An algorithmic framework for the optimization of deep neural networks architectures and hyperparameters

Julie Keisler, El-Ghazali Talbi, Sandra Claudel, Gilles Cabriel

AN ALGORITHMIC FRAMEWORK FOR THE OPTIMIZATION OF DEEP NEURAL NETWORKS ARCHITECTURES AND HYPERPARAMETERS

Julie Keisler
EDF Lab Paris-Saclay
University of Lille & INRIA
julie.keisler@edf.fr

El-Ghazali Talbi
University of Lille & INRIA
el-ghazali.talbi@univ-lille.fr

Sandra Claudel
EDF Lab Paris-Saclay
sandra.claudel@edf.fr

Gilles Cabriel
EDF Lab Paris-Saclay
gilles.cabriel@edf.fr

ABSTRACT

In this paper, we propose an algorithmic framework to automatically generate efficient deep neural networks and optimize their associated hyperparameters. The framework is based on evolving directed acyclic graphs (DAGs), defining a more flexible search space than the existing ones in the literature. It allows mixtures of different classical operations: convolutions, recurrences and dense layers, but also more newfangled operations such as self-attention. Based on this search space we propose neighbourhood and evolution search operators to optimize both the architecture and hyper-parameters of our networks. These search operators can be used with any metaheuristic capable of handling mixed search spaces. We tested our algorithmic framework with an evolutionary algorithm on a time series prediction benchmark. The results demonstrate that our framework was able to find models outperforming the established baseline on numerous datasets.

Keywords Metaheuristics · Evolutionary Algorithm · AutoML · Neural Architecture Search · Hyperparameter optimization · Directed Acyclic Graphs · Time Series Forecasting

1 Introduction

With the recent successes of deep learning in many research fields, deep neural networks (DNN) optimization stimulates the growing interest of the scientific community [Talbi, 2021]. While each new learning task requires the handcrafted design of a new DNN, automated deep learning facilitates the creation of powerful DNNs. Interests are to give access to deep learning to less experienced people, to reduce the tedious tasks of managing many parameters to reach the optimal DNN, and finally, to go beyond what humans can design by creating non-intuitive DNNs that can ultimately prove to be more efficient.

Optimizing a DNN means automatically finding an optimal architecture for a given learning task: choosing the operations and the connections between those operations and the associated hyperparameters. The first task is also known as Neural Architecture Search [Elsken et al., 2019], also named NAS, and the second, as HyperParameters Optimization (HPO). Most works from the literature try to tackle only one of these two optimization problems. Many papers related to NAS [White et al., 2021, Loni et al., 2020b, Wang et al., 2019b, Sun et al., 2018b, Zhong, 2020] focus on designing optimal architectures for computer vision tasks with a lot of stacked convolution and pooling layers. Because each DNN training is time-consuming, researchers tried to reduce the search space by adding many constraints preventing from finding irrelevant architectures. It affects the flexibility of the designed search spaces and limits the hyperparameters optimization.

We introduce in this paper a new optimization framework for AutoML based on the evolution of Directed Acyclic Graphs (DAGs). The encoding and the search operators may be used with various deep learning and AutoML problems. We ran experiments on time series forecasting tasks and demonstrate on a large variety of datasets that our framework

can find DNNs which compete with or even outperform state-of-the-art forecasters. In summary, our contributions are as follows:

- The precise definition of a flexible and complete search space based on DAGs, for the optimization of DNN architectures and hyperparameters.
- The design of efficient neighbourhoods and variation operators for DAGs. With these operators, any meta-heuristic designed for a mixed and variable-size search space can be applied. In this paper, we investigate the use of evolutionary algorithms.
- The validation of the algorithmic framework on popular time series forecasting benchmarks [Godahewa et al., 2021]. We outperformed 13 statistical and machine learning models on 24 out of 40 datasets, proving the efficiency and robustness of our framework.

The paper is organized as follows: we review section 2, the literature on deep learning models for time series forecasting and AutoML. Section 3 defines our search space. Section 4 presents our neighbourhoods and variation operators within evolutionary algorithms. Section 5 details our experimental results obtained on popular time series forecasting benchmarks. Finally, section 6 gives a conclusion and introduces further research opportunities.

2 Related Work

2.1 Deep learning for time series forecasting

Time series forecasting has been studied for decades. The field has been dominated for a long time by statistical tools such as ARIMA, Exponential Smoothing (ES), or (S)ARIMAX, this last model allowing the use of exogenous variables. It now opens itself to deep learning models [Liu et al., 2021]. These new models recently achieved great performances on many datasets. Three main parts compose typical DNNs: an input layer, several hidden layers and an output layer. In this paper, we define a search space designed to search for the best-hidden layers, given a meta-architecture (see Figure 5), for a specific time series forecasting task. Next, we introduce the usual DNN layers considered in our search space.

The first layer type from our search space is the fully-connected layer, or Multi-Layer Perceptron (MLP). The input vector is multiplied by a weight matrix. Most architectures use such layers as simple building blocks for dimension matching, input embedding or output modelling. The N-Beats model is a well-known example of a DNN based on fully-connected layers for time series forecasting [Oreshkin et al., 2019].

The second layer type [LeCun et al., 2015] is the convolution layer (CNN). Inspired by the human brain's visual cortex, it has mainly been popularised for computer vision. The convolution layer uses a discrete convolution operator between the input data and a small matrix called a filter. The extracted features are local and time-invariant if the considered data are time series. Many architectures designed for time series forecasting are based on convolution layers such as WaveNet [Oord et al., 2016] and Temporal Convolution Networks [Lea et al., 2017].

The third layer type is the recurrent layer (RNN), specifically designed for sequential data processing, therefore, particularly suitable for time series. These layers scan the sequential data and keep information from the sequence past in memory to predict its future. A popular model based on RNN layers is the Seq2Seq network [Cho et al., 2014]. Two RNNs, an encoder and a decoder, are sequentially connected by a fixed-length vector. Various versions of the Seq2Seq model have been introduced in the literature, such as the DeepAR model [Salinas et al., 2020], which encompasses an RNN encoder in an autoregressive model. The major weakness of RNN layers is the modelling of long-term dynamics due to the vanishing gradient. Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) layers have been introduced [Hochreiter and Schmidhuber, 1997, Chung et al., 2014] to overcome this problem.

Finally, the layer type from our search space is the attention layer. The attention layer has been popularized within the deep learning community as part of Vaswani's transformer model [Vaswani et al., 2017]. The attention layer is more generic than the convolution. It can model the dependencies of each element from the input sequence with all the others. In the vanilla transformer [Vaswani et al., 2017], the attention layer does not factor the relative distance between inputs in its modelling but rather the element's absolute position in the sequence. The Transformer-XL [Dai et al., 2019], a transformer variant created to tackle long-term dependencies tasks, introduces a self-attention version with relative positions. Cordonnier et al. [2019] used this new attention formulation to show that, under a specific configuration of parameters, the attention layers could be trained as convolution layers. Within our search space, we chose this last formulation of attention, with the relative positions.

The three first layers (i.e. MLP, CNN, RNN) were frequently mixed into DNN architectures. Sequential and parallel combinations of convolution, recurrent and fully connected layers often compose state-of-the-art DNN models for time series forecasting. Layer diversity enables the extraction of different and complementary features from input data to

allow a better prediction. Some recent DNN models introduce transformers into hybrid DNNs. In Lim et al. [2021], the authors developed the Temporal Fusion Transformer, a hybrid model stacking transformer layers on top of an RNN layer. With this in mind, we built a flexible search space which generalizes hybrid DNN models including MLPs, CNNs, RNNs and transformers.

2.2 Search spaces for automated deep learning

Designing an efficient DNN for a given task requires choosing an architecture and tuning its many hyperparameters. It is a difficult fastidious, and time-consuming optimization task. Moreover, it requires expertise and restricts the discovery of new DNNs to what humans can design. Research related to the automatic design and optimization of DNNs has therefore risen this last decade [Talbi, 2021]. The first challenge with automatic deep learning (AutoDL), and more specifically the neural architecture search (NAS), is the search space design. If the solution encoding is too broad and allows too many architectures, we might need to evaluate many architectures to explore the search space. However, training many DNNs would require considerable computing time and become unfeasible. On the contrary, if the search space is too small, we might miss promising solutions. Besides, encoding of DNNs defining the search space should follow some rules [Talbi, 2021]:

- Completeness: all candidate DNNs solutions should be encoded in the search space.
- Connexity: a path should always be possible between two encoded DNNs in the search space.
- Efficiency: the encoding should be easy to manipulate by the search operators (i.e. neighbourhoods, variation operators) of the search strategy.
- Constraint handling: the encoding should facilitate the handling of the various constraints to generate feasible DNNs.

A complete classification of encoding strategies for NAS is presented in Talbi [2021] and reproduced in Figure 1. We can discriminate between direct and indirect encodings. With direct strategies, the DNNs are completely defined by the encoding, while indirect strategies need a decoder to find the architecture back. Amongst direct strategies, one can discriminate between two categories: flat and hierarchical encodings. In flat encodings, all layers are individually encoded [Loni et al., 2020a, Sun et al., 2018a, Wang et al., 2018, 2019a]. The global architecture can be a single chain, with each layer having a single input and a single output, which is called chain structured [Assunção et al., 2018], but more complex patterns such as multiple outputs, skip connections, have been introduced in the extended flat DNNs encoding [Chen et al., 2021]. For hierarchical encodings, they are bundled in blocks [Pham et al., 2018, Shu et al., 2019, Liu et al., 2017, Zhang et al., 2019]. If the optimization is made on the sequencing of the blocks, with an already chosen content, this is referred to as inner-level fixed [Camero et al., 2021, White et al., 2021]. If the optimization is made on the blocks' content with a fixed sequencing, it is called outer level fixed. A joint optimization with no level fixed is also an option [Liu et al., 2019]. Regarding the indirect strategies, one popular encoding is the one-shot architecture [Bender et al., 2018, Brock et al., 2017]. One single large network resuming all candidates from the search space is trained. Then the architectures are found by pruning some branches. Only the best promising architectures are retrained from scratch.

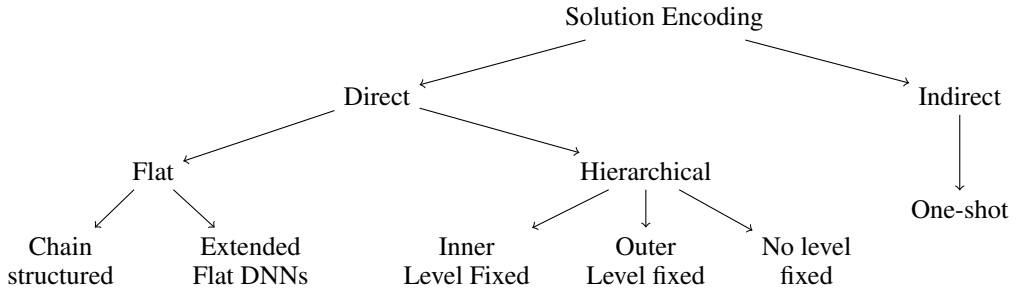


Figure 1: Classification of encoding strategies for NAS [Talbi, 2021].

Our search space can be categorized as a direct and extended flat encoding. Each layer is individually encoded by our search space. It is more flexible than the search spaces designed in literature. First, we tackle both the optimization of the architecture and the hyperparameters. Second, the diversity of candidate DNNs is much broader than what can be found in the literature. We allow a combination of recurrent, convolution, attention-based and fully connected layers, leading to innovative, original yet well-performing DNNs. To our knowledge, this encoding has never been investigated in the literature.

2.3 AutoML for time series forecasting

The automated design of DNNs called Automated Deep Learning (AutoDL), belongs to a larger field [Hutter et al., 2019] called Automated Machine Learning (AutoML). AutoML aims to automatically design well-performing machine learning pipelines, for a given task. Works on model optimization for time series forecasting mainly focused on AutoML rather than AutoDL [Alsharef et al., 2022]. The optimization can be performed at several levels: input features selection, extraction and engineering, model selection and hyperparameters tuning. Initial research works used to focus on one of these subproblems, while more recent works offer complete optimization pipelines.

The first subproblems, input features selection, extraction and engineering, are specific to our learning task: time series forecasting. This tedious task can significantly improve the prediction scores by giving the model relevant information about the data. Methods to select the features are among computing the importance of each feature on the results or using statistical tools on the signals to extract relevant information. Next, the model selection aims at choosing among a set of diverse machine learning models the best-performing one on a given task. Often, the models are trained separately, and the best model is chosen. In general, the selected model has many hyperparameters, such as the number of hidden layers, activation function or learning rate. Their optimization usually allows for improving the performance of the model.

Nowadays, many research works implement complete optimization pipelines combining those subproblems for time series forecasting. The Time Series Pipeline Optimization framework [Dahl, 2020], is based on an evolutionary algorithm to automatically find the right features thanks to input signal analysis, then the model and its related hyperparameters. AutoAI-TS [Shah et al., 2021] is also a complete optimization pipeline, with model selection performed among a wide assortment of models: statistical models, machine learning, deep learning models and hybrids models. Finally, the framework Auto-Pytorch-TS [Deng et al., 2022] is specific to deep learning models optimization for time series forecasting. The framework uses Bayesian optimization with multi-fidelity optimization.

Except for AutoPytorch-TS, cited works covering the entire optimization pipeline for time series do not deepen model optimization and only perform model selection and hyperparameters optimization. However, time series data becomes more complex, and there is a growing need for more sophisticated and data-specific DNNs. In this work, we only tackle the model selection and hyperparameters optimization parts of the pipeline. We made this choice to show the effectiveness of our framework for designing better DNNs. If we had implemented feature selection, it would have been harder to determine whether the superiority of our results came from the input features pool or the model itself. We discuss this further in Section 5.

3 Search space definition

The development of a complete optimization framework for AutoDL needs the definition of the search space, the objective function and the search algorithm. In this section, the handled optimization problem is formulated. Then, the search space and its characteristics are detailed.

3.1 Optimization problem formulation

Our optimization problem consists in finding the best possible DNN for a given time series forecasting problem. To do so, we introduce an ensemble Ω representing our search space, which contains all considered DNNs. We then consider our time series dataset $\mathscr{D}$. For any subset $\mathscr{D}_0 = (X_0, Y_0)$, we define the forecast error ℓ as:

$$\begin{aligned} \ell \colon \Omega \times \mathscr{D} &\to \mathbb{R} \\ f \times \mathscr{D}_0 &\mapsto \ell\big(f(\mathscr{D}_0)\big) = \ell\big(Y_0, f(X_0)\big). \end{aligned}$$

The explicit formula for ℓ will be given later in the paper. Each element f from Ω is a DNN defined as an operator parameterized by three parameters. First, its architecture $\alpha \in \mathscr{A}$. $\mathscr{A}$ is the search space of all considered architectures and will be detailed in Subsection 3.2. Given the DNN architecture α, the DNN is then parameterized by its hyperparameters $\lambda \in \Lambda(\alpha)$, with $\Lambda(\alpha)$ the search space of the hyperparameters induced by the architecture α and defined Subsection 3.3. Finally, α and λ generate an ensemble of possible weights $\Theta(\alpha, \lambda)$, from which the DNN optimal weights θ are found by gradient descent when training the model. The architecture α and the hyperparameters λ are optimized by our framework.

We consider the multivariate time series forecasting task. Our dataset $\mathscr{D} = (X, Y)$ is composed of a target variable $Y = \{\mathbf{y}_t\}_{t=1}^{T}$, with $\mathbf{y}_t \in \mathbb{R}^N$ and a set of explanatory variables (features) $X = \{\mathbf{x}_t\}_{t=1}^{T}$, with $\mathbf{x}_t \in \mathbb{R}^{F_1 \times F_2}$. The size

of the target Y at each time step is T and F_1, F_2 are the shapes of the input variable X at each time step. We choose to represent $\mathbf{x}_t$ by a matrix to extend our framework's scope, but it can equally be defined as a vector by taking $F_2 = 1$. The framework can be applied to univariate signals by taking $T = 1$. We partition our time indexes into three groups of successive time steps and split accordingly $\mathcal{D}$ into three datasets: $\mathcal{D}_{train}$, $\mathcal{D}_{valid}$ and $\mathcal{D}_{test}$.

After choosing an architecture α and a set of hyperparamaters λ, we build the DNN $f^{\alpha,\lambda}$ and use $\mathcal{D}_{train}$ to train $f^{\alpha,\lambda}$ and optimize its weights θ by stochastic gradient descent:

$$\hat{\theta} \in \underset{\theta \in \Theta(\alpha,\lambda)}{\arg\min} \big(\ell(f_\theta^{\alpha,\lambda}, \mathcal{D}_{train})\big).$$

The forecast error of the DNN parameterized by $\hat{\theta}$ on $\mathcal{D}_{valid}$ is used to assess the performance of the selected α and λ. The best architecture and hyperparameters are optimized by solving:

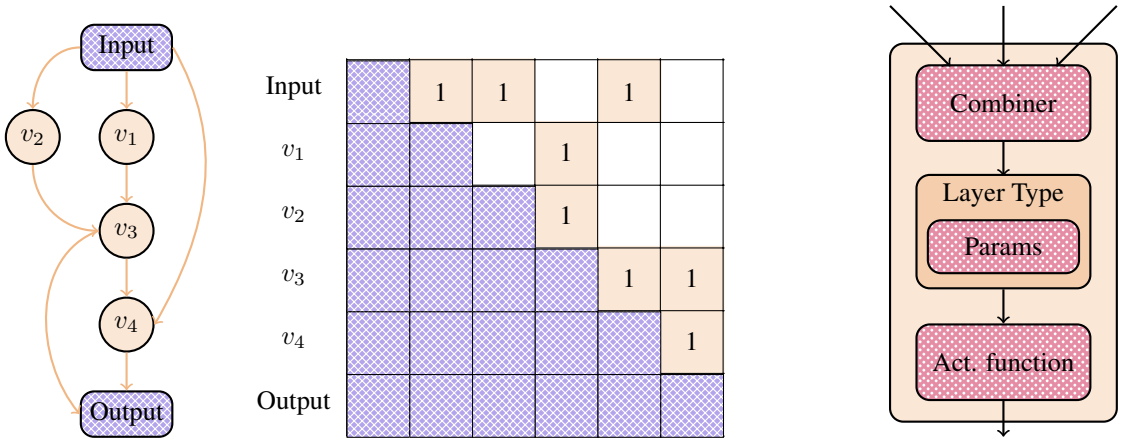
$$(\hat{\alpha}, \hat{\lambda}) \in \underset{\alpha \in \mathcal{A}}{\arg\min} \Big(\underset{\lambda \in \Lambda(\alpha)}{\arg\min} \big(\ell(f_{\hat{\theta}}^{\alpha,\lambda}, \mathcal{D}_{valid})\big)\Big).$$

The function $(\alpha, \lambda) \mapsto \ell(f_{\hat{\theta}}^{\alpha,\lambda}, \mathcal{D}_{valid})$ corresponds to the objective function of our algorithmic framework. We finally will evaluate the performance of our algorithmic framework by computing the forecast error on $\mathcal{D}_{test}$ using the DNN with the best architecture, hyperparameters and weights:

$$\ell(f_{\hat{\theta}}^{\hat{\alpha},\hat{\lambda}}, \mathcal{D}_{test}).$$

In practice, the second equation optimizing α and λ can be solved separately or jointly. If we fix λ for each α, the optimization is made only on the architecture and is referred to as Neural Architecture Search (NAS). If α is fixed, then the optimization is only made on the model hyperparameters and is referred to as HyperParameters Optimization (HPO). Our algorithmic framework allows us to fix α or λ during parts of the optimization to perform a hierarchical optimization: ordering optimisation sequences during which only the architecture is optimised, and others during which only the hyperparameters are optimised. In the following, we will describe our search space $\Omega = (\mathcal{A} \times \{\Lambda(\alpha), \alpha \in \mathcal{A}\})$.

3.2 Architecture Search Space



(a) Architecture (b) Adjacency matrix representation (c) Inside node v_i

Figure 2: DNN encoding as a directed acyclic graph (DAG). The elements in blue (crosshatch) are fixed by the framework, the architecture elements from α are displayed in beige and the hyperparameters λ are in pink (dots).

First, we define our architecture search space $\mathcal{A}$. We propose to model a DNN by a Directed Acyclic Graph (DAG) with a single input and output [Fiore and Devesas Campos, 2013]. A DAG $\Gamma = (\mathcal{V}, \mathcal{E})$ is defined by its nodes (or vertices) set $\mathcal{V} = \{v_1, ..., v_n\}$ and its edges set $\mathcal{E} \subseteq \{(v_i, v_j) | v_i, v_j \in \mathcal{V}\}$. Each node v represents a DNN layer as defined in Subsection 2.1, such as a convolution, a recurrence, or a matrix product. To eliminate isolated nodes, we impose each node to be connected by a path to the input and the output. The graph acyclicity implies a partial ordering of the nodes. If a path exists from the node v_a to a node v_b, then we can define a relation order between them: $v_a < v_b$. Acyclicity

prevents the existence of a path from v_b to v_a. However, this relation order is not total. When dealing with symmetric graphs where all nodes are not connected, several nodes' ordering may be valid for the same graph. For example in Figure 2a, the orderings $v_1 > v_2$ and $v_2 > v_1$ are both valid.

Hence, a DAG Γ is represented by a sorted list $\mathcal{L}$, such that $|\mathcal{L}| = m$, containing the graph nodes, and its adjacency matrix $M \in \mathbb{R}^{m \times m}$ [Zhang et al., 2019]. The matrix M is built such that: $M(i,j) = 1 \Leftrightarrow (v_i, v_j) \in \mathcal{E}$. Because of the graph's acyclicity, the matrix is upper triangular with its diagonal filled with zeros. The input node has no incoming connection, and the output node has no outcoming connection, meaning $\sum_{i=1}^{m} M_{i,1} = 0$ and $\sum_{j=1}^{m} M_{m,j} = 0$. Besides, the input is necessarily connected to the first node and the last node to the output for any graph, enforcing: $M_{1,2} = 1$ and $M_{m-1,m} = 1$. As isolated nodes do not exist in the graph, we need at least a non-zero value on every row and column, except for the first column and last row. We can express this property as: $\forall i < m : \sum_{j=i+1}^{m} M_{i,j} > 0$ and $\forall j > 1 : \sum_{i=j+1}^{m} M_{i,j} > 0$. Finally, the ordering of the partial nodes does not allow a bijective encoding: several matrices M may encode the same DAG.

To summarize, we have $\mathcal{A} = \{\Gamma = (\mathcal{V}, \mathcal{E}) = (\mathcal{L}, M)\}$. The graphs Γ are parameterized by their size m which is not equal for all graphs. As we will see in Section 4.1 the DNNs size may vary during the optimization framework.

3.3 Hyperparameters Search Space

For any fixed architecture $\alpha \in \mathcal{A}$, let's define our hyperparameters search space induced by $\alpha : \Lambda(\alpha)$. As mentioned above, the DAG nodes represent the DNN hidden layers. A set of hyperparameters λ, also called a graph node, is composed of a combiner, a layer operation and an activation function (see Figure 2c). Each layer operation is associated with a specific set of parameters, like output or hidden dimensions, convolution kernel size or dropout rate. We provide in Appendix A a table with all available layer types and their associated parameters. The hyperparameters search space $\Lambda(\alpha)$ is made of sets λ composed with a combiner, the layer's parameters and the activation function.

First, we need a combiner as each node can receive an arbitrary number of input connections. The parents' latent representations should be combined before being fed to the layer type. Taking inspiration from the Google Brain Team Evolved Transformer [So et al., 2019], we propose three types of combiners: element-wise addition, element-wise multiplication and concatenation. The input vectors may have different channel numbers and the combiner needs to level them. This issue is rarely mentioned in the literature, where authors prefer to keep a fixed channel number [Liu et al., 2018]. In the general case, for element-wise combiners, the combiner output channel matches the maximum channel number of latent representation. We apply zero-padding on the smaller inputs. For the concatenation combiner, we consider the sum of the channel number of each input. Some layer types, for instance, the pooling and the convolution operators, have kernels. Their calculation requires that the number of channels of the input vector is larger than this kernel. In these cases, we also perform zero-padding after the combiner to ensure that we have the minimum number of channels required.

When building the DNN, we dimension asynchronously each layer operation. We first compute the layer operation input shape according to the input vectors and the combiner. After building the operation we compute its output shape for the next layer. Finally, the node's remaining part is the activation function. We choose this last parameter among a large set detailed in Appendix A. To summarize, we define every node as the sequence of combiner $\rightarrow$ layer type $\rightarrow$ activation function. In our search space $\Lambda(\alpha)$, the nodes are encoded by arrays containing the combiner name, the layer type name, the value of each layer operation's parameters and finally, the activation function name. The set $\mathcal{L}$ mentioned in the previous section, which contains the nodes, is then a variable-length list containing the arrays representing each node.

4 Search algorithm

The search space $\Omega = (\mathcal{A} \times \{\Lambda(\alpha), \alpha \in \mathcal{A}\})$ that we defined in the previous Section is a mixed and variable space: it contains integers, float, and categorical values, and the dimension of its elements, the DNNs, is not fixed. We need to design a search algorithm able to efficiently navigate through this search space. While several metaheuristics can solve mixed and variable-size optimization problems [Talbi, 2023], we chose to start with an evolutionary algorithm. For the manipulation of directed acyclic graphs, this metaheuristic was the most intuitive for us. It has been used in other domains, for example on graphs representing logic circuits [Aguirre and Coello Coello, 2003]. The design of other metaheuristics in our search space and their comparison with the evolutionary algorithm are left to future work.

4.1 Evolutionary algorithm design

Evolutionary algorithms represent popular metaheuristics which are well adapted to solve mixed and variable-space optimization algorithms [Talbi, 2023]. They have been widely used for the automatic design of DNNs [Li et al., 2022].

The idea is to evolve a randomly generated population of DAGs to converge towards an optimal DNN. An optimal solution should be a DNN with a small error on our forecasting task. The designed metaheuristic is based on several search operators: selection, mutation, crossover and replacement. The initial population is randomly generated. We then evolve this population during G generations. At the beginning of each new generation g, we build the new population starting from the scores obtained by the individuals from the previous generation. We use the tournament selection to pick the best individual among randomly drawn sub-groups. Part of the individuals from this new population comes from the tournament selection, while we randomly draw the remaining ones. The randomly drawn individuals ensure the algorithm to not be dependent on the initial population. Afterwards, the DAGs composing the new population are transformed using variation operators such as crossover and mutation, described thereafter. The generated individuals are called offsprings. After their evaluation, the worst offsprings are replaced by the best individuals from the previous generation. Therefore, the best individuals are kept in memory and used for evolution during the entire process. The replacement rate should stay small to prevent a premature convergence of the algorithm toward a local optimum. The complete framework is shown in Figure 3.

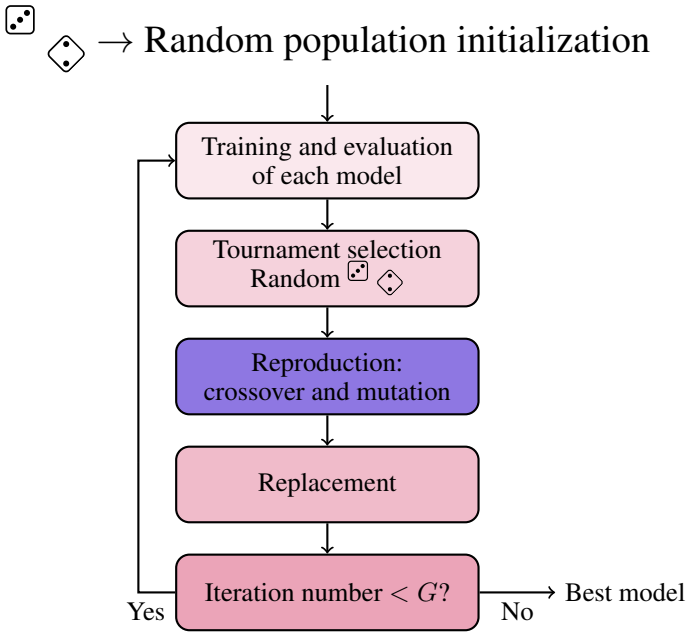


Figure 3: Evolutionary algorithm framework.

To design the algorithm evolution operators, we split them into two categories: hyperparameters specific operators and architecture operators. The idea is to allow a sequential or joint optimization of the hyperparameters and the architecture. The involved layer types do not have the same hyperparameters. Thus, drawing a new layer means modifying all its parameters and one can lose the optimization made on the previous layer type. Using sequential optimization, the algorithm can first find well-performing architectures and layers types during the architecture search and then fine-tune the found DNNs during the hyperparameters search.

4.2 Architecture evolution

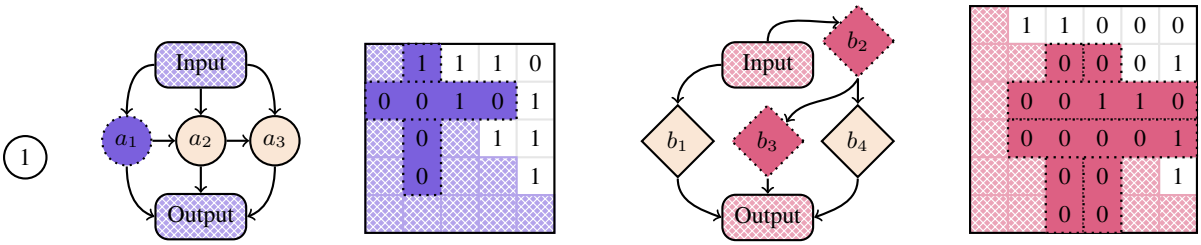
In this section, we introduce the architecture-specific search operators. By architecture, we mean the search space $\mathcal{A}$ defined above: the node's operations and the edges between them. The mutation operator is made of several simple operations inspired by the transformations used to compute the Graph Edit Distance [Abu-Aisheh et al., 2015]: insertion, deletion and substitution of both nodes and edges. Given a graph $\Gamma = (\mathcal{L}, M)$, the mutation operator will draw the set $\mathcal{L}' \subseteq \mathcal{L}$ and apply a transformation to each node of $\mathcal{L}'$. Let's have $v_i \in \mathcal{L}'$ the node that will be transformed:

- **Node insertion:** we draw a new node with its combiner, operation and activation function. We insert the new node in our graph at the position $i + 1$. We draw its incoming and outgoing edges by verifying that we do not generate an isolated node.
- **Node deletion:** we delete the node v_i. In the case where it generates other isolated nodes, we draw new edges.
- **Parents modification:** we modify the incoming edges for v_i and make sure we always have at least one.

- **Children modification:** we modify the outgoing edges for v_i and make sure we always have at least one.
- **Node modification:** we draw the new content of v_i, the new combiner, the operation and/or the activation function.

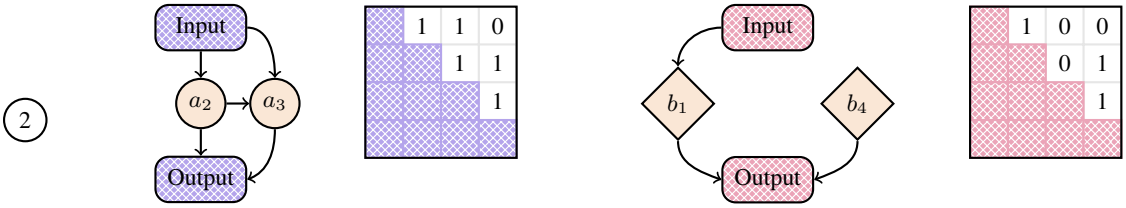
The crossover idea is to inherit patterns belonging to both parents. The primary crossover operator applies on two arrays and swaps two subparts of those arrays. We draw two subgraphs from our parents, which can be of different sizes, and we swap them. This transformation has an impact on edges. To reconstruct the offsprings, we tried to preserve at most the original edges from the parents and the swapped subgraphs. An illustration of the crossover can be found in Figure 4.

Figure 4: Crossover operator illustration.



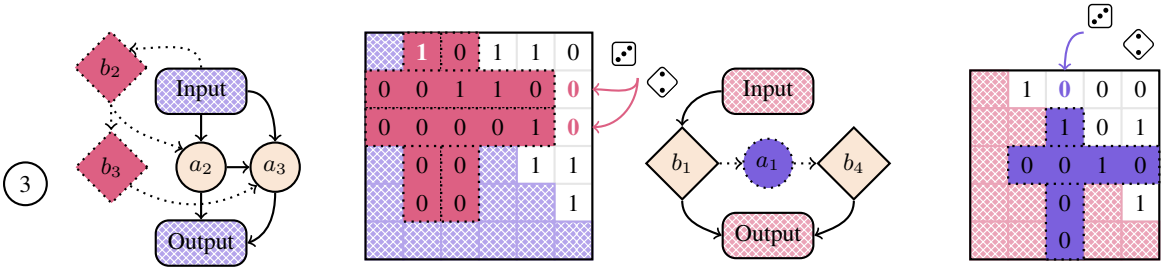
Parent 1 Parent 2

(a) 1st step: we select the layers that would be exchanged with dotted lines)



Offspring 1 Offspring 2

(b) 2nd step: we remove the selected layers from the children graphs



Offspring 1 Offspring 2

(c) 3rd step: we insert the layers and reconstruct the edges.

4.3 Hyperparameters evolution

One of the architecture mutations consists in disturbing the node content. In this case, the node content is modified, including the operation. A new set of hyperparameters is then drawn. To refine this search, we defined specific mutations for the search space $\Lambda(\alpha)$. In the hyperparameters case, edges and nodes number are not affected. As for architecture-specific mutation, the operator will draw the set $\mathcal{L}' \subseteq \mathcal{L}$ and apply a transformation on each node of $\mathcal{L}'$. For each node v_i from $\mathcal{L}'$, we draw h_i hyperparameters, which will be modified by a neighbouring value. The hyperparameters in our search space belong to three categories:

- **Categorical values:** the new value is randomly drawn among the set of possibilities deprived of the actual value. For instance, the activation functions, combiners, and recurrence types (LSTM/GRU) belong to this type of categorical variable.
- **Integers:** we select the neighbours inside a discrete interval around the actual value. For instance, it has been applied to convolution kernel size and output dimension.
- **Float:** we select the neighbours inside a continuous interval around the actual value. Such a neighbourhood has been defined for instance to the dropout rate.

5 Experimental study

5.1 Experimental protocol

We evaluated our optimization algorithm framework on the established benchmark of Monash Time Series Forecasting Repository [Godahewa et al., 2021]. For these experiments, we configured our algorithm to have a population of 40 individuals and a total of 100 generations. We investigated a sequential optimization of the architecture and the hyperparameters. We alternate at a certain generation frequency between two scopes: search operators applied to the architecture α with λ fixed, and search operators applied to the hyperparameters λ with α fixed. Thus, the architecture-centred sections of the optimization will diversify the population DNNs, while the hyperparameter-centred parts will perform a finer optimization of the obtained DNNs. We ran our experiments on 5 cluster nodes, each equipped with 4 Tesla V100 SXM2 32GB GPUs, using Pytorch 1.11.0 and Cuda 10.2. The experiments were all finished in less than 72 hours.

The Monash time series forecasting archive is a benchmark containing more than 40 datasets and the results of 13 forecasting models on each of these prediction tasks [Godahewa et al., 2021]. The time series are of different kinds and have variable distributions. More information on each dataset from the archive is available B. It allows us to test our framework generalization and robustness abilities.

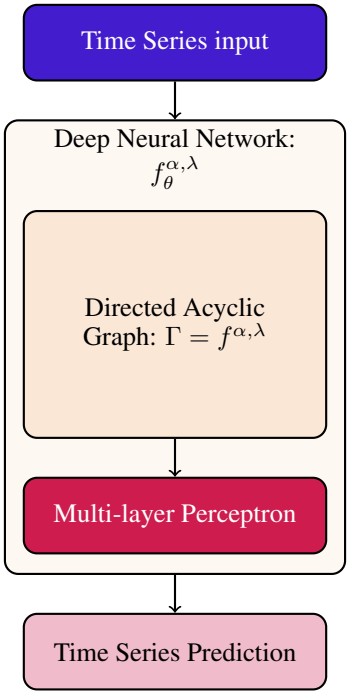


Figure 5: Meta model for Monash time series datasets.

The paper's authors accompanied their dataset with a GitHub allowing them to directly and very easily compare different statistical, machine learning and deep learning models on the forecast of different time series. We followed the indications to integrate new deep learning models, and only changed the models' core. We kept their data preparation functions, data loaders, training parameters (number of epochs, batch size), as well as the training and evaluation functions, to have the most accurate comparison possible. For each dataset, we also took over the configurations of

the directory, notably the prediction horizons and lags. Figure 5 represents our meta-model which was used to replace the repository's models. The Multi-layer Perceptron at the end of the model is used to retrieve the time series output dimension, as the number of channels may vary within the Directed Acyclic Graph.

We compared our results with the benchmark models, which are composed of statistical models, machine learning and deep learning models. The metric used to evaluate the models' performance, our forecast error ℓ, is the Mean Absolute Scaled Error (MASE), an absolute mean error divided by the average difference between two consecutive time steps [Hyndman and Koehler, 2006]. Given a time series $Y = (\mathbf{y}_1, ..., \mathbf{y}_n)$ and the predictions $\hat{Y} = (\hat{\mathbf{y}}_1, ..., \hat{\mathbf{y}}_n)$, the MASE can be defined as:

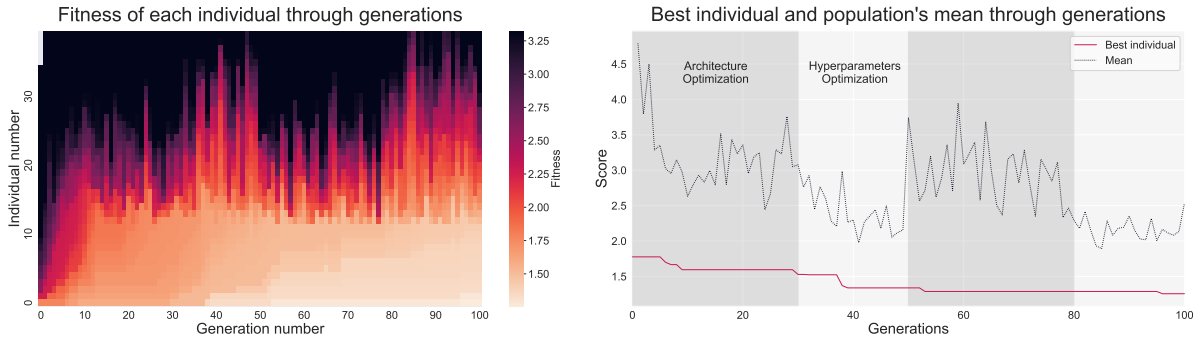
$$\text{MASE}(Y, \hat{Y}) = \frac{n-1}{n} \times \frac{\sum_{t=1}^{n} |\mathbf{y}_t - \hat{\mathbf{y}}_t|}{\sum_{t=2}^{n} |\mathbf{y}_t - \mathbf{y}_{t-1}|}.$$

In our case, for $f \in \Omega$, $\mathcal{D}_0 = (X_0, Y_0) \subseteq \mathcal{D}$, we have $\ell(Y_0, f(X_0) = \text{MASE}\big(Y_0, f(X_0)\big)$.

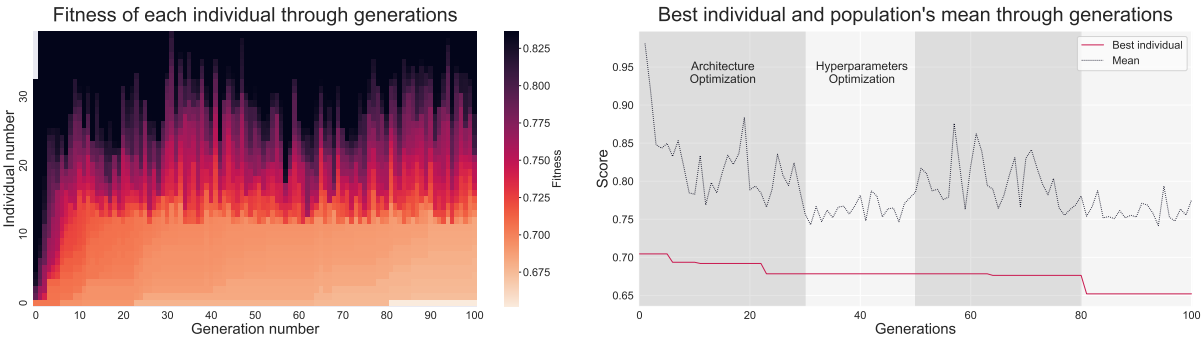
Table 1: Mean MASE for each dataset, we only reported: the best MASE for statistical models, machine learning models and our optimization framework results. Statistical models: SES, Theta, TBATS, ETS, ARIMA. Machine Learning models: PR, CatBoost, FFNN, DeepAR, N-Beats, WaveNet, Transformer, Informer.

Name / **Dataset**	**Models** *Stat. models*	*ML/DL models*	*Our framework*
Aus. elec	1.174	**0.705**	0.893
Births	1.453	1.537	**1.233**
Bitcoin	2.718	**2.664**	4.432
Carparts	0.897	0.746	**0.744**
Covid deaths	5.326	5.459	**4.535**
Dominick	0.582	0.705	**0.510**
Elec. hourly	3.690	**1.606**	1.652
Elec. weekly	1.174	0.705	**0.652**
Fred MD	**0.468**	0.601	0.489
Hospital	0.761	0.769	**0.751**
KDD	1.394	1.185	**1.161**
Kaggle weekly	0.622	0.628	**0.561**
M1 monthly	**1.074**	1.123	1.081
M1 quart.	**1.658**	1.700	1.683
M1 yearly	**3.499**	4.355	3.732
M3 monthly	**0.861**	0.934	0.926
M3 other	**1.814**	2.127	2.227
M3 quart.	1.174	1.182	**1.099**
M3 yearly	**2.774**	2.961	2.800
M4 daily	1.153	1.141	**1.066**
M4 hourly	2.663	1.662	**1.256**
M4 monthly	**0.948**	1.026	0.993
M4 quart.	**1.161**	1.239	1.198
M4 weekly	0.504	0.453	**0.430**
NN5 daily	**0.858**	0.916	0.898
NN5 weekly	0.872	0.808	**0.739**
Pedestrians	0.957	0.247	**0.222**
Rideshare	1.530	2.908	**1.410**
Saugeen	1.425	1.411	**1.296**
Solar 10mn	**1.034**	1.450	1.426
Solar weekly	0.848	0.574	**0.511**
Sunspot	0.067	0.003	**0.002**
Temp. rain	1.174	0.687	**0.686**
Tourism monthly	1.526	**1.409**	1.453
Tourism quart.	1.592	1.475	**1.469**
Tourism yearly	3.015	2.977	**2.690**
Traffic hourly	1.922	0.821	**0.729**
Traffic weekly	1.116	1.094	**1.030**
Vehicle trips	1.224	**1.176**	1.685
Weather	0.677	0.631	**0.614**
Total bests	11	5	24

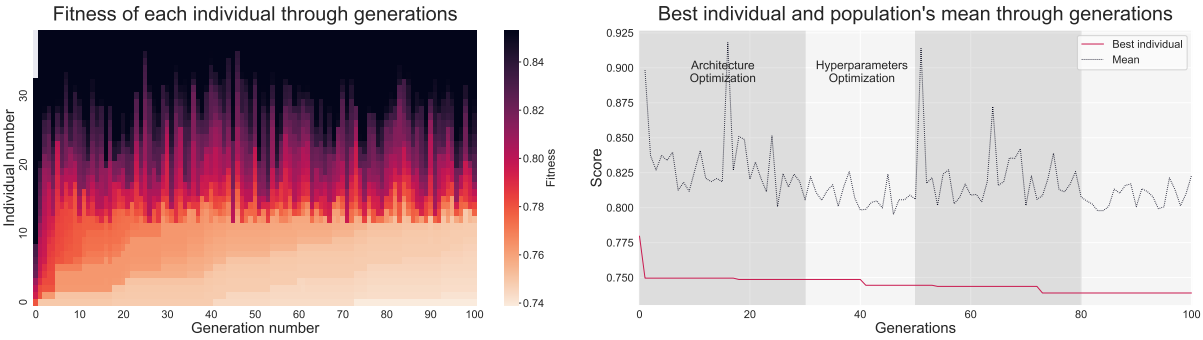
The results are reported in Table 1. Our model succeeded in outperforming the best baseline for 24 out of 40 datasets. For the remaining 16 datasets, our framework obtained errors close to the best baseline. It is worth noting that our model outperformed the machine learning and deep learning models from the benchmark for 34 out of 40 datasets and was among the top 3 models for 34 out of 40 datasets. In Figure 6 we show the convergence of the evolutionary algorithm. At the right positions, we displayed the mean loss for each generation and the best individual loss. On the mean loss for each generation curve, we can identify the phases where the architecture is optimized, where we have more variability and the phases where the hyperparameters are optimized, with less variability. On the left positions, we represented heatmaps with the loss for each individual at each generation. The best individual is not over-represented in the population before the optimization ends.



(a) M4 Hourly dataset, best MASE: 1.288



(b) Electricity weekly dataset, best MASE: 0.652



(c) NN5 weekly dataset, best MASE: 0.739

Figure 6: Evolutionary algorithm convergence. Left: heatmap with the loss for each individual for every generation. Right: population mean loss and best individual's loss through generations. Darker grey backgrounds represent generations during which the architecture is optimized, and lighter grey backgrounds represent generations during which the hyperparameters are optimized.

5.2 Best models analysis

In the AutoDL literature, few efforts are usually made to analyze the generated DNNs. In Shu and Cai [2019] the authors established that architectures with wide and shallow cell structures are favoured by the NAS algorithms and that

they suffer from poor generalization performance. We can rightfully ask ourselves about the efficiency of our framework and some of these questions may be answered thanks to a light models analysis. By the end of this section, we will try to answer some inquiries about our framework outputs. To answer those questions we defined some structural indicators, and we computed them in Table 2 for the best model for each dataset from Godahewa et al. [2021]:

- ***Nodes***: it represents the number of nodes (i.e. operations) in the graph.
- ***Width***: it represents the network width which can be defined as the maximum of incoming or outgoing edges to all nodes within the graph.
- ***Depth***: it defines the network depth which corresponds to the size of the longest path in the graph.
- ***Dim***: it is the maximum channel dimension, relative to the number of input and output channels (ratio). It indicates how complex the latent spaces from the neural network might become, compared to the dataset complexity.
- ***Edges***: it represents the number of edges, relative to the number of nodes in the graph. It indicates how complex the graph can be and how sparse the adjacency matrix is.
- The last 7 indicators correspond to the number of the appearance of each layer type within the DNN.

Do our framework always converge to complex models, or is it able to find simple DNNs?

From Table 2 and Figure 7a, one can see that we have multiple simple graphs with only two layers. Knowing that the last feed-forward layer is enforced by our meta-model (see Figure 5), our DAG is only composed of one layer. Another indicator of this simplicity is the percentage of feed-forward layers found in the best models. 41% of the layers are feed-forward according to the table 2 although our search space offers more complex layers such as convolution, recurrence or attention layers less frequently picked. This proves that even without regularization penalties, our algorithmic framework does not systematically search for over-complicated models.

Do our algorithmic framework always converge to similar architectures for different datasets?

The framework is meanwhile able to find complex models as in Figure 7b, which partially answers our question. The indicators in Table 2 suggest that we found models with various numbers of nodes, from 2 to 10 and with diverse edge densities. The best model for the electricity hourly dataset (see Figure 7d) has an average of 1.5 incoming or outgoing edges by node whereas the best model for the electricity weekly dataset has an average of 3 incoming or outgoing edges by nodes. This is true even within performing graphs for the same dataset. In Figure 8 we displayed two different graphs having similar (and good) performance on the Dominick dataset. Both graphs do not have the same number of nodes and different architectures, the first one being a deep sparse graph while the other is wider with a lot of edges.

What is the diversity of the layer types within the best models?

The graphs from Figure 8 have also quite different layer types. If both are mainly based on identity, pooling and feed-forward operations, the second graph introduces convolution and dropout layers. In general, to answer this question, the fully-connected layers seem to dominate all layer types, as it represents 41% of the chosen layers in the best models (see Table 2). The identity layer is also more frequently picked. It is finally interesting to notice that all other layers are selected at the same frequency.

Are the best models still "deep" neural networks or are they wide and shallow as stated in Shu and Cai [2019]?

To answer this question, the observations from Shu and Cai [2019] also apply to our results. Our models are often almost as wide as they are deep. This observation needs more investigation as one of the reasons mentioned in the paper: the premature evaluation of architecture before full convergence does not apply here. Our models are smaller than the ones studied in the paper and thus converge faster, even when they are a bit deeper.

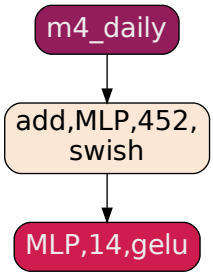
A last remark is related to the latent spaces generated by our models. Except for a few models, our models tend to always generate bigger latent space than the number of input and output channels. On average, the maximum size of the latent representation within a network is 17 times bigger than the input and/or output channels number.

5.3 Nondeterminism and instability of DNNs

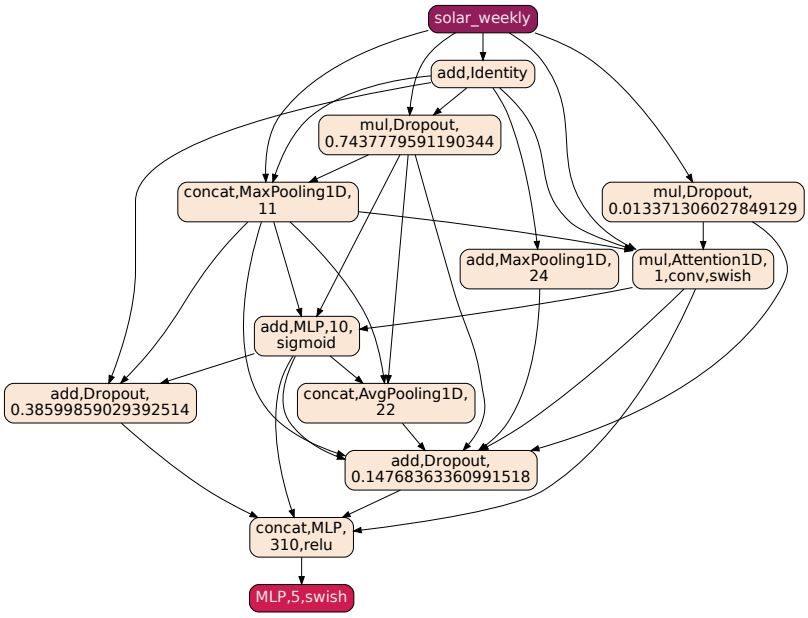
An often overlooked robustness challenge with DNN optimization is their uncertainty in performance [Summers and Dinneen, 2021]. A unique model with a fixed architecture and set of hyperparameters can produce a large variety of

Table 2: Structural indicators of the best model for each dataset found by our algorithmic framework.

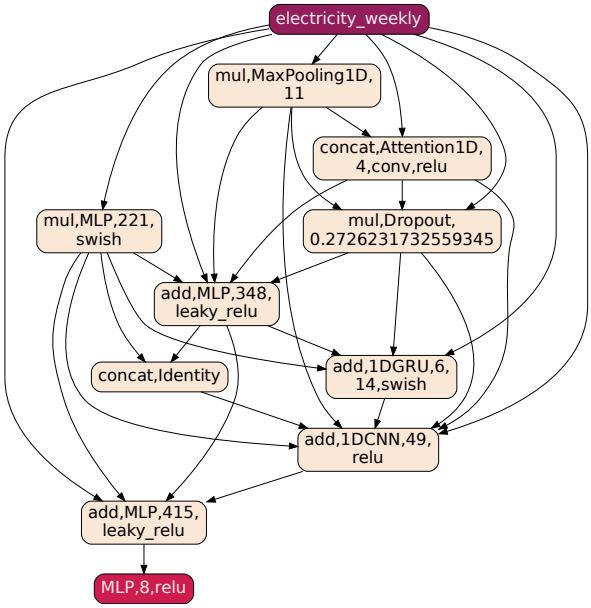
Name Dataset	**Structural Features**											
						Number of Nodes by layer types						
	nodes	***width***	***depth***	***dim***	***edges***	***MLP***	***Att***	***CNN***	***RNN***	***Drop***	***Id***	***Pool***
Aus. elec	3	2	3	1.02	1.33	2	0	0	0	0	1	0
Births	10	7	8	16.26	3.0	3	1	1	1	0	3	1
Bitcoin	9	5	7	14.2	2.11	3	0	1	2	2	1	0
Carparts	6	4	6	8.7	2.0	3	1	0	0	1	0	1
Dominick	8	6	6	40.4	2.22	5	0	1	0	1	1	1
Elec. hourly	6	3	6	2.88	1.5	4	0	0	0	0	0	2
Elec. weekly	10	8	8	8.75	3.0	4	1	1	1	1	1	1
Fred MD	8	6	7	12.0	2.38	2	2	0	0	2	1	1
Hospital	9	6	6	20.0	2.2	2	1	2	0	0	2	2
KDD	4	3	4	2.29	1.75	4	0	0	0	0	0	0
Kaggle weekly	5	3	5	25.3	1.6	4	0	0	0	0	0	1
M1 monthly	4	3	4	16.1	1.75	2	1	0	1	0	0	0
M1 quart.	10	6	8	56.38	3.0	5	1	0	1	0	3	0
M1 yearly	7	4	5	52.3	1.71	3	1	1	0	1	0	1
M3 monthly	8	5	7	11.27	2.25	3	2	0	2	0	1	0
M3 other	4	2	3	33.88	1.25	2	0	0	0	0	2	0
M3 quart.	2	1	2	20.6	1.0	2	0	0	0	0	0	0
M3 yearly	10	7	8	67.0	3.0	4	0	0	3	0	2	1
M4 daily	2	1	2	32.29	1.0	2	0	0	0	0	0	0
M4 hourly	6	4	5	2.9	1.83	2	0	1	2	0	1	0
M4 monthly	3	2	3	10.06	1.33	2	0	0	0	0	1	0
M4 quart.	5	4	4	38.0	1.8	3	0	1	0	0	1	0
M4 weekly	7	6	5	3.37	2.14	1	1	2	0	0	2	1
NN5 daily	6	4	5	5.6	1.67	4	0	0	0	2	0	0
NN5 weekly	4	3	3	1.0	1.5	1	0	1	0	0	0	2
Pedestrians	9	5	6	3.97	2.33	2	1	1	1	1	2	1
Rideshare	4	3	4	1	1.5	3	0	0	0	0	1	0
Saugeen	2	1	2	0.3	1	1	0	0	0	0	1	0
Solar 10mn												
Solar weekly	12	7	9	51.67	2.75	3	1	0	0	4	1	3
Sunspot	8	4	7	0.25	1.75	3	0	2	0	1	1	1
Temp. rain	8	5	6	12.4	2	4	1	0	0	1	0	2
Tourism monthly	4	2	4	8.38	1.5	2	0	1	0	0	1	0
Tourism quart.	7	5	6	25.75	2.14	2	0	1	2	0	2	0
Tourism yearly	9	6	6	68.0	2.22	3	1	1	1	1	2	0
Traffic hourly	9	6	7	2.76	2.44	4	1	1	0	2	0	1
Traffic weekly	7	5	5	2.20	1.71	1	0	3	1	0	2	0
Vehicle trips	8	5	7	9.83	2	2	1	0	3	0	0	2
Weather	8	4	6	38.8	2.38	2	1	3	0	0	1	1
Summary	6.4	4.13	5.25	17.2	1.91	41%	8.2%	9.8%	8.5%	7.8%	14.4%	10.1%



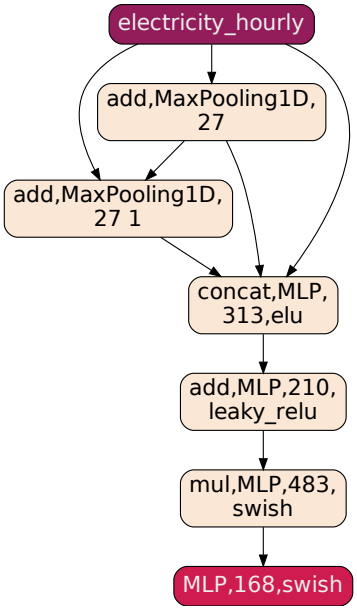
(a) M4 Daily, MASE: 1.066



(b) Solar Weekly, MASE: 0.511

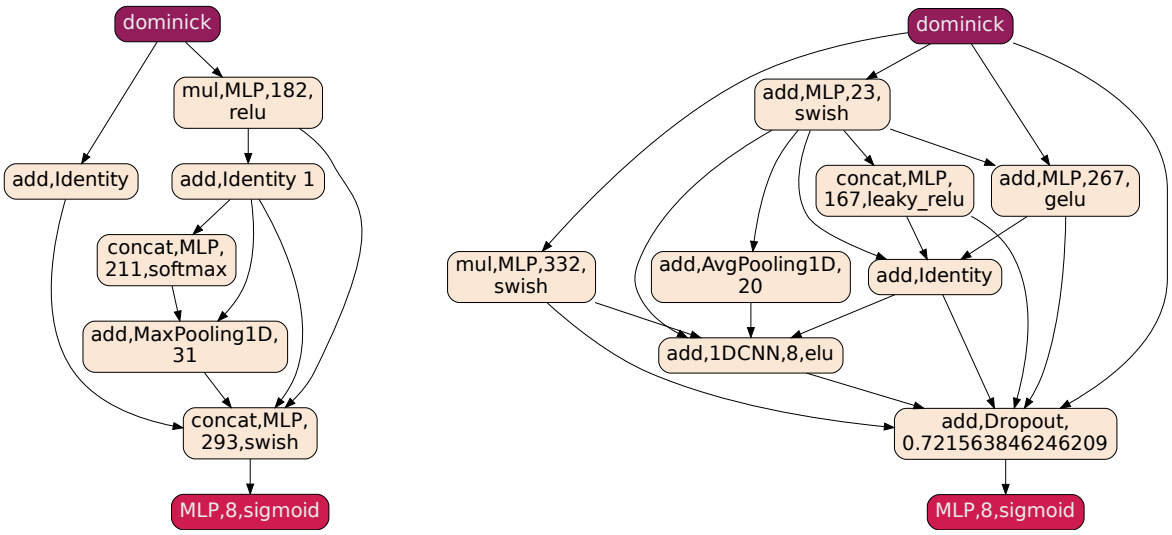


(c) Electricity weekly, MASE: 0.652



(d) Elec. hourly, MASE: 1.634

Figure 7: Best DNNs output by our algorithmic framework.

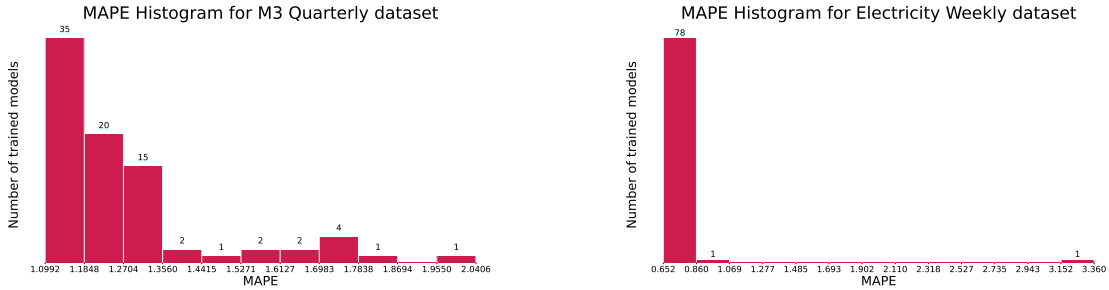


(a) Dominick, MASE: 0.5114

(b) Dominick, MASE: 0.5119

Figure 8: Two different models having similar good performance on Dominick dataset.

results on a dataset. Figure 9 shows the results on two datasets: M3 Quarterly and Electricity Weekly. For both datasets, we selected the best models found with our optimization and drew 80 seeds summing all instability and nondeterministic aspects of our models. We trained these models and plotted the MASE Figure 9. On the M3 Quarterly, the MASE reached values two times bigger than our best result. On the Electricity Weekly, it went up to five times worst. To overcome this problem, we represented the parametrization of stochastic aspects in our models as a hyperparameter, which we added to our search space. Despite its impact on the performance, we have not seen any work on NAS, HPO or AutoML trying to optimize the seed of DNNs. Our plots of Figure 9 showed that the optimization was effective as no other seeds gave better results than the one picked by our algorithmic framework.



(a) M3 Quarterly, best MASE: 1.099

(b) Electricity Weekly, best MASE: 0.652

Figure 9: Mape histogram of the best model performances with multiple seeds for two datasets.

6 Conclusion and Future Work

In this work, we introduced a novel algorithmic framework for the joint optimization of DNNs architectures and hyperparameters. We first introduced a search space based on Directed Acyclic Graphs, highly flexible for the architecture, and also allow for fine-tuning of hyperparameters. Based on this search space we designed search operators compatible with any metaheuristic able to handle a mixed and variable-size search space. The algorithmic framework is generic and has been efficient on the time series forecasting task using an evolutionary algorithm.

Further work would be dedicated to the investigation of other metaheuristics (e.g. swarm intelligence, simulated annealing) to evolve the DAGs. The reformulation of the studied optimization problems by including multiple objectives (e.g. complexity of DNNs) and robustness represent also important perspectives. To further improve the performance on time series forecasting tasks we can develop a more complete pipeline including a features selection strategy.

We can imagine further research work testing our framework on different learning tasks. Considering the forecasting task, the output models show that combining different state-of-the-art DNN operations within a single model is an interesting lead to improve the models' performance. Such models are quite innovative within the deep learning community and studies on their conduct and performances could be carried out.

Acknowledgments

This work has been funded by Electricité de France (EDF). The supercomputers used to run the experiments belong to EDF.

A Available operations and hyperparameters

Table 3: Operations available in our search space and used for the Monash time series archive dataset and their hyperparameters that can be optimized.

Operation	**Optimized hyperparameters**	
Identity	-	
Fully-Connected (MLP)	Output shape	Integer
Attention	Initialization type	[convolution, random]
	Heads number	Integer
1D Convolution	Kernel size	Integer
Recurrence	Output shape	Integer
	Recurrence type	[LSTM, GRU, RNN]
Pooling	Pooling size	Integer
	Pooling type	[Max, Average]
Dropout	Dropout Rate	Float

Activation functions, $\forall x \in \mathbb{R}^{\mathbb{D}}$

- Id: $\mathrm{id}(x) = x$
- Sigmoid: $\mathrm{sigmoid}(x) = \frac{1}{1+\mathrm{e}^{-x}}$
- Swish: $\mathrm{swish}(x) = x \times \mathrm{sigmoid}(\beta x) = \frac{x}{1+e^{-\beta x}}$
- Relu: $\mathrm{relu}(x) = \max(0, x)$
- Leaky-relu: $\mathrm{leakyRelu}(x) = \mathrm{relu}(x) + \alpha \times \min(0, x)$, in our case: $\alpha = 10^{-2}$
- Elu: $\mathrm{elu}(x) = \mathrm{relu}(x) + \alpha \times \min(0, e^x - 1)$
- Gelu: $\mathrm{gelu}(x) = x\mathbb{P}(X \leq x) \approx 0.5x(1 + \tanh[\sqrt{2/\pi}(x + 0.044715x^3)])$
- Softmax: $\sigma(\mathbf{x})_j = \frac{\mathrm{e}^{x_j}}{\sum_{d=1}^{D} \mathrm{e}^{x_d}}$ $\forall j \in \{1, \ldots, D\}$

B Monash datasets presentation

Table 4: Information about the Monash datasets [Godahewa et al., 2021].

Dataset	Domain	Nb of series	Multivariate	Lag	Horizon
Aus. elec	Energy	5	No	420	336
Births	Nature	1	No	9	30
Bitcoin	Economic	18	No	9	30
Carparts	Sales	2674	Yes	15	12
Dominick	Sales	115704	No	10	8
Elec. hourly	Energy	321	Yes	30	168
Elec. weekly	Energy	321	Yes	65	8
Fred MD	Economic	107	Yes	15	12
Hospital	Health	767	Yes	15	12
KDD	Nature	270	No	210	168
Kaggle weekly	Web	145063	Yes	10	8
M1 monthly	Multiple	1001	No	15	-
M1 quart.	Multiple	1001	No	5	-
M1 yearly	Multiple	1001	No	2	-
M3 monthly	Multiple	3003	No	15	-
M3 other	Multiple	3003	No	2	-
M3 quart.	Multiple	3003	No	5	-
M3 yearly	Multiple	3003	No	2	-
M4 daily	Multiple	100000	No	9	-
M4 hourly	Multiple	100000	No	210	-
M4 monthly	Multiple	100000	No	15	-
M4 quart.	Multiple	100000	No	5	-
M4 weekly	Multiple	100000	No	65	-
NN5 daily	Banking	111	Yes	9	-
NN5 weekly	Banking	111	Yes	65	8
Pedestrians	Transport	66	No	210	24
Rideshare	Transport	2304	Yes	210	168
Saugeen	Nature	1	No	9	30
Solar 10mn	Energy	137	Yes	50	1008
Solar weekly	Energy	137	Yes	6	5
Sunspot	Nature	1	No	9	30
Temp. rain	Nature	32072	Yes	9	30
Tourism monthly	Tourism	1311	No	2	-
Tourism quart.	Tourism	1311	No	5	-
Tourism yearly	Tourism	1311	No	2	-
Traffic hourly	Transport	862	Yes	30	168
Traffic weekly	Transport	862	Yes	65	8
Vehicle trips	Transport	329	No	9	30
Weather	Nature	3010	No	9	30

References

Zeina Abu-Aisheh, Romain Raveaux, Jean-Yves Ramel, and Patrick Martineau. An exact graph edit distance algorithm for solving pattern recognition problems. In *4th International Conference on Pattern Recognition Applications and Methods 2015*, 2015.

Arturo Hernández Aguirre and Carlos A Coello Coello. Evolutionary synthesis of logic circuits using information theory. *Artificial Intelligence Review*, 20:445–471, 2003.

Ahmad Alsharef, Karan Aggarwal, Manoj Kumar, Ashutosh Mishra, et al. Review of ml and automl solutions to forecast time-series data. *Archives of Computational Methods in Engineering*, pages 1–15, 2022.

Filipe Assunção, Nuno Lourenço, Penousal Machado, and Bernardete Ribeiro. DENSER: Deep Evolutionary Network Structured Representation. *arXiv:1801.01563 [cs]*, June 2018. doi: 10.1007/s10710-018-9339-y. URL `http://arxiv.org/abs/1801.01563`. arXiv: 1801.01563.

Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *International conference on machine learning*, pages 550–559. PMLR, 2018.

Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.

Andrés Camero, Hao Wang, Enrique Alba, and Thomas Bäck. Bayesian neural architecture search using a training-free performance metric. *Applied Soft Computing*, 106:107356, 2021.

Donghui Chen, Ling Chen, Zongjiang Shang, Youdong Zhang, Bo Wen, and Chenghu Yang. Scale-Aware Neural Architecture Search for Multivariate Time Series Forecasting. *arXiv:2112.07459 [cs]*, December 2021. URL `http://arxiv.org/abs/2112.07459`. arXiv: 2112.07459.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. On the relationship between self-attention and convolutional layers. *arXiv preprint arXiv:1911.03584*, 2019.

Siem Morten Johannes Dahl. *TSPO: an autoML approach to time series forecasting*. PhD thesis, 2020.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Difan Deng, Florian Karl, Frank Hutter, Bernd Bischl, and Marius Lindauer. Efficient automated deep learning for time series forecasting. *arXiv preprint arXiv:2205.05511*, 2022.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

Marcelo Fiore and Marco Devesas Campos. The algebra of directed acyclic graphs. In *Computation, Logic, Games, and Quantum Foundations. The Many Facets of Samson Abramsky*, pages 37–51. Springer, 2013.

Rakshitha Godahewa, Christoph Bergmeir, Geoffrey I Webb, Rob J Hyndman, and Pablo Montero-Manso. Monash time series forecasting archive. *arXiv preprint arXiv:2105.06643*, 2021.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

Rob J Hyndman and Anne B Koehler. Another look at measures of forecast accuracy. *International journal of forecasting*, 22(4):679–688, 2006.

Colin Lea, Michael D Flynn, Rene Vidal, Austin Reiter, and Gregory D Hager. Temporal convolutional networks for action segmentation and detection. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 156–165, 2017.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

Nan Li, Lianbo Ma, Guo Yu, Bing Xue, Mengjie Zhang, and Yaochu Jin. Survey on evolutionary deep learning: Principles, algorithms, applications and open issues. *arXiv preprint arXiv:2208.10658*, 2022.

Bryan Lim, Sercan Ö Arık, Nicolas Loeff, and Tomas Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, 37(4):1748–1764, 2021.

Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 82–92, 2019.

Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*, 2017.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

Zhenyu Liu, Zhengtong Zhu, Jing Gao, and Cheng Xu. Forecast methods for time series data: A survey. *IEEE Access*, 9:91896–91912, 2021. doi: 10.1109/ACCESS.2021.3091162.

Mohammad Loni, Sima Sinaei, Ali Zoljodi, Masoud Daneshtalab, and Mikael Sjödin. Deepmaker: A multi-objective optimization framework for deep neural networks in embedded systems. *Microprocessors and Microsystems*, 73: 102989, 2020a.

Mohammad Loni, Sima Sinaei, Ali Zoljodi, Masoud Daneshtalab, and Mikael Sjödin. DeepMaker: A multi-objective optimization framework for deep neural networks in embedded systems. *Microprocessors and Microsystems*, 73:102989, March 2020b. ISSN 01419331. doi: 10.1016/j.micpro.2020.102989. URL `https://linkinghub.elsevier.com/retrieve/pii/S0141933119301176`.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

Boris N. Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. N-BEATS: neural basis expansion analysis for interpretable time series forecasting. *CoRR*, abs/1905.10437, 2019. URL `http://arxiv.org/abs/1905.10437`.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pages 4095–4104. PMLR, 2018.

David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3):1181–1191, 2020.

Syed Yousaf Shah, Dhaval Patel, Long Vu, Xuan-Hong Dang, Bei Chen, Peter Kirchner, Horst Samulowitz, David Wood, Gregory Bramble, Wesley M Gifford, et al. Autoai-ts: Autoai for time series forecasting. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2584–2596, 2021.

Wei Wang Shu, Yao and Shaofeng Cai. Understanding architectures learnt by cell-based neural architecture search. *CoRR*, abs/1909.09569, 2019. URL `http://arxiv.org/abs/1909.09569`. Withdrawn.

Yao Shu, Wei Wang, and Shaofeng Cai. Understanding architectures learnt by cell-based neural architecture search. *arXiv preprint arXiv:1909.09569*, 2019.

David So, Quoc Le, and Chen Liang. The evolved transformer. In *International Conference on Machine Learning*, pages 5877–5886. PMLR, 2019.

Cecilia Summers and Michael J Dinneen. Nondeterminism and instability in neural network optimization. In *International Conference on Machine Learning*, pages 9913–9922. PMLR, 2021.

Yanan Sun, Bing Xue, Mengjie Zhang, and Gary G Yen. A particle swarm optimization-based flexible convolutional autoencoder for image classification. *IEEE transactions on neural networks and learning systems*, 30(8):2295–2309, 2018a.

Yanan Sun, Bing Xue, Mengjie Zhang, and Gary G. Yen. A Particle Swarm Optimization-based Flexible Convolutional Auto-Encoder for Image Classification. *arXiv:1712.05042 [cs]*, November 2018b. doi: 10.1109/TNNLS.2018.2881143. URL `http://arxiv.org/abs/1712.05042`. arXiv: 1712.05042.

El-Ghazali Talbi. Automated design of deep neural networks: A survey and unified taxonomy. *ACM Computing Surveys (CSUR)*, 54(2):1–37, 2021.

El-Ghazali Talbi. Metaheuristics for variable-size mixed optimization problems: a survey and taxonomy. *Submitted to IEEE Trans. on Evolutionary Algorithms*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Bin Wang, Yanan Sun, Bing Xue, and Mengjie Zhang. Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2018.

Bin Wang, Yanan Sun, Bing Xue, and Mengjie Zhang. Evolving deep neural networks by multi-objective particle swarm optimization for image classification. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 490–498, 2019a.

Bin Wang, Yanan Sun, Bing Xue, and Mengjie Zhang. Evolving Deep Neural Networks by Multi-objective Particle Swarm Optimization for Image Classification. *arXiv:1904.09035 [cs]*, April 2019b. URL `http://arxiv.org/abs/1904.09035`. arXiv: 1904.09035.

Colin White, Willie Neiswanger, and Yash Savani. Bananas: Bayesian optimization with neural architectures for neural architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10293–10301, 2021.

Muhan Zhang, Shali Jiang, Zhicheng Cui, Roman Garnett, and Yixin Chen. D-vae: A variational autoencoder for directed acyclic graphs. *Advances in Neural Information Processing Systems*, 32, 2019.

Guoqiang Zhong. DNA computing inspired deep networks design. page 8, 2020.