing nucleotides, are integrated into GenomeNet's repository and are accompanied by their respective Model and Dataset Cards. Alongside the interface, we offer a RESTful API that allows seamless integration into existing bioinformatics workflows or platforms, enhancing its applicability across various research settings.

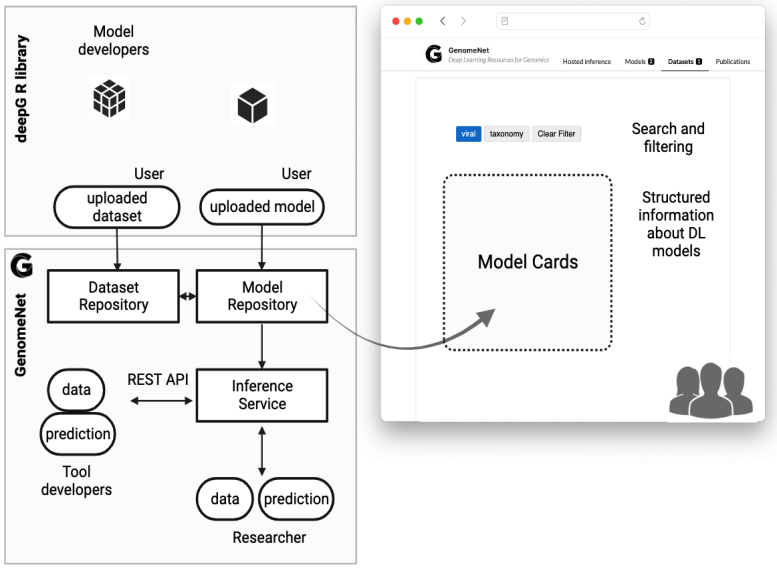


Figure 1: Integrative Workflow of the GenomeNet Ecosystem A) Illustration of deepG's role in processing genomic datasets to train deep learning models. Metadata such as performance metrics and dataset characteristics are generated automatically at this stage. B) Depiction of GenomeNet's platform features, including the uploading process with integrated Model and Dataset Cards, which serve as comprehensive metadata descriptors for each uploaded DL model. C) Demonstration of the end-user experience, emphasizing the repository search functionality for model selection and the online inference service for real-time predictive analytics. Arrows guide the directional flow, elucidating the synergistic interplay among deepG, GenomeNet, and the end-users, thereby providing a coherent understanding of the integrated GenomeNet ecosystem.

The DeepG library

Deep learning (DL) models in computational biology have been applied to an increasing number of challenges[1], such as virus detection[2], antibiotic resistance prediction[3], and contamination removal[4]. The development of such models often uses DL application programming interfaces (API) such as Keras, which enables researchers to stack neural layers into deep neural networks. However, these interfaces do not provide solutions for efficient data handling and network training for genomic data modalities.

With *deepG*, we provide a software library that includes adaptations for genomic datasets on the nucleotide and amino acid level and provides an easy-to-use interface for training and applying DL networks. In *deepG*, the standard workflow starts from collections of FASTA files divided into two or more sets that correspond to class labels. The *deepG* data generator iterates over the input files and trains a deep neural network that can be applied to new datasets to perform predictions (**Fig. 1A, S1**).
A common challenge of applying DL to genomics is that the input length of genomic sequences is typically longer compared to datasets of the traditional DL domains, especially when input samples are full (meta)genomes. To address this, the *deepG* library also includes specialized

neural networks to account for long-range dependencies that span multiple batches of samples (stateful long short-term memory) as well as architectures designed for long input sequences such as WaveNet[5], which implements residual and parameterized skip connections with dilated convolutions to speed up convergence in this data regime. Further, we have found that processing nucleotides sequences in a naïve and purely sequential fashion often leads to highly ineffective training, since this approach successively processes potentially varying nucleotide distributions of input samples, leading to repeated regimes of under- and overfitting (**Fig. 1B**). We identified and implemented combinations of subsampling strategies to mitigate these problems (**Fig. 1A**), which makes model training easy to use for a wide range of research questions. Furthermore, the library supports more customized and advanced training methods, such as the training of language models and fine-tuning (**Fig. 1C**) (see **Supplementary Methods**).

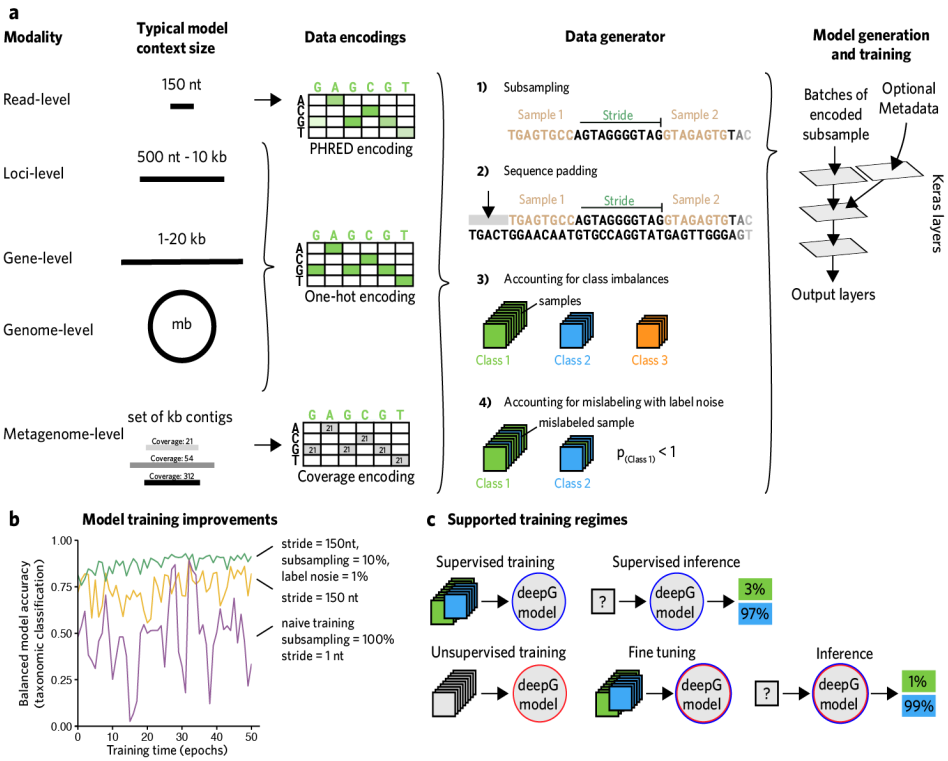


Figure 1: *deepG* facilitates the development of deep neural networks for a wide range of genomic data modalities. a) *deepG* is designed to work with different data types, from read-level data to metagenomic level. To process the nucleotide data, *deepG* generates a data encoding dependent on the input format (i.e., probability encoding in case of quality scores are provided within a FASTQ file, or coverage encoding if community coverage is provided for contigs in case of metagenomic data input, see **Supplementary Methods**). Next, the data generator within *deepG* runs several subsampling methods to prevent overfitting, supporting different stride sizes and padding regimes. *deepG* can adjust class weights and handle label noise caused by mislabeled samples to address class imbalances. **b)** Since the inputs of *deepG* are often sequences from different organisms, subsampling and data augmentation regimes are important to control for differences in the underlying data distributions. **c)** Besides the

supervised training, where one model is directly trained from sets of labeled samples, *deepG* supports unsupervised training, where a data representation can be trained from unlabeled samples.

Case Studies

We validated our method by developing classifiers targeting common bioinformatic tasks, and show that our approach achieves accuracy comparable to, or even superior to, highly specialized state of the art tools. To demonstrate the wide range of possible applications, we trained supervised models at the read-level, locus-level (within non-coding regions), gene-level, genome-level, and metagenomic-level (**Fig. 2a**). We provide code notebooks that fully reproduce these use-cases at http://deepg.de.

At the read-length level, we trained a model that is able to discriminate between bacterial and human sequences, which can be used for screening human contamination in metagenomic sequences. After 3 hours of training with a context size of 150 nucleotides (nt) (corresponding to the typical read-level size), our model achieved a balanced accuracy of 97% when trained on a set of bacterial genomes and a human reference genome. We evaluated this model on a paired end metagenomic dataset with a processing speed of over 250,000 reads per minute on a consumer-grade graphics processing unit (GPU), demonstrating a practical processing speed. In this case, the model was trained on data from the full human genome (non-read size) but optimized for the application on read data. *deepG* also supports the model development on read-level data (FASTQ files) and can account for this in the encoding of the network by using a probability encoding instead of one-hot encoding – which is of potential interest for the direct processing of long and low-quality sequencing reads (**Fig. S2**). This use case demonstrates that *deepG* can successfully run on small sequences such as 150 nt, which enables direct alignment-free applications. While the model was trained here on fragments larger than sequencing reads, the context size of the model was set to 150 nt to allow for the inference process of FASTQ files. However, models might be trained directly from FASTQ files holding the read information. This use case showcases the ability of *deepG* to efficiently process small sequence data and apply it to real-world metagenomic screening scenarios. When comparing the performance of model-based contamination detection on a synthetic evaluation dataset, where 75% of the reads originate from *E. coli* and the remaining 25% of the reads originate from a human genome, the *deepG* model shows similar accuracy compared to the alignment-based read removal tools BMTagger[6] with 97.58% vs. 97.98% accuracy (non-significant differences on ten datasets according to a paired Wilcoxon test). However, compared to alignment-based methods, the *deepG* model generalizes better to other eukaryotes: When replacing the human contaminant with mouse reads, our methods successfully predict 98.13% of the reads correctly, compared to 75.25% of BMTagger (**Fig. 2a**).

At the locus level, we trained a model that is capable of predicting CRISPR arrays, which are variable-length features present in non-coding parts of sequences. The CRISPR array identification problem is of interest, since it cannot be captured with models leveraging the profile of a multiple sequence alignment using profile Hidden Markov Models (pHMM)[7,8], and there are no clear conserved motifs, but there are conserved higher-order structures[9]. The *deepG* model outperforms strategies that are purely based on the identification of local

alignment on regions that are classified as false positive, such as *Staphylococcus aureus* repeat like elements[10], with an accuracy of 95% on these CRISPR-like sequences and reaches an area under the receiver operating characteristic curve (AUC ROC) of 0.98 (**Fig. 2b,c**). This shows that *deepG* models have higher representational power by taking semantics, syntactic, and synteny of genomic sequences into account and work well on non-coding parts of the genome. Such models can be used similarly to pHMM, but have the advantage of not requiring a multiple sequence alignment and working on a collection of genes or other sequences such as genomic islands only, while also being able to model more powerful relationships compared to pHMMs. This shows that *deepG* models can be applied to a wider range of sequences and genomic regions compared to pHMMs.

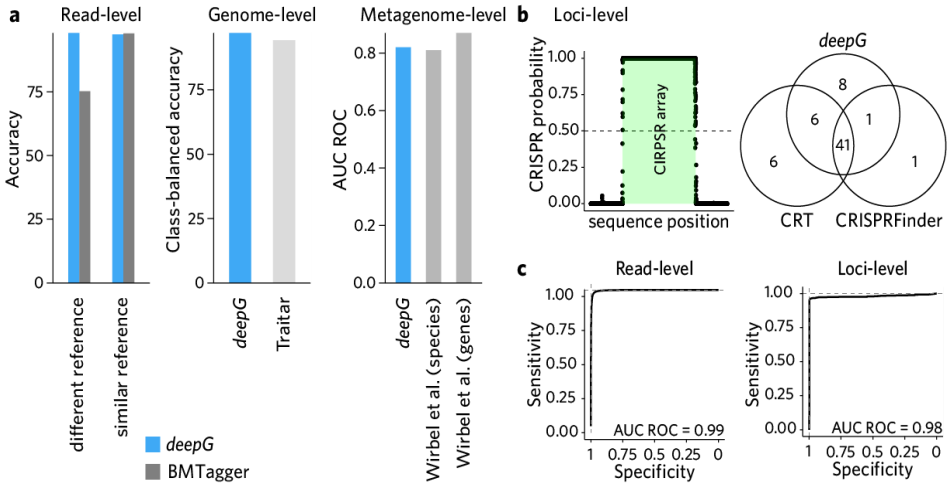


Figure 2: deepG performance achieves accuracy comparable or superior to highly specialized state of the art tools. a) Comparison to baseline methods on the read-level (contamination detection), genome-level (sporulation prediction), and metagenome-level (CRC prediction). Gene-level predictions (16S rRNA detection) are not shown, since both methods achieve an accuracy of 100% b) *deepG* for CRISPR-detection (loci-level). Left: Predicted CRISPR probability (points) goes up around a known CRISPR locus (green). Right: Venn diagram showing the number of CRISPR arrays detected with each method. c) ROC of deepG predictions on the read-level and the loci-level task.

At the gene level, we used *deepG* to build and train a classifier capable of detecting 16S rRNA genes (context size of 500 nt), achieving a balanced accuracy of 0.975 within 1 hour of training using the genomic gene pool as the background. *deepG* can apply such models within a sliding window over an input sequence to screen for possible hits, as demonstrated when we identified the location of this gene in *E. faecalis*. We compared the predictive performance with Barrnap[11], a bacterial ribosomal RNA predictor based on HMMs on 1,059 genomes. In 99.3% of genomes, both tools agree. For two genomes, where Barrnap outputs warnings due to a low alignment fraction, the *deepG* model reports no hits, while the *deepG* reports false positives on 5 genomes that could be filtered out using a reasonable length cutoff of at least 800 nt and mean aggregated confidence of at least 0.8.

To demonstrate the application of *deepG* at the genome level, we predicted bacterial morphology from the genomic sequence. Here, we used labels provided by the BacDive database[12] on sporulation, which is of particular interest due to their potential epidemiological danger. With the library, we trained a model that predicts the ability to sporulate based on subsequences of a length of 1M nt. We applied the model on full genomes (test data) unseen during training, with an average inference time of less than 7 seconds per genome. Since the aforementioned context size of 1M nt might be smaller than the size of the desired genome to be predicted, *deepG* runs multiple predictions over the genomic sequence (every 100,000 nt) and aggregates these to the final prediction. On this task, *deepG* had a balanced accuracy of 97.1% on the test set and incorrectly classified 15 genomes out of 512, while Traitar, a tool that infers sporulation and other phenotype properties using presence/absence information of gene families[13], falsely classified 29 genomes (94.3% balanced accuracy). The improved performance of *deepG* compared to established approaches like Traitar demonstrates the application of *deepG* on predicting bacterial sporulation from genomic sequence, due to *deepG*'s comparably high accuracy and short prediction time. Our model was trained for around 10 days on a single data center GPU and trained on a different dataset than Traitar, which could also be responsible for the increase in accuracy.

To demonstrate the application of *deepG* to full metagenomes, we trained a supervised model on the Chinese cohort from the colorectal cancer (CRC) study (128 metagenomes)[14]. The input data are the full metagenome samples of the study, with individuals grouped into CRC and healthy subjects. The AUC ROC of the resulting *deepG* model is 0.82 – similar to the AUC ROC score reported by the authors of the original study, with 0.81 and 0.87 AUC ROC based on different analysis strategies. Therefore, *deepG* achieved a similar performance without the need for any alignment, taxonomic annotations, or functional annotations. Additionally, *deepG* is not limited to coding regions or the identification of functional groups. Furthermore, *deepG*'s ability to train on full metagenomes allows for a more comprehensive data analysis compared to methods relying on annotations or coding regions. To facilitate model training in scenarios where the sequence length is long, where one input sample might be a whole metagenomic sample, *deepG* can be run using a "set training"[15] regime, where no order is encoded within subsamples (contigs).

While most aforementioned applications describe the construction of a supervised binary classifier, where a model is trained from scratch to discriminate between two classes, *deepG* can also run in a multi-label setting – e.g., for the prediction of multiple classes at a taxonomic rank or when metagenomes are grouped into more than two sets. Moreover, since *deepG* models are implemented with the Keras functional API, it supports custom and more advanced models – e.g., when the user has access to additional information, such as clinical metadata, that can be used as input for the model. Besides supervised training, *deepG* also supports unsupervised training with support for additional training modes, such as Contrastive Predictive Coding[16] and *Self-GenomeNet*[*]. In this setting, a model is trained on unlabeled data and can later serve as a fundament for supervised tasks to increase performance and speed up model convergence, as this enables the efficient use of unlabeled datasets by efficiently capturing

representations. *deepG* also supports Tensorboard, a tool that allows users to track training runs and generate custom metrics, such as balanced accuracy (**Fig. S3**).

While further tools for implementing DL models for genome sequence data are available (as reviewed in Alharbi *et al.*[17], the python packages Janggu[18] and Selene[19] also apply DL models on genomic input), these tools are designed with a focus on applying DL on human genome data. *deepG* further implements data augmentation strategies to handle scenarios that arise when DL is applied to microbial data collections. Furthermore, we demonstrated *deepG* to be effective for bacterial and viral and mixed sequence origins of different species, from 150 nt read-level to (meta)genomes using set learning.

To support the full taxonomic range and different sequence lengths, *deepG* comes with a range of data augmentation methods and training schemes that are required for such input types (**Fig. 1C**) and makes DL training possible for non-human datasets. Another key feature of *deepG* is its support for both supervised and unsupervised models. This provides researchers the capability to use pre-trained models, e.g., to improve supervised tasks to speed up training time and accuracy. It could also be used for clustering based on the neural representation which could be used as an alternative to classical clustering methods which may provide more accurate and robust results compared to traditional methods[20].

The *deepG* software resource will enable many researchers to create and apply customized learning models when the classification problem is very specific and other software tools are outdated or unavailable, or where classical machine learning tools reach their limitations, e.g. due to Markov Assumptions when applying pHMM models or the lack of known features. *deepG* code, documentation, and interactive case studies can be found at the following URL: https://deepG.de.

Discussion

We have presented deepG, a versatile deep learning library designed to meet the specific challenges encountered in genomics research. Our GenomeNet models have shown promise in a range of applications, from taxonomic classifications to functional genomics. However, like all models, they are not without limitations. One possible area for improvement is the optimization for less-represented organisms, which could enhance the library's overall performance and utility.

The strength of the deepG-GenomeNet ecosystem lies not only in its current capabilities but also in its potential for growth through community contributions. GenomeNet.de's model card system allows for a level of transparency that encourages user contributions and critique. We envision a community-driven platform where researchers can easily share, validate, and utilize models for a range of genomics applications. Future developments include the integration of new data types and formats, enhanced data visualization tools, and the creation of a community forum for users to discuss challenges, solutions, and best practices.

While deepG and GenomeNet show great potential and flexibility in tackling various challenges in genomics research, some limitations are worth noting. At present, the inference service can only handle files up to 2 MB, which could be a drawback for users who need to analyze larger datasets quickly. Also, there's no strict limit on model sizes, but using GPU-accelerated inference is only possible if the model fits within the memory of a T4 GPU. This sets a practical upper limit for those needing GPU power.

deepG and GenomeNet offer a new paradigm in genomics research by reducing barriers to entry and fostering a collaborative environment. The platform's versatility and adaptability make it poised to contribute significantly to future advancements in the field. We anticipate that deepG will be a cornerstone in the evolution of genomics, offering a robust and scalable solution for complex problems in the domain.

References

1. Eraslan, G., Avsec, Ž., Gagneur, J. & Theis, F. J. Deep learning: new computational modelling techniques for genomics. *Nat. Rev. Genet.* **20**, 389–403 (2019).
2. Tampuu, A., Bzhalava, Z., Dillner, J. & Vicente, R. ViraMiner: Deep learning on raw DNA sequences for identifying viral genomes in human samples. *PLoS One* **14**, e0222271 (2019).
3. Arango-Argoty, G. *et al.* DeepARG: a deep learning approach for predicting antibiotic resistance genes from metagenomic data. *Microbiome* **6**, 23 (2018).
4. Deng, Z.-L., Münch, P. C., Mreches, R. & McHardy, A. C. Rapid and accurate identification of ribosomal RNA sequences via deep learning. *Nucleic Acids Res.* (2022) doi:10.1093/nar/gkac112.
5. van den Oord, A. *et al.* WaveNet: A Generative Model for Raw Audio. *arXiv [cs.SD]* (2016).
6. Rotmistrovsky, K. & Agarwala, R. BMTagger: Best Match Tagger for removing human reads from metagenomics datasets. *Unpublished* (2011).
7. Eddy, S. R. Profile hidden Markov models. *Bioinformatics* **14**, 755–763 (1998).
8. Wheeler, T. J. & Eddy, S. R. nhmmer: DNA homology search with profile HMMs. *Bioinformatics* **29**, 2487–2489 (2013).

9. Brouns, S. J. J. *et al.* Small CRISPR RNAs guide antiviral defense in prokaryotes. *Science* **321**, 960–964 (2008).
10. Zhang, Q. & Ye, Y. Not all predicted CRISPR–Cas systems are equal: isolated cas genes and classes of CRISPR like elements. *BMC Bioinformatics* **18**, 1–12 (2017).
11. Seemann, T. & Booth, T. Barrnap: basic rapid ribosomal RNA predictor. *GitHub repository* (2018).
12. Söhngen, C., Bunk, B., Podstawka, A., Gleim, D. & Overmann, J. BacDive—the Bacterial Diversity Metadatabase. *Nucleic Acids Res.* **42**, D592–D599 (2013).
13. Weimann, A. *et al.* From Genomes to Phenotypes: Traitar, the Microbial Trait Analyzer. *mSystems* **1**, (2016).
14. Wirbel, J. *et al.* Meta-analysis of fecal metagenomes reveals global microbial signatures that are specific for colorectal cancer. *Nat. Med.* **25**, 679–689 (2019).
15. Zaheer, M. *et al.* Deep Sets. *arXiv [cs.LG]* (2017).
16. van den Oord, A., Li, Y. & Vinyals, O. Representation Learning with Contrastive Predictive Coding. *arXiv [cs.LG]* (2018).
17. Alharbi, W. S. & Rashid, M. A review of deep learning applications in human genomics using next-generation sequencing data. *Hum. Genomics* **16**, 26 (2022).
18. Kopp, W., Monti, R., Tamburrini, A., Ohler, U. & Akalin, A. Deep learning for genomics using Janggu. *Nat. Commun.* **11**, 3488 (2020).
19. Chen, K. M., Cofer, E. M., Zhou, J. & Troyanskaya, O. G. Selene: a PyTorch-based deep learning library for sequence data. *Nat. Methods* **16**, 315–318 (2019).
20. Karim, M. R. *et al.* Deep learning-based clustering approaches for bioinformatics. *Brief. Bioinform.* **22**, 393–415 (2021).

References

Achiam J, Adler S, Agarwal S, Ahmad L, Akkaya I, Aleman FL, Almeida D, Altenschmidt J, Altman S, Anadkat S, *et al.* (2023). "Gpt-4 technical report." *arXiv preprint arXiv:2303.08774*.

Agrawal R, Imieliński T, Swami A (1993). "Mining association rules between sets of items in large databases." In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pp. 207–216.

Alipanahi B, Delong A, Weirauch MT, Frey BJ (2015). "Predicting the sequence specificities of DNA-and RNA-binding proteins by deep learning." *Nature biotechnology*, **33**(8), 831–838.

Auslander N, Gussow AB, Benler S, Wolf YI, Koonin EV (2020). "Seeker: alignment-free identification of bacteriophage genomes by deep learning." *Nucleic acids research*, **48**(21), e121–e121.

Bajwa A, Rastogi R, Kathail P, Shuai RW, Ioannidis N (2024). "Characterizing uncertainty in predictions of genomic sequence-to-activity models." In *Machine Learning in Computational Biology*, pp. 279–297. PMLR.

Baker B, Gupta O, Naik N, Raskar R (2016). "Designing neural network architectures using reinforcement learning." *arXiv preprint arXiv:1611.02167*.

Bardes A, Ponce J, LeCun Y (2021). "Vicreg: Variance-invariance-covariance regularization for self-supervised learning." *arXiv preprint arXiv:2105.04906*.

Bartoszewicz JM, Nasri F, Nowicka M, Renard BY (2022). "Detecting DNA of novel fungal pathogens using ResNets and a curated fungi-hosts data collection." *Bioinformatics*, **38**(Supplement_2), ii168–ii174.

Bartoszewicz JM, Seidel A, Rentzsch R, Renard BY (2020). "DeePaC: predicting pathogenic potential of novel DNA with reverse-complement neural networks." *Bioinformatics*, **36**(1), 81–89.

Berthelot D, Carlini N, Goodfellow I, Papernot N, Oliver A, Raffel CA (2019). "Mixmatch: A holistic approach to semi-supervised learning." *Advances in neural information processing systems*, **32**.

Bridle JS (1990). "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition." In *Neurocomputing: Algorithms, architectures and applications*, pp. 227–236. Springer.

Brier GW (1950). "Verification of forecasts expressed in terms of probability." *Monthly weather review*, **78**(1), 1–3.

Chen KM, Cofer EM, Zhou J, Troyanskaya OG (2019). "Selene: a PyTorch-based deep learning library for sequence data." *Nature methods*, **16**(4), 315–318.

Chen T, Goodfellow I, Shlens J (2015). "Net2net: Accelerating learning via knowledge transfer." *arXiv preprint arXiv:1511.05641*.

Chen T, Kornblith S, Norouzi M, Hinton G (2020). "A simple framework for contrastive learning of visual representations." In *International conference on machine learning*, pp. 1597–1607. PMLR.

Chen X, Xie L, Wu J, Tian Q (2021). "Progressive darts: Bridging the optimization gap for nas in the wild." *International Journal of Computer Vision*, **129**, 638–655.

Cho K (2014). "On the Properties of Neural Machine Translation: Encoder-decoder Approaches." *arXiv preprint arXiv:1409.1259*.

Chrabaszcz P, Loshchilov I, Hutter F (2017). "A downsampled variant of imagenet as an alternative to the cifar datasets." *arXiv preprint arXiv:1707.08819*.

Ciortan M, Defrance M (2021). "Contrastive self-supervised clustering of scRNA-seq data." *BMC bioinformatics*, **22**(1), 280.

Dalla-Torre H, Gonzalez L, Mendoza-Revilla J, Carranza NL, Grzywaczewski AH, Oteri F, Dallago C, Trop E, de Almeida BP, Sirelkhatim H, *et al.* (2023). "The nucleotide transformer: Building and evaluating robust foundation models for human genomics." *BioRxiv*, pp. 2023–01.

DeGroot MH, Fienberg SE (1983). "The comparison and evaluation of forecasters." *Journal of the Royal Statistical Society: Series D (The Statistician)*, **32**(1-2), 12–22.

Devlin J, Chang MW, Lee K, Toutanova K (2018). "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805*.

Dosovitskiy A (2020). "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929*.

El-Yaniv R, *et al.* (2010). "On the Foundations of Noise-free Selective Classification." *Journal of Machine Learning Research*, **11**(5).

Elsken T, Metzen JH, Hutter F (2018). "Efficient multi-objective neural architecture search via lamarckian evolution." *arXiv preprint arXiv:1804.09081*.

Elsken T, Metzen JH, Hutter F (2019). "Neural architecture search: A survey." *Journal of Machine Learning Research*, **20**(55), 1–21.

Fang Z, Tan J, Wu S, Li M, Xu C, Xie Z, Zhu H (2019). "PPR-Meta: a tool for identifying phages and plasmids from metagenomic fragments using deep learning." *Gigascience*, **8**(6), giz066.

Fiannaca A, La Paglia L, La Rosa M, Renda G, Rizzo R, Gaglio S, Urso A, *et al.* (2018). "Deep learning models for bacteria taxonomic classification of metagenomic data." *BMC bioinformatics*, **19**(7), 198.

Floridi L, Chiriatti M (2020). "GPT-3: Its nature, scope, limits, and consequences." *Minds and Machines*, **30**, 681–694.

Gage P (1994). "A new algorithm for data compression." *The C Users Journal*, **12**(2), 23–38.

Gal Y, Ghahramani Z (2016). "Dropout as a bayesian approximation: Representing model uncertainty in deep learning." In *international conference on machine learning*, pp. 1050–1059. PMLR.

Gawlikowski J, Tassi CRN, Ali M, Lee J, Humt M, Feng J, Kruspe A, Triebel R, Jung P, Roscher R, *et al.* (2023). "A survey of uncertainty in deep neural networks." *Artificial Intelligence Review*, **56**(Suppl 1), 1513–1589.

Geifman Y, El-Yaniv R (2017). "Selective classification for deep neural networks." *Advances in neural information processing systems*, **30**.

Goodfellow I (2016). "Deep learning."

Goyal P, Caron M, Lefaudeux B, Xu M, Wang P, Pai V, Singh M, Liptchinsky V, Misra I, Joulin A, *et al.* (2021). "Self-supervised pretraining of visual features in the wild." *arXiv preprint arXiv:2103.01988*.

Gündüz HA, Binder M, To XY, Mreches R, Bischl B, McHardy AC, Münch PC, Rezaei M (2023). "A self-supervised deep learning method for data-efficient training in genomics." *Communications Biology*, **6**(1), 928. doi:10.1038/s42003-023-05310-2.

Gündüz HA, Mreches R, Moosbauer J, Robertson G, To XY, Franzosa EA, Huttenhower C, Rezaei M, McHardy AC, Bischl B, Münch PC, Binder M (2024). "Optimized model architectures for deep learning on genomic data." *Communications Biology*, **7**(1), 516. doi:10.1038/s42003-024-06161-1.

Guo C, Pleiss G, Sun Y, Weinberger KQ (2017). "On calibration of modern neural networks." In *International conference on machine learning*, pp. 1321–1330. PMLR.

Gündüz HA, Giri S, Binder M, Bischl B, Rezaei M (2023). "Uncertainty Quantification for Deep Learning Models Predicting the Regulatory Activity of DNA Sequences." In *2023 International Conference on Machine Learning and Applications (ICMLA)*, pp. 566–573. doi:10.1109/ICMLA58977.2023.00084.

Han J, Moraga C (1995). "The influence of the sigmoid function parameters on the speed of backpropagation learning." In *International workshop on artificial neural networks*, pp. 195–201. Springer.

He K, Zhang X, Ren S, Sun J (2015). "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034.

He K, Zhang X, Ren S, Sun J (2016). "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

He X, Zhao K, Chu X (2021). "AutoML: A survey of the state-of-the-art." *Knowledge-based systems*, **212**, 106622.

Heather JM, Chain B (2016). "The sequence of sequencers: The history of sequencing DNA." *Genomics*, **107**(1), 1–8.

Henaff O (2020). "Data-efficient image recognition with contrastive predictive coding." In *International conference on machine learning*, pp. 4182–4192. PMLR.

Hendrycks D, Gimpel K (2016). "A baseline for detecting misclassified and out-of-distribution examples in neural networks." *arXiv preprint arXiv:1610.02136*.

Hie B, Bryson BD, Berger B (2020). "Leveraging uncertainty in machine learning accelerates biological discovery and design." *Cell systems*, **11**(5), 461–477.

Hochreiter S (1997). "Long Short-term Memory." *Neural Computation MIT-Press*.

Holley RW, Apgar J, Everett GA, Madison JT, Marquisee M, Merrill SH, Penswick JR, Zamir A (1965). "Structure of a ribonucleic acid." *Science*, **147**(3664), 1462–1465.

Huang G, Liu Z, Van Der Maaten L, Weinberger KQ (2017). "Densely Connected Convolutional Networks." In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269. doi:10.1109/CVPR.2017.243.

Hüllermeier E, Waegeman W (2021). "Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods." *Machine learning*, **110**(3), 457–506.

Irwin-Harris W, Sun Y, Xue B, Zhang M (2019). "A graph-based encoding for evolutionary convolutional neural network architecture design." In *2019 IEEE Congress on Evolutionary Computation (CEC)*, pp. 546–553. IEEE.

Ji Y, Zhou Z, Liu H, Davuluri RV (2021). "DNABERT: pre-trained Bidirectional Encoder Representations from Transformers model for DNA-language in genome." *Bioinformatics*, **37**(15), 2112–2120.

Kandasamy K, Neiswanger W, Schneider J, Poczos B, Xing EP (2018). "Neural architecture search with bayesian optimisation and optimal transport." *Advances in neural information processing systems*, **31**.

Kaplan S, Giryes R (2020). "Self-supervised neural architecture search." *arXiv preprint arXiv:2007.01500*.

Katharopoulos A, Vyas A, Pappas N, Fleuret F (2020). "Transformers are rnns: Fast autoregressive transformers with linear attention." In *International conference on machine learning*, pp. 5156–5165. PMLR.

Kelley DR, Reshef YA, Bileschi M, Belanger D, McLean CY, Snoek J (2018). "Sequential regulatory activity prediction across chromosomes with convolutional neural networks." *Genome research*, **28**(5), 739–750.

Kingma DP (2013). "Auto-encoding variational bayes." *arXiv preprint arXiv:1312.6114*.

Klein A, Falkner S, Bartels S, Hennig P, Hutter F (2017). "Fast bayesian optimization of machine learning hyperparameters on large datasets." In *Artificial intelligence and statistics*, pp. 528–536. PMLR.

Klein A, Falkner S, Springenberg JT, Hutter F (2022). "Learning curve prediction with Bayesian neural networks." In *International conference on learning representations*.

Koh PW, Pierson E, Kundaje A (2017). "Denoising genome-wide histone ChIP-seq with convolutional neural networks." *Bioinformatics*, **33**(14), i225–i233.

Kopp W, Monti R, Tamburrini A, Ohler U, Akalin A (2020). "Deep learning for genomics using Janggu." *Nature communications*, **11**(1), 3488.

Krizhevsky A, Sutskever I, Hinton GE (2012). "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*, **25**.

Lakshminarayanan B, Pritzel A, Blundell C (2017). "Simple and scalable predictive uncertainty estimation using deep ensembles." *Advances in neural information processing systems*, **30**.

LeCun Y, Bottou L, Bengio Y, Haffner P (1998). "Gradient-based learning applied to document recognition." *Proceedings of the IEEE*, **86**(11), 2278–2324.

Levi D, Gispan L, Giladi N, Fetaya E (2022). "Evaluating and calibrating uncertainty prediction in regression tasks." *Sensors*, **22**(15), 5540.

Liang Q, Bible PW, Liu Y, Zou B, Wei L (2020). "DeepMicrobes: taxonomic classification for metagenomics with deep learning." *NAR Genomics and Bioinformatics*, **2**(1), lqaa009.

Liu C, Dollár P, He K, Girshick R, Yuille A, Xie S (2020). "Are labels necessary for neural architecture search?" In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IV 16*, pp. 798–813. Springer.

Liu C, Zoph B, Neumann M, Shlens J, Hua W, Li LJ, Fei-Fei L, Yuille A, Huang J, Murphy K (2018a). "Progressive neural architecture search." In *Proceedings of the European conference on computer vision (ECCV)*, pp. 19–34.

Liu H, Simonyan K, Yang Y (2018b). "Darts: Differentiable architecture search." *arXiv preprint arXiv:1806.09055*.

Liu S, Mallol-Ragolta A, Parada-Cabaleiro E, Qian K, Jing X, Kathan A, Hu B, Schuller BW (2022). "Audio self-supervised learning: A survey." *Patterns*, **3**(12).

Lu AX, Zhang H, Ghassemi M, Moses A (2020). "Self-supervised contrastive learning of protein representations by mutual information maximization." *BioRxiv*, pp. 2020–09.

Luo R, Tian F, Qin T, Chen E, Liu TY (2018). "Neural architecture optimization." *Advances in neural information processing systems*, **31**.

Maas AL, Hannun AY, Ng AY, *et al.* (2013). "Rectifier nonlinearities improve neural network acoustic models." In *Proc. icml*, volume 30, p. 3. Atlanta, GA.

MacDonald S, Foley H, Yap M, Johnston RL, Steven K, Koufariotis LT, Sharma S, Wood S, Addala V, Pearson JV, *et al.* (2023). "Generalising uncertainty improves accuracy and safety of deep learning analytics applied to oncology." *Scientific Reports*, **13**(1), 7395.

Maćkiewicz A, Ratajczak W (1993). "Principal components analysis (PCA)." *Computers & Geosciences*, **19**(3), 303–342.

Mehrtash A, Wells WM, Tempany CM, Abolmaesumi P, Kapur T (2020). "Confidence calibration and predictive uncertainty estimation for deep medical image segmentation." *IEEE transactions on medical imaging*, **39**(12), 3868–3878.

Mendoza-Revilla J, Trop E, Gonzalez L, Roller M, Dalla-Torre H, de Almeida BP, Richard G, Caton J, Lopez Carranza N, Skwark M, *et al.* (2024). "A foundational large language model for edible plant genomes." *Communications Biology*, **7**(1), 835.

Min B, Ross H, Sulem E, Veyseh APB, Nguyen TH, Sainz O, Agirre E, Heintz I, Roth D (2023). "Recent advances in natural language processing via large pre-trained language models: A survey." *ACM Computing Surveys*, **56**(2), 1–40.

Mnih V (2013). "Playing atari with deep reinforcement learning." *arXiv preprint arXiv:1312.5602*.

Mreches R, To XY, Gündüz HA, Moosbauer J, Klawitter S, Deng ZL, Robertson G, Rezaei M, Asgari E, Franzosa EA, Huttenhower C, Bischl B, McHardy AC, Binder M, Münch PC (2024). "GenomeNet: A platform for deep learning on (meta)genomic sequences."

Müller R, Kornblith S, Hinton GE (2019). "When does label smoothing help?" *Advances in neural information processing systems*, **32**.

Naeini MP, Cooper G, Hauskrecht M (2015). "Obtaining well calibrated probabilities using bayesian binning." In *Proceedings of the AAAI conference on artificial intelligence*, volume 29.

Nair V, Hinton GE (2010). "Rectified linear units improve restricted boltzmann machines." In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814.

Niculescu-Mizil A, Caruana R (2005). "Predicting good probabilities with supervised learning." In *Proceedings of the 22nd international conference on Machine learning*, pp. 625–632.

Nixon J, Dusenberry MW, Zhang L, Jerfel G, Tran D (2019). "Measuring Calibration in Deep Learning." In *CVPR workshops*, volume 2.

Oord Avd, Li Y, Vinyals O (2018). "Representation learning with contrastive predictive coding." *arXiv preprint arXiv:1807.03748*.

Oren M, Hassid M, Adi Y, Schwartz R (2024). "Transformers are multi-state rnns." *arXiv preprint arXiv:2401.06104*.

Parikh AP, Täckström O, Das D, Uszkoreit J (2016). "A decomposable attention model for natural language inference." *arXiv preprint arXiv:1606.01933*.

Patel K, Beluch W, Zhang D, Pfeiffer M, Yang B (2021). "On-manifold adversarial data augmentation improves uncertainty calibration." In *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 8029–8036. IEEE.

Pearce T, Leibfried F, Brintrup A (2020). "Uncertainty in neural networks: Approximately bayesian ensembling." In *International conference on artificial intelligence and statistics*, pp. 234–244. PMLR.

Pham H, Guan M, Zoph B, Le Q, Dean J (2018). "Efficient neural architecture search via parameters sharing." In *International conference on machine learning*, pp. 4095–4104. PMLR.

Polyak BT (1964). "Some methods of speeding up the convergence of iteration methods." *Ussr computational mathematics and mathematical physics*, **4**(5), 1–17.

Quang D, Xie X (2016). "DanQ: a hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences." *Nucleic acids research*, **44**(11), e107–e107.

Quang D, Xie X (2019). "FactorNet: a deep learning framework for predicting cell type specific transcription factor binding from nucleotide-resolution sequential data." *Methods*, **166**, 40–47.

Rainio O, Teuho J, Klén R (2024). "Evaluation metrics and statistical tests for machine learning." *Scientific Reports*, **14**(1), 6086.

Real E, Aggarwal A, Huang Y, Le QV (2019). "Regularized evolution for image classifier architecture search." In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pp. 4780–4789.

Real E, Moore S, Selle A, Saxena S, Suematsu YL, Tan J, Le QV, Kurakin A (2017). "Large-scale evolution of image classifiers." In *International conference on machine learning*, pp. 2902–2911. PMLR.

Ren J, Song K, Deng C, Ahlgren NA, Fuhrman JA, Li Y, Xie X, Poplin R, Sun F (2020). "Identifying viruses from metagenomic data using deep learning."

Rives A, Meier J, Sercu T, Goyal S, Lin Z, Liu J, Guo D, Ott M, Zitnick CL, Ma J, *et al.* (2021). "Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences." *Proceedings of the National Academy of Sciences*, **118**(15), e2016239118.

Romano Y, Patterson E, Candes E (2019). "Conformalized quantile regression." *Advances in neural information processing systems*, **32**.

Ronaghi M, Karamohamed S, Pettersson B, Uhlén M, Nyrén P (1996). "Real-time DNA sequencing using detection of pyrophosphate release." *Analytical biochemistry*, **242**(1), 84–89.

Sanger F, Air GM, Barrell BG, Brown NL, Coulson AR, Fiddes JC, Hutchison III CA, Slocombe PM, Smith M (1977a). "Nucleotide sequence of bacteriophage φX174 DNA." *nature*, **265**(5596), 687–695.

Sanger F, Nicklen S, Coulson AR (1977b). "DNA sequencing with chain-terminating inhibitors." *Proceedings of the national academy of sciences*, **74**(12), 5463–5467.

Scheppach A, Gündüz HA, Dorigatti E, Münch PC, McHardy AC, Bischl B, Rezaei M, Binder M (2023). "Neural Architecture Search for Genomic Sequence Data." In *2023 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, pp. 1–10. doi:10.1109/CIBCB56990.2023.10264875.

Sennrich R (2015). "Neural machine translation of rare words with subword units." *arXiv preprint arXiv:1508.07909*.

Shang J, Sun Y (2020). "CHEER: hierarCHical taxonomic classification for viral mEtagEnomic data via deep leaRning."

Shrikumar A, Greenside P, Kundaje A (2017). "Reverse-complement parameter sharing improves deep learning models for genomics." *BioRxiv*, p. 103663.

Sun Y, Ming Y, Zhu X, Li Y (2022). "Out-of-distribution detection with deep nearest neighbors." In *International Conference on Machine Learning*, pp. 20827–20840. PMLR.

Sutskever I, Martens J, Dahl G, Hinton G (2013). "On the importance of initialization and momentum in deep learning." In *International conference on machine learning*, pp. 1139–1147. PMLR.

Sykes AO (1993). "An introduction to regression analysis."

Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z (2016). "Rethinking the inception architecture for computer vision." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826.

Tampuu A, Bzhalava Z, Dillner J, Vicente R (2019). "ViraMiner: Deep learning on raw DNA sequences for identifying viral genomes in human samples." *PloS one*, **14**(9), e0222271.

Thulasidasan S, Chennupati G, Bilmes JA, Bhattacharya T, Michalak S (2019). "On mixup training: Improved calibration and predictive uncertainty for deep neural networks." *Advances in neural information processing systems*, **32**.

Turkoglu MO, Becker A, Gündüz HA, Rezaei M, Bischl B, Daudt RC, D' Aronco S, Wegner J, Schindler K (2022). "FiLM-Ensemble: Probabilistic Deep Learning via Feature-wise Linear Modulation." In S Koyejo, S Mohamed, A Agarwal, D Belgrave, K Cho, A Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 22229–22242. Curran Associates, Inc. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/8bd31288ad8e9a31d519fdeede7ee47d-Paper-Conference.pdf.

Vahidi A, Wimmer L, Gündüz HA, Bischl B, Hüllermeier E, Rezaei M (2024). "Diversified Ensemble of Independent Sub-networks for Robust Self-supervised Representation Learning." In A Bifet, J Davis, T Krilavičius, M Kull, E Ntoutsi, I Žliobaitė (eds.), *Machine Learning and Knowledge Discovery in Databases. Research Track. ECML PKDD 2024*, pp. 38–55. Springer Nature Switzerland, Cham. doi:10.1007/978-3-031-70341-6_3.

Van Den Oord A, Dieleman S, Zen H, Simonyan K, Vinyals O, Graves A, Kalchbrenner N, Senior A, Kavukcuoglu K, *et al.* (2016). "Wavenet: A generative model for raw audio." *arXiv preprint arXiv:1609.03499*, **12**.

Van den Oord A, Kalchbrenner N, Espeholt L, Vinyals O, Graves A, *et al.* (2016). "Conditional image generation with pixelcnn decoders." *Advances in neural information processing systems*, **29**.

Van Den Oord A, Kalchbrenner N, Kavukcuoglu K (2016). "Pixel recurrent neural networks." In *International conference on machine learning*, pp. 1747–1756. PMLR.

Van der Maaten L, Hinton G (2008). "Visualizing data using t-SNE." *Journal of machine learning research*, **9**(11).

Vaswani A (2017). "Attention is all you need." *Advances in Neural Information Processing Systems*.

Wang R, Zang T, Wang Y (2019). "Human mitochondrial genome compression using machine learning techniques." *Human Genomics*, **13**(1), 1–8.

Watson JD, Crick FH (1953). "The structure of DNA." In *Cold Spring Harbor symposia on quantitative biology*, volume 18, pp. 123–131. Cold Spring Harbor Laboratory Press.

Wen Y, Tran D, Ba J (2020). "Batchensemble: an alternative approach to efficient ensemble and lifelong learning." *arXiv preprint arXiv:2002.06715*.

Wen Y, Vicol P, Ba J, Tran D, Grosse R (2018). "Flipout: Efficient pseudo-independent weight perturbations on mini-batches." *arXiv preprint arXiv:1803.04386*.

Wistuba M (2017). "Bayesian optimization combined with incremental evaluation for neural network architecture optimization." In *Proceedings of the International Workshop on Automatic Selection, Configuration and Composition of Machine Learning Algorithms*.

Wong C, Houlsby N, Lu Y, Gesmundo A (2018). "Transfer learning with neural automl." *Advances in neural information processing systems*, **31**.

Xu D, Tian Y (2015). "A comprehensive survey of clustering algorithms." *Annals of data science*, **2**, 165–193.

Yang J, Zhou K, Li Y, Liu Z (2021). "Generalized out-of-distribution detection: A survey." *arXiv preprint arXiv:2110.11334*.

Zeng H, Edwards MD, Liu G, Gifford DK (2016). "Convolutional neural network architectures for predicting DNA–protein binding." *Bioinformatics*, **32**(12), i121–i127.

Zhang H, Cisse M, Dauphin YN, Lopez-Paz D (2017). "mixup: Beyond empirical risk minimization." *arXiv preprint arXiv:1710.09412*.

Zhang X, Xie X, Ma L, Du X, Hu Q, Liu Y, Zhao J, Sun M (2020). "Towards characterizing adversarial defects of deep learning software from the lens of uncertainty." In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pp. 739–751.

Zhang Z, Cofer EM, Troyanskaya OG (2021a). "AMBIENT: accelerated convolutional neural network architecture search for regulatory genomics." *bioRxiv*, pp. 2021–02.

Zhang Z, Park CY, Theesfeld CL, Troyanskaya OG (2021b). "An automated framework for efficiently designing deep convolutional neural networks in genomics." *Nature Machine Intelligence*, **3**(5), 392–400.

Zhong Z, Yan J, Wu W, Shao J, Liu CL (2018). "Practical block-wise neural network architecture generation." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2423–2432.

Zhou J, Troyanskaya OG (2015). "Predicting effects of noncoding variants with deep learning–based sequence model." *Nature methods*, **12**(10), 931–934.

Zhou Z, Ji Y, Li W, Dutta P, Davuluri R, Liu H (2023). "Dnabert-2: Efficient foundation model and benchmark for multi-species genome." *arXiv preprint arXiv:2306.15006*.

Zoph B, Le QV (2016). "Neural architecture search with reinforcement learning." *arXiv preprint arXiv:1611.01578*.

Zoph B, Vasudevan V, Shlens J, Le QV (2018). "Learning transferable architectures for scalable image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710.

Eidesstattliche Versicherung

(Siehe Promotionsordnung vom 12. Juli 2011, §8 Abs. 2 Pkt. 5)

Hiermit erkläre ich an Eides statt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

München, den 30.09.2024 Hüseyin Anil Gündüz