2023 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB) | 979-8-3503-1017-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/CIBCB56990.2023.10264875

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-DARTS, 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. Our experiments show that architectures optimized by our NAS methods outperform manually developed architectures while having significantly fewer parameters.**

***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

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

segmentation. Although there has been some research on what kind of models work 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.

The code for our methods and experiments is available at `https://github.com/GenomeNet/Genome-NAS`.

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.

In genome sequence data, one-dimensional Convolutional Neural Networks (1D-CNNs) are capable of learning relationships between local features, thereby detecting local patterns and motifs in DNA sequences. These CNNs generate sequential output when provided with sequential input. For tasks such as sequence classification, where properties of entire sequences are learned, it becomes necessary to consolidate this sequential output into a fixed-size vector. A widely used method for achieving this is pooling. For instance, max pooling can be interpreted as detecting the presence of specific patterns, while average pooling calculates pattern frequencies [23]. However, pooling layers fall short when learning interactions between input features that are distant. As an alternative, Recurrent Neural Networks (RNNs) offer a way to combine sequential input data. Architectures like Long Short-Term Memory (LSTM) RNNs [24] are explicitly designed to learn long-range dependencies.

An early application of CNNs for genome sequence data was the DeepSEA framework [5], which predicted biochemical properties of segments 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 [25], 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, [26] 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 based on the one used in [26], 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* $x^{(i)}, i \in \{1, \dots, V\}$ which 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 [27].

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 [28] due to the low effective dimensionality of this problem, and was also shown to be competitive in NAS [29], [30]. It can easily be used with any given search space.

Successive Halving (SH) [18], also called Sequential Halving [31], 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 [32] 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

[1]While [26] also perform NAS for RNNs, they do not use a DAG for it and have a different search space.

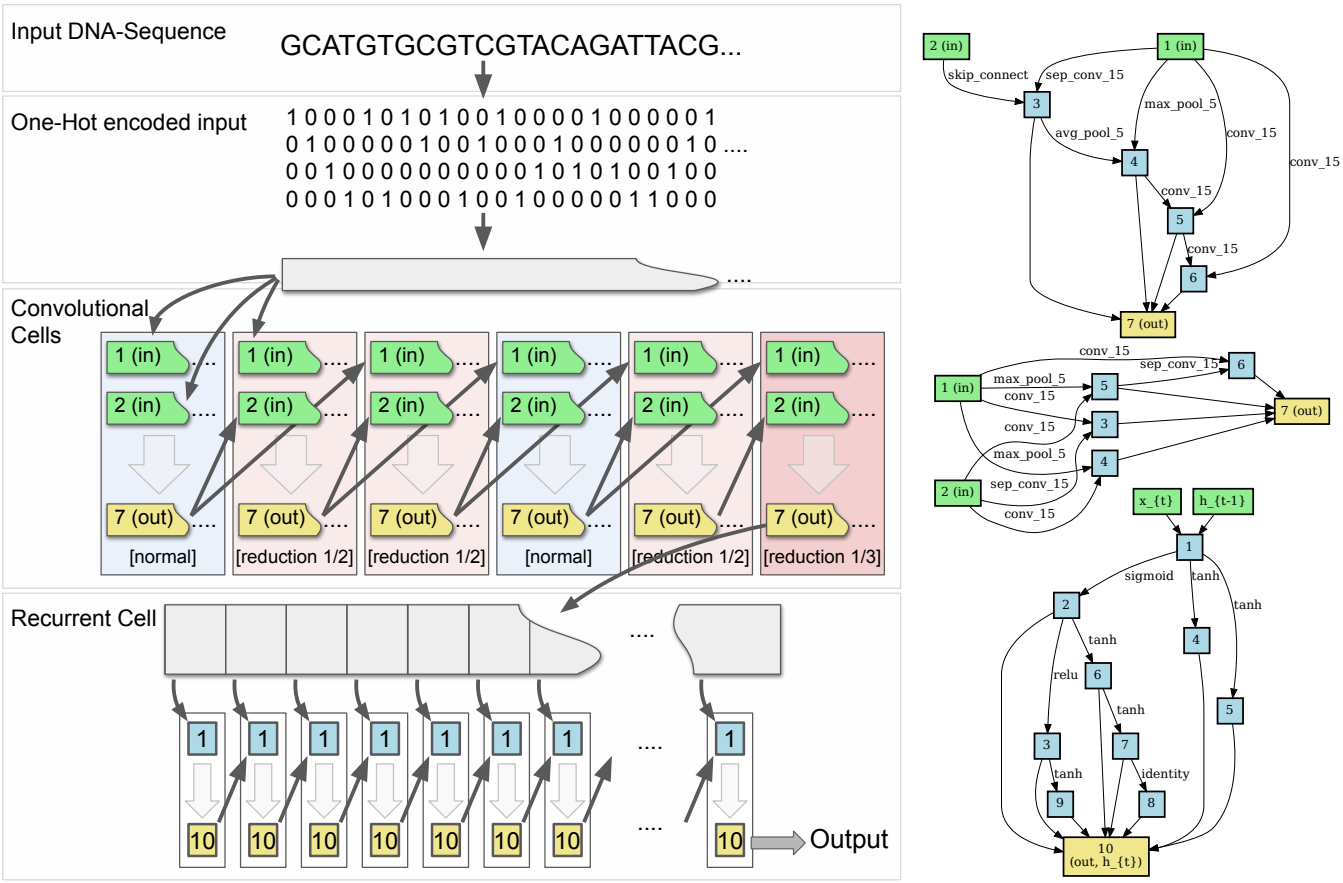

Figure 1. Neural Architectures. **Left**: Macro-Architecture, i.e. the way in which cells are connected. The input sequence of length n is one-hot encoded as a $4 \times n$ matrix and fed through an initial convolution layer. The data is then successively fed through convolutional cells, which get their 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.

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 edges, the operation (excluding *zero*) with the largest $\alpha^{(\mathbf{i},\mathbf{j})}$ 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 [33] for generating these candidates. Instead of using a Gaussian Process as surrogate model, as is often done, BONAS uses a graph-convolutional network [34] 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. [35] introduced BioNAS where they modified the NAS of [25] for protein binding prediction. They consider convolution, pooling, and fully-connected layers as their search space. AMBER [36] 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 [37] 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, they do not optimize the architecture of the recurrent network itself.

Similar 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 [38].

The operations $\mathcal{O}_{\mathrm{cnn}}$ 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 [26], and as in [16] each convolution operation is preceded by ReLU activation and followed by batch normalization. The operations $\mathcal{O}_{\mathrm{rnn}}$ 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 [26], 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 [26], 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 [26] 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, a one-hot encoded 1000-bp DNA sequence, is used to predict 919 binary chromatin features in a multi-label prediction setting. The data, provided by [5][2] under a CC Attribution 3.0 license, is already split into training, validation, and test sets. 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 we compare these architectures to those chosen by NAS algorithms. Each training run was conducted on a single Nvidia A100 40GB accelerator card 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.

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

(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 the 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 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 [39] score of 21.90 and an average ROC-AUC score [40] 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 8 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. Further research should therefore focus on evaluating our methods on other tasks where sequence models are being used, such as DNA methylation [41] or gene expression prediction [42]. While our specific implementation was set up to make a single prediction for a given input sequence it is straightforward to adapt the RNN layer to give a sequential output instead of a single vector.

A general limitation of the NAS approach is its requirement for a large amount of resources. Using smaller proxy tasks may be one approach to alleviate this, as it has been successfully in other domains [16].

Broader impact statement The fact that our NAS algorithms were able to beat expert-designed architectures on a benchmark dataset is a promising sign that NAS algorithms are able to provide a better design and a better performance for deep learning in genomics. Although NAS methods may seem expensive on the surface, 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 [43] and could contribute to the design of novel therapeutica due to the modeling of protein folding [44].

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.

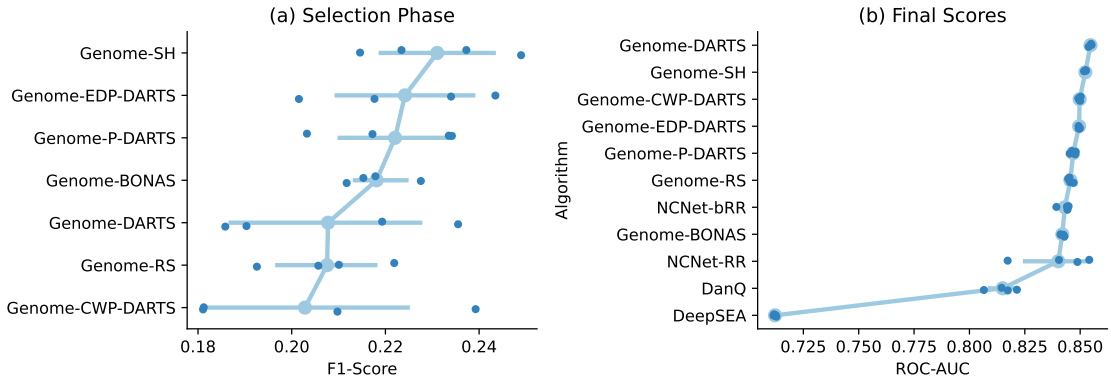

Figure 2. Performance of algorithms in the two phases of the evaluation. (a) selection phase: validation set performance of models, which is used to select models for the evaluation phase. Architecture search was performed four different times, each dot represents the performance of a separate NAS run. 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 [39] AS RECOMMENDED BY [6], ROC-AUC THE AREA UNDER THE RECEIVER OPERATOR CHARACTERISTIC ([40], 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

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 a 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 the large 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.

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] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017.

[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," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=S1eYHoC5FX

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

[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] A. Tampuu, Z. Bzhalava, J. Dillner, and R. Vicente, "Viraminer: Deep learning on raw dna sequences for identifying viral genomes in human samples," *PLOS ONE*, vol. 14, no. 9, pp. 1–17, 09 2019.

[24] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

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

[26] 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.

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

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

[29] S. Xie, A. Kirillov, R. Girshick, and K. He, "Exploring randomly wired neural networks for image recognition," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1284–1293.

[30] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in *Uncertainty in artificial intelligence*. PMLR, 2020, pp. 367–377.

[31] 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.
[32] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *Journal of Machine Learning Research*, vol. 18, no. 185, pp. 1–52, 2018.
[33] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, vol. 25, 2012.
[34] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *International Conference on Learning Representation, ICLR*, 2016.
[35] 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.
[36] 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.
[37] Z. Zhang, E. M. Cofer, and O. G. Troyanskaya, "Ambient: accelerated convolutional neural network architecture search for regulatory genomics," *bioRxiv*, 2021.
[38] L. Koumakis, "Deep learning models in genomics; are we there yet?" *Computational and Structural Biotechnology Journal*, vol. 18, pp. 1466–1473, 2020.
[39] 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.
[40] 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.
[41] J. Zhou, Q. Chen, P. R. Braun, K. A. Perzel Mandell, A. E. Jaffe, H. Y. Tan, T. M. Hyde, J. E. Kleinman, J. B. Potash, G. Shinozaki *et al.*, "Deep learning predicts dna methylation regulatory variants in the human brain and elucidates the genetics of psychiatric disorders," *Proceedings of the National Academy of Sciences*, vol. 119, no. 34, p. e2206069119, 2022.
[42] 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.
[43] 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.
[44] 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.