Applied Soft Computing 145 (2023) 110555

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

POPNASv3: A pareto-optimal neural architecture search solution for image and time series classification

Andrea Falanti [*], Eugenio Lomurno [*], Danilo Ardagna, Matteo Matteucci

DEIB, Politecnico di Milano, Via Giuseppe Ponzio, 34, Milano, 20133, Italy

ARTICLE INFO

Article history:
Received 6 February 2023
Received in revised form 7 June 2023
Accepted 13 June 2023
Available online 17 June 2023

Keywords:
Neural architecture search
Auto machine learning
Convolutional neural networks
Image classification
Time series classification
Pareto optimality
Multi-objective optimisation
Recurrent neural networks

ABSTRACT

The growing demand for machine learning applications in industry has created a need for fast and efficient methods to develop accurate machine learning models. Automated Machine Learning (AutoML) algorithms have emerged as a promising solution to this problem, designing models without the need for human expertise. Given the effectiveness of neural network models, Neural Architecture Search (NAS) specialises in designing their architectures autonomously, with results that rival the most advanced hand-crafted models. However, this approach requires significant computational resources and hardware investment, making it less attractive for real-world applications. This article presents the third version of Pareto-Optimal Progressive Neural Architecture Search (POPNASv3), a new NAS algorithm that employs Sequential Model-Based Optimisation and Pareto optimality. This choice makes POPNASv3 flexible to different hardware environments, computational budgets and tasks, as the algorithm can efficiently explore user-defined search spaces of varying complexity. Pareto optimality extracts the architectures that achieve the best trade-off with respect to the metrics considered, reducing the number of models sampled during the search and dramatically improving time efficiency without sacrificing accuracy. The experiments performed on image and time series classification datasets provide evidence that POPNASv3 can explore a large set of different operators and converge to optimal architectures suited to the type of data provided under different scenarios.[1]

1. Introduction

Machine learning (ML) is currently the most researched topic in the field of artificial intelligence (AI), as its application can solve and automate tasks that are difficult to solve with standard software programming. Over the last decade, deep learning [1,2] has gradually become the leading field in AI applications. Indeed, neural networks can take advantage of the huge amount of data collected by modern information systems and the advances in parallel computing technology to effectively learn multiple tasks such as image classification, speech recognition, time series prediction, and many others [3]. These complex models have gradually become the state-of-the-art solution in various domains, starting with image classification, of which famous architecture examples are AlexNet [4], VGG [5], and ResNet [6].

Machine learning techniques make it possible to automatically learn from data, but AI experts must design effective models and training procedures to properly fit the data and generalise the predictions to real-world scenarios. In the field of deep learning, this stage is referred to as *architecture engineering* and concerns the choice of layers used in the neural network model, the number of layers to stack, their hyperparameters (e.g. the number of filters, the kernel sizes), and other training details such as the best loss function for the task and the regularisation techniques to use. This process is tedious and time-consuming, as the model designer must iteratively run multiple experiments and adjust the architecture based on the results. In addition, the results of neural networks are difficult to interpret and explain, making it unintuitive how to modify the models to achieve performance improvements.

Automated Machine Learning (AutoML) is the solution to automate the model design step without the need for human expertise. It has recently gained considerable interest in the research community [7], as it could be one of the next breakthroughs in the field of AI, facilitating the mass adoption of ML in industry. Given the power and adaptability of neural networks, this approach has also been studied in deep learning and is known as *Neural Architecture Search* (NAS). The first successful application of NAS was published by Zoph et al. [8], who treated the search as a reinforcement learning (RL) problem, where a controller is able to output network structures and learn from their results. While

[*] Corresponding authors.
E-mail addresses: andrea.falanti@polimi.it (A. Falanti), eugenio.lomurno@polimi.it (E. Lomurno), danilo.ardagna@polimi.it (D. Ardagna), matteo.matteucci@polimi.it (M. Matteucci).

[1] The algorithm is open source and available on Zenodo at https://zenodo.org/record/7439022. The repository contains the code, the URIs of the datasets used in the experiments, and the models stored in ONNX format.

https://doi.org/10.1016/j.asoc.2023.110555
1568-4946/© 2023 Elsevier B.V. All rights reserved.

the performance of the final networks was on par with state-of-the-art hand-crafted architectures, the search required multiple GPUs and more than 30,000 cumulative GPU hours. For NAS to be successful in real-world scenarios, the search needs to be performed in a reasonable amount of time without the need for large hardware investments. However, the search space and methodology should be kept flexible to address different tasks and datasets in order to generalise the performance to problems not considered in the algorithm design. Subsequent NAS work has focused more on search efficiency and integrated multi-objective optimisation to address potential deployment constraints.

In this work, we propose the third version of Pareto Optimal Progressive Neural Architecture Search (POPNASv3), a Sequential Model-Based Optimisation [9] method whose workflow can be adapted to different tasks and data sources. Our algorithm can be applied to image and time series classification with minimal changes in its configuration, supporting large search spaces without sacrificing either time efficiency or performance in classification metrics. Results show that the top architectures found by our method are competitive with other state-of-the-art methods, with slightly lower accuracy but greater time efficiency, while exploring larger search spaces on average than other NAS algorithms. The search space can be potentially unbounded with our progressive approach, and POPNASv3 requires no pre-training, making it a good candidate for rapid prototyping and for studying the most promising combinations of operators among a large set.

This paper extends the previous versions of POPNAS [10,11] by introducing the following key contributions:

- The extension of the workflow to time series classification tasks, adding operators relevant to this domain, such as recurrent layers and dilated convolutions, and adapting the already supported operators to the new input dimensionality.
- The addition of two post-search procedures, the *model selection* and *final training* steps, which automate the optimisation of the architecture for final deployment. The model selection is performed on the top 5 architectures found during the search, training them more extensively and automatically tuning their macro-configuration. Final training is performed on the best configuration found during model selection, training the model to convergence.
- We adapt the training process to support different training devices, allowing each network to be trained on either a single GPU, multiple GPUs using synchronous variable replicas, or TPUs. We show that the time predictor is able to learn how the training time of the sampled networks varies on each device, making the algorithm robust to the different training strategies.
- The ability to use residual connections on cell outputs and fine-grained tuning to change the tensor shapes of each operator, allowing heterogeneous operators to be freely mixed within architectures.

The paper is divided into the following sections. Section 2 introduces NAS, the main works in the field regarding image classification, and the rationale between their design choices. It also gives an overview of the task of time series classification and the state-of-the-art in the field. Section 3 describes the POPNASv3 method in detail, reporting on the workflow shared with previous versions and extending it with the additions and improvements introduced by the new version. Section 4 covers the experiments performed, the configuration used, and the comparison of the results with other techniques. Finally, Section 5 summarises the contributions of the work in relation to the experimental results.

2. Background and related works

Neural Architecture Search (NAS) is a rapidly evolving area of research in the deep learning community, combining a variety of different techniques studied in machine learning. In Section 2.1, we provide a general description of NAS and summarise the intuitions behind the most endorsed works in the literature, on which the majority of subsequent algorithms are based. Most NAS algorithms are designed to solve image classification problems and rarely extend their search methods to other domains. In order to study the generalisation capabilities of our algorithm to different input domains, we decided to extend POPNASv3 to time series classification tasks, whose description and related state-of-the-art works are briefly presented in Section 2.2.

2.1. Neural architecture search

The goal of NAS is to autonomously design complex neural network architectures in a single end-to-end process that does not require human intervention. NAS is a young but popular field of research in the AI community, which still lacks proper standardisation, given the large number of techniques involved in NAS works. A tentative classification of NAS algorithms is provided by Elsken et al. [12], who characterises the methods according to three main features:

- The *search space*, which defines the set of all possible architectures that can be sampled by the algorithm.
- The *search strategy*, which defines how the algorithm samples the search space to find the optimal architectures for the task at hand.
- The *performance evaluation strategy*, which defines how to evaluate the quality of the architectures in the least amount of time.

Early NAS works [8,13,14], while successful in terms of the quality achieved by the final models, required enormous amounts of GPU time to find optimal architectures, making the application of NAS in real-world contexts prohibitively expensive. Several solutions and techniques have been considered to implement each NAS feature, with the aim of reducing the search time while maintaining the accuracy achieved by the top architectures.

Regarding the search space, each method typically defines a set of operators and how they are connected, as well as the number of layers or the maximum depth of the generated architectures. The number of architectures expressed by a defined search space is exponential in the number of operators, layers, and possible inputs per layer, so the cardinality of the search space grows rapidly even when targeting small architectures. NASNet [14] proposed a cell-based approach, where the search focuses on normal cells and reduction cells, which are basic units that aggregate multiple operations. This idea is inspired by famous models such as ResNet [6] and InceptionNet [15], where the final architecture is built by repeating a designed module several times. Similarly, the cells found by NASNet are repeated several times within the final neural network architecture, which is effective in restricting the cardinality to feasible sizes compared to freely searching the entire architecture structure. Thus, most algorithms focus their search effort on the *micro-architecture*, finding good cells for the task, while the *macro-architecture*, i.e. how and how many of these cells are stacked together, is often tuned after the search.

Instead, in terms of search strategy and performance estimation strategy, there are very heterogeneous proposals in the literature. PNAS [16] proposes a Sequential Model-Based Optimisation (SMBO) [9] technique, where the search starts with shallower models that are progressively extended with new layers, based on the indication of a surrogate model, called *predictor*.

The predictor provides an estimate of the accuracy achievable by each candidate architecture, and is adapted at runtime to the training results of previously sampled networks, making it resilient to search space and hyperparameter changes. Works such as AmoebaNet [13] and NSGANet [17] use evolutionary algorithms to perform the architecture search, while the weights are optimised using standard gradient descent techniques. In evolutionary algorithms, the search space represents the *phenotype*, while the architectures searched by the algorithm are mapped to *genotypes* via encoding. These works maintain a population of architectures at each iteration, whose genotypes are modified by mutation and crossover operations to produce an offspring. Mutations alter the encoding by randomly swapping bits, which in NAS encodings often results in adding/removing a layer or adding/removing a connection between two layers. Crossover operations, on the other hand, mix two encodings to create a new one that combines the characteristics of the parents. ENAS [18] increases search efficiency by combining a reinforcement learning search strategy with weight sharing, since the weights learned by previously trained architectures can be reused in similar ones to speed up the training phase. Several works have also started to consider NAS as a multi-objective optimisation problem [10,11,19,20], addressing potential deployment constraints and increasing search efficiency without sacrificing accuracy.

DARTS [21] improves search efficiency by applying a relaxation to the search space to make it continuous. This allows the entire search space to be represented as a single large-scale network, the *supernet*, where each edge between a pair of layers (i, j) is parameterised by a continuous variable α_{ij}. This change makes it possible to optimise both the weights and the structure of the model using gradient descent, alternating two specialised training steps. The final architecture is a subgraph of the supernet, selected by choosing the paths with the maximum parameters α at each branch. The efficiency of DARTS has contributed to the popularity of a new line of methods based on *one-shot* models [22,23], in which the supernet is defined and pre-trained before the actual neural architecture search is performed. During the search, each architecture sampled by the algorithm can be seen as a composition of paths of the supernet, whose weights are taken from those learned by the supernet and just fine-tuned to give an extremely fast evaluation of the network's accuracy. Nevertheless, the training of supernets is challenging because they are more sensitive to hyperparameter changes and weight co-adaptation, requiring special training techniques to avoid favouring certain architectures during the final estimation phase. The memory capacity of the training device limits the cardinality of the search space, as the entire network must reside on the device, but techniques such as elastic kernels and progressive shrinking proposed by Once-for-All [24] can mitigate this problem. Furthermore, one-shot models still require a significant amount of GPU hours for pre-training, which is only amortised if it can be reused multiple times on different NAS problems.

2.2. Time series classification

Time Series Classification (TSC) tasks present new challenges due to the variety of data sources used in this field and the different data correlations that can be exploited with this type of data. The goal of time series classification is to predict the correct label for each input series among multiple label options. TSC datasets can be univariate, if they have a single feature per time step, or multivariate, if they have multiple features per time step. Each feature represents a different time series from which the model can learn potential correlations for predicting the label. The data sources of time series samples are heterogeneous: for example, time series can represent audio data, spatial coordinates, or sensor readings from medical devices. These types of signals are sampled at different frequencies and may exhibit either long or short term behaviour. In addition, due to privacy concerns, these datasets tend to have significantly fewer samples than those used for image classification. These characteristics make it more difficult to find a single optimal model that can fit data from any domain well.

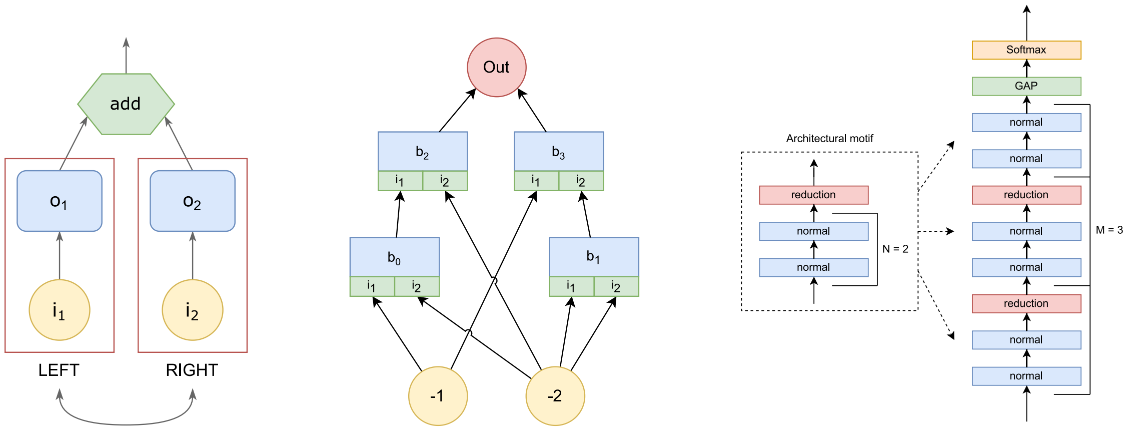
In the literature, TSC tasks have been approached through different ways of representing temporal patterns with the aim of extracting useful information. The most common approaches are distance-based, dictionary-based, interval-based and shapelet-based techniques [25]. State-of-the-art methods aggregate multiple data representations, making them robust to different application tasks. Most of these works have been tested on the UCR [26] and UEA [27] archives, which are curated groups of datasets commonly used as benchmarks to evaluate a model's performance on different TSC domains.

InceptionTime [28] is one of the most popular deep learning methods for TSC, based on learning an ensemble of convolutional neural networks. Each model shares the same structure, applying multiple convolutional layers with different kernel sizes in parallel to extract features from both short and long time windows. ROCKET [29] is another approach based on convolutional operators, but interestingly it does not train the weights as in deep learning methods, but simply generates them randomly. ROCKET demonstrates that it is possible to extract meaningful features from completely different datasets using only an order of 10000 randomly initialised convolutions with different properties. To produce the final output, a linear classifier is trained on the maximum value and the fraction of positive values extracted by each convolution, making ROCKET significantly faster to train than other state-of-the-art methods while achieving similar performance. The authors found that dilated convolutions contribute significantly to the effectiveness of the algorithm, a valuable insight for designing convolutional neural networks for these tasks.

Considering only the average accuracy, HIVE-COTEv2 [25] is currently the best performing method on the UCR and UEA archives. HIVE-COTEv2 is based on 4 different classifiers, which are themselves ensembles of several techniques and models. The final output is produced by a control unit that reweights the class probabilities produced by each classifier, based on an estimate of their quality. Although HIVE-COTEv2 outperforms the previously cited methods, it is computationally slower as it trains more models and does not support GPU parallelisation. In the NAS literature, the only work focusing on TSC is NAS-T [30], which uses an evolutionary algorithm to fine-tune a pre-trained ResNet [31] model with additional layers taken from the defined search space. NAS-T stops after 5000 network evaluations, or 48 h of GPU time, on each dataset of the UCR archive. The results of the experiments indicate that NAS-T is a marginal improvement over the first version of HIVE-COTE.

3. POPNASv3 method

For industrial applications of NAS, it is critical to achieve results similar to state-of-the-art architectures on generic tasks with reasonable hardware and GPU time investments. The goal of POPNASv3 is to provide fast prototyping of neural network architectures that optimises computational time without significant loss of accuracy compared to more expensive NAS methods. To achieve this, we have extended the POPNASv2 [11] method, which was inspired by the original POPNAS [10] and PNAS [16] methods. The algorithm proposed in POPNASv3 is characterised by a Sequential Model-Based Optimisation (SMBO) strategy, enriched by the presence of accuracy and time predictors to estimate the accuracy and training time of candidate architectures,



(a) block structure (b) cell structure example (c) macro-architecture overview

Fig. 1. An overview of all structural units defined in POPNASv3: *block*, *cell* and *architectural motif*. (a) A *block* is the combination of two operators into a single output. Since POPNASv3 uses *addition* as the join operator, which is commutative, LEFT–RIGHT and RIGHT–LEFT blocks are isomorphic and therefore computationally equivalent, so only one is generated. (b) An example of a *cell* specification, which is a DAG consisting of several interconnected blocks with a single output. Note that in addition to *lookback* inputs, each block can use another block previously listed in the cell specification as an input. (c) The standard macro-architecture used by POPNASv3 during the search process. An *architectural motif* consists of *N* normal cells followed by a reduction cell. *M* architectural motifs are stacked to form the final architecture, followed by Global Average Pooling (GAP) and Softmax as standard in classification networks.

respectively. Thus, our work treats NAS as a time-accuracy optimisation problem, selecting only those architectures that belong to the Pareto front computed on the estimates made by the predictors, since these models achieve the best trade-off between the two metrics. Pareto optimisation can reduce the number of networks to be sampled in each step, while maintaining a set of time-efficient diversified architectures. Training on structurally different architectures helps the predictors to find the key aspects that contribute to the gaps in training time and accuracy, progressively improving the accuracy of the Pareto front at each iteration. In this section, we provide a comprehensive explanation of how POPNASv3 implements the three main NAS features, i.e. the search space (Section 3.1), the performance estimation strategy (Section 3.2), and the search strategy (Section 3.3), to achieve the intended goal. We also describe the additional steps, namely *model selection* and *final training*, which are performed after the main search process to optimise the final architecture configuration and train it to convergence (Section 3.4).

3.1. Search space

The set of architectures included in the search space is defined by the chosen set of operators $\mathcal{O}$ and the set of hidden states $\mathcal{I}$ that can be used as input in the cell. Our work uses a cell-based approach, where each cell is composed of several micro-units called blocks, similar to PNAS [16] and NASNet [14]. A block (Fig. 1(a)) is specified by a tuple (in_1, op_1, in_2, op_2), with $in_1, in_2 \in \mathcal{I}$ and $op_1, op_2 \in \mathcal{O}$, where in_1 is processed by op_1 and in_2 by op_2. The new tensors are merged with a join operator, which in our case is fixed to the *add* operation to keep the number of output channels constant. A cell (Fig. 1(b)) is simply the composition of several blocks nested as a directed acyclic graph (DAG). Following the original intuitions of PNAS, POPNASv3 defines two types of cells: *normal* cells, whose output preserves the dimensionality of the input, and *reduction* cells, which halve the spatial resolution and double the filters between input and output. Normal and reduction cells have the same structure to maximise search efficiency: the only difference between them is that in reduction cells, all operators that directly process the output of a previous cell (i.e. a lookback input) perform a stride of factor 2 to reduce the spatial dimensionality. If the tensor shape has to be changed due to the architectural design, operators that cannot change the spatial dimensionality or the number of filters are followed by max pooling and pointwise convolution layers, respectively.

POPNASv3 focuses mainly on convolutional and pooling operators. The operators are defined in the configuration and recognised by regexes, allowing the kernel sizes and dilation rates to be parameterised for different tasks. For example, the 2D convolution operator is specified as the regex "`\d+x\d+(:\d+dr)? conv`", where the first two numbers define the kernel size and the optional third the dilation rate, which is set to 1 by default. For time series, it is also possible to define some popular RNN units, namely the long short-term memory (LSTM) [32] and the gated recurrent unit (GRU) [33]. All supported operators are:

- identity
- @x@(:@dr)? conv
- @x@(:@dr)? dconv (depthwise-separable convolution)
- @x@-@x@ conv (spatial-separable convolution)
- @x@ tconv (transpose convolution)
- @x@ maxpool
- @x@ avgpool
- LSTM (time series only)
- GRU (time series only)

where @ can be replaced by any number. Note that the convolutional and pooling operators can be used in time series classification by simply adjusting their kernel size to be monodimensional (e.g., @ conv, @ maxpool). All convolutional operators are implemented as a sequence of convolution, batch normalisation [34] and Swish [35] activation function. Most NAS methods use ReLU as the activation function because it can be derived immediately, making it very time efficient. However, in our training procedure, it rarely caused learning difficulties in flat architectures due to the dying neurons problem. From our preliminary studies, Swish seems to be more resilient to the initialisation of the learning rate and weights, helping to obtain more accurate estimates of the quality of all the networks included in the search space. However, we have found that using Swish results in an increase in network training time of more than 10% compared to ReLU, which is a not insignificant trade-off to pay in order to maximise search stability for general use.

The input set can be modified by defining the maximum allowed *lookback* input, i.e. the maximum distance in cells from which the current cell can take hidden states (outputs of previous cells). For example, the default lookback distance in the POPNASv3 configuration is two: this means that a cell can use the output of the previous cell (-1 lookback) and the penultimate cell (-2 lookback) as inputs for its blocks. In addition to lookback inputs, each block in a cell can use any preceding block in the cell specification as an input, allowing blocks to be nested in complex multi-level DAGs. In fact, each cell can be considered as a DAG using lookbacks as inputs and terminating with a single output node. Any unused block outputs are concatenated into the cell output, followed by a pointwise convolution to keep the number of filters consistent throughout the network. If enabled in the experiment configuration, the cell output can be made residual by summing the cell output with the closest lookback input used by the cell.

For the macro-architecture (Fig. 1(c)), we group the cells into units called *architectural motifs*. Each architectural motif consists of N consecutive normal cells, followed by a single reduction cell. At search time, the algorithm stacks M motifs to build each architecture, followed by Global Average Pooling (GAP) and Softmax to compute the classification label. POPNASv3 focuses on the micro-architecture search, while the macro-architecture hyperparameters are tuned later in the model selection phase.

3.2. Performance estimation strategy

All architectures sampled by the algorithm are trained for a short number of epochs in order to obtain a quick estimate of the quality achievable during longer training. This technique is often referred to in the NAS literature as *proxy training*. The training procedure is performed by the AdamW [36] optimiser with a cosine decay restart [37] schedule for both the learning rate and the weight decay, which allows for fast convergence and allows to quickly distinguish the most promising architectures from the suboptimal ones. Two surrogate models, called *accuracy predictor* and *time predictor*, guide the algorithm in selecting the most promising architectures at each step.

3.2.1. Accuracy predictor

The accuracy predictor is a modified version of the LSTM model proposed by PNAS, enriched with Self-Attention [38]. The structure of the accuracy predictor is shown in Fig. 2. This type of network can work on cell specifications, i.e. any list of blocks (in_1, op_1, in_2, op_2), without any additional pre-processing. The only requirement for applying the predictor is to separate the input values and operator values used in the cell into two tensors, encoded as 1-indexed categorical integer values, following the order of definition in the search space. Given the maximum number of blocks B that can be stacked in a single cell, both tensors have the form (B, 2), representing sequences of two inputs and two operators, respectively, belonging to each block of the cell. The input tensors of cells with less than B blocks are padded with zeros. For example, considering a search space with $\mathcal{I}$ = { -2, -1, 0, 1 }, $\mathcal{O}$ = { identity, 3×3 conv, 2×2 maxpool }, and a target number of blocks $B = 3$, the cell specification [(-2, 3×3 conv, -1, 3×3 conv), (0, 3×3 conv, -2, 2×2 maxpool)] is transformed into the following two tensors: [[1, 2], [3, 1], [0, 0]] for the cell inputs, [[2, 2], [2, 3], [0, 0]] for the cell operators.

The accuracy predictor embeds the inputs and operators separately to extract features specific to them. The input and operator embeddings are then concatenated and reshaped to retrieve the features associated with each block. This new hidden state is processed in parallel by two convolutions, outputting the Q and K matrices of the Self-Attention layer, which runs immediately after the convolutions. Finally, the Attention layer is followed by a bidirectional LSTM and a single sigmoid unit, whose output represents the expected accuracy of the architecture achieved after proxy training.

3.2.2. Time predictor

The POPNASv3 time predictor is implemented with a CatBoost [39] regressor, a state-of-the-art ML method based on gradient boosting and decision trees. In contrast to the case of accuracy prediction, we extrapolate a specific set of features from the cell structure instead of passing the cell encoding directly to the model. In particular, we developed the so-called *dynamic reindex* formula to represent the time efficiency of each operator $o \in \mathcal{O}$ compared to the others. The dynamic reindex function maps each operator to a value between 0 and 1, which is proportional to the estimated time impact of the considered operator compared to the one estimated to take the most time. These values are computed at runtime, after the algorithm has finished training the architectures composed of a block. Given as input the set of training times $\mathcal{T} = \{t_{op} | \forall op \in \mathcal{O}\}$ of the monoblock cells with symmetric encoding $[(-1, op, -1, op)]$ and the training time t_0 of the empty cell, i.e. the architecture consisting only of GAP and Softmax layers, the dynamic reindex formula is expressed as follows:

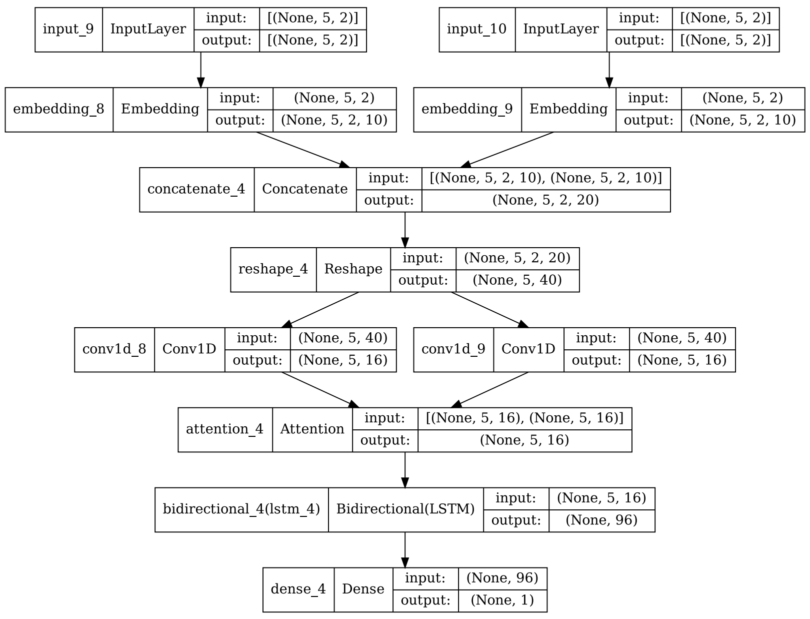
$$dynamic\_reindex_{op} = \frac{t_{op} - t_0}{max(T) - t_0}. \tag{1}$$

The training time t_0 required by the empty cell is considered a bias, representing the computational effort required by elements common to all architectures of the search space. The training time due to data pre-processing, data augmentation, and other training-related events, as well as the time spent training the output layers included in all architectures (i.e., GAP and Softmax), is captured by t_0 and therefore excluded from the dynamic reindex values. This allows the time predictor to more accurately learn the differences in computational cost between all operators in the search space. The features selected for the predictor are:

- the number of blocks;
- the number of stacked cells in the final architecture;
- the sum of the dynamic reindex value of all operators within the cell (OP score);
- the number of concatenated tensors in the cell output (number of unused blocks in the cell DAG);
- the use of multiple lookbacks (boolean);
- the depth of the cell DAG (in blocks);
- the number of block dependencies;
- the percentage of the total OP score involved in the heaviest cell path;
- the percentage of the total OP score due to operators using lookbacks as input.

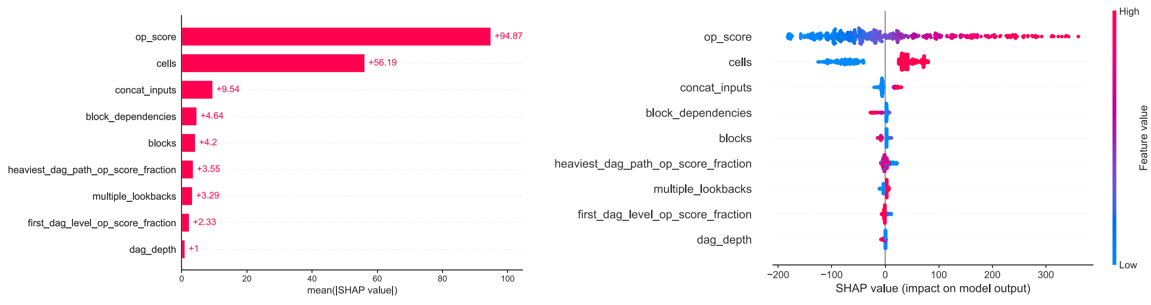
These features are related to the structural changes occurring in the cell's DAG and to the efficiency of the operators on the training device, making them suitable for any dataset, task and hardware environment.

POPNASv3 includes the SHAP [40] library, which we used to analyse the time features and check which ones are more relevant for the training time variations between the architectures. Fig. 3 shows the plots of the importance and impact of each feature on the final output, generated during a preliminary experiment performed to validate the design of the feature set. As expected, the most important features for estimating the required training time of a cell are the OP score, which is the sum of the dynamic reindex value of all operators connected in the cell DAG, and



input_9 | InputLayer | input: [(None, 5, 2)] | output: [(None, 5, 2)]
input_10 | InputLayer | input: [(None, 5, 2)] | output: [(None, 5, 2)]
embedding_8 | Embedding | input: (None, 5, 2) | output: (None, 5, 2, 10)
embedding_9 | Embedding | input: (None, 5, 2) | output: (None, 5, 2, 10)
concatenate_4 | Concatenate | input: [(None, 5, 2, 10), (None, 5, 2, 10)] | output: (None, 5, 2, 20)
reshape_4 | Reshape | input: (None, 5, 2, 20) | output: (None, 5, 40)
conv1d_8 | Conv1D | input: (None, 5, 40) | output: (None, 5, 16)
conv1d_9 | Conv1D | input: (None, 5, 40) | output: (None, 5, 16)
attention_4 | Attention | input: [(None, 5, 16), (None, 5, 16)] | output: (None, 5, 16)
bidirectional_4(lstm_4) | Bidirectional(LSTM) | input: (None, 5, 16) | output: (None, 96)
dense_4 | Dense | input: (None, 96) | output: (None, 1)

Fig. 2. The neural network structure designed for the POPNASv3 accuracy predictor. The model receives as input two tensors, whose values are extracted from the examined cell specification. The first tensor contains the input values used by the cell, while the other tensor contains the operators. Inputs and operators are coded as 1-indexed categorical values, following the order of definition in the search space. The input tensors are organised as a sequence of blocks, eventually zero-padded for cell specifications with fewer blocks than the number targeted by the search process. The accuracy predictor embeds the two tensors separately, as they refer to conceptually different entities. The embeddings are later merged to represent the embedding of a series of blocks, processed by self-attention, a bidirectional LSTM and a single sigmoid unit to achieve the final result.



(a) SHAP importance per feature (b) SHAP impact on output per feature

Fig. 3. Analysis of the importance of each feature used in the time predictor, carried out using the SHAP library. The results are extracted from a preliminary experiment performed on the CIFAR100 dataset (training data up to 4 blocks, for the prediction of the extension to 5 blocks). (a) An overview of how much the features are involved in the prediction outputs. (b) The actual numerical impact of the features on the training time predictions, in seconds.

the number of cells stacked in the final architecture. The search configuration fixes the maximum number of cells stacked in the macro-architecture, but the actual number of stacked cells can be reduced if the cell specification never uses sequential lookback, i.e. a -1 input value, so that some cells are not connected to the final output of the architecture. The number of unused blocks follows the two features mentioned above, and this is to be expected since it involves a structural change in the cell. If several block outputs are not used by other blocks, they are concatenated and then a pointwise convolution is performed to adjust the number of filters. The FLOPs performed by the pointwise convolution increase with the number of filters in the concatenation tensor, resulting in a time overhead proportional to the number of unused blocks. Interestingly, features such as block dependencies, DAG depth, and the percentage of OP score involved in the heaviest cell DAG path are not as relevant, although they should give a measure of how parallelizable the execution of the neural network layers is on the training device. This observation may change with different frameworks, implementations and hardware.

3.3. Search strategy

Following the PNAS [16] strategy, POPNASv3 starts the search from the simplest architectures of the search space and progressively builds more complex architectures by adding blocks to those previously sampled. This type of search, which alternates between fitting models and using their results to explore unseen regions of the search space, is referred to in the literature as Sequential Model-Based Optimisation [9]. In our work, time and accuracy predictors learn from previous network results which blocks are more suitable to expand the currently evaluated architectures. The models are expanded one block at a time until the target number of blocks is reached. A complete overview of the search strategy is given in Algorithm 1.

The procedure starts with the training of the *empty cell*, which is the architecture composed only of the GAP and Softmax layers.

Algorithm 1 POPNASv3 search strategy

```
Require: $B$ (max num of blocks), $E$ (epochs), $K$ (beam size), $J$ (exploration beam size),
    $hp$ (networks hyperparameters), $dataset$.
 1: $S_0 = \{\text{empty cell}\}$
 2: $\mathcal{A}_0, \mathcal{T}_0 = \text{train-cells}(S_0, E, \text{hp, dataset})$
 3: $S_1 = \mathcal{B}_1$
 4: $\mathcal{A}_1, \mathcal{T}_1 = \text{train-cells}(S_1, E, \text{hp, dataset})$
 5: dyn-reindex-map = compute-dynamic-reindex-map($\mathcal{T}_0, \mathcal{T}_1$)
 6: for $b = 2 : B$ do
 7:     $\mathcal{P}_{acc} = \text{fit}(\mathcal{A}_{0\to b-1}, S_{0\to b-1})$
 8:     $\mathcal{F}_{b-1} = \text{extract-time-features}(S_{b-1}, \text{dyn-reindex-map})$
 9:     $\mathcal{P}_{time} = \text{fit}(\mathcal{T}_{0\to b-1}, \mathcal{F}_{0\to b-1})$
10:     $S'_b = \text{expand-cells}(S_{b-1})$
11:     $\hat{\mathcal{A}}'_b = \text{predict}(S'_b, \mathcal{P}_{acc})$
12:     $\hat{\mathcal{T}}'_b = \text{predict}(S'_b, \mathcal{P}_{time})$
13:     $S_b = \text{build-pareto-front}(S'_b, \hat{\mathcal{A}}'_b, \hat{\mathcal{T}}'_b, K)$
14:     $\tilde{\mathcal{O}}, \tilde{\mathcal{I}} = \text{build-exploration-sets}(S_b)$
15:     if $|\tilde{\mathcal{O}}| > 0 \lor |\tilde{\mathcal{I}}| > 0$ then
16:         $S_{b,exp} = \text{build-exploration-pareto-front}(S'_b, \hat{\mathcal{A}}'_b, \hat{\mathcal{T}}'_b, S_b, J)$
17:     else
18:         $S_{b,exp} = \{\}$
19:     end if
20:     $\mathcal{A}_b, \mathcal{T}_b = \text{train-cells}(S_b \cup S_{b,exp}, E, \text{hp, dataset})$
21: end for
```

Algorithm 2 Model selection

```
Require: $R$ (search results), $H_s$ (search hyperparameters), $H_{ms}$ (model selection hyperparameters),
    $K$ (top networks to consider), $P_{min}$, $P_{max}$, $dataset$.
 1: $M, N, F = \text{get-original-macro-config}(H_s)$
 2: config = merge-configs($H_s, H_{ms}$)
 3: $M_{mod} = [0, 1]$
 4: $N_{mod} = [0, 1, 2]$
 5: $F_{mod} = [0.85, 1, 1.5, 1.75, 2]$
 6: $\mathcal{C} = \text{get-top-k-cells}(R, K)$
 7: for $c \in \mathcal{C}$ do
 8:     $a_{c,M,N,F} = \text{train-network}(c, M, N, F, \text{config, dataset})$
 9:     for $(m_{mod}, n_{mod}) \in (M_{mod} \times N_{mod})$ do
10:         $M' = M + m_{mod}$
11:         $N' = N + n_{mod}$
12:         $F' = \max(\{f' = F \cdot f_{mod} | f_{mod} \in F_{mod} \land P_{min} < \text{get-params}(c, M', N', f') < P_{max}\})$
13:         $a_{c,M',N',F'} = \text{train-network}(c, M', N', F', \text{config, dataset})$
14:     end for
15: end for
16: return $(c, M', N', F')$ with $\max(a_{c,M',N',F'})$
```

The importance of the empty cell is that it captures elements common to all architectures of the search space. The achieved accuracy a_0 and the required training time t_0 can therefore be fed to the predictors to calibrate the bias and improve the estimates. Taking the training time t_0 as an example, it measures the time required to execute the final layers common to all architectures, plus the time overhead due to data pre-processing and data augmentation procedures. After training the empty cell, the algorithm bootstraps the search by training all architectures with a single block contained in the search space. The accuracy $\mathcal{A}_1$ and time $\mathcal{T}_1$ achieved by these architectures are collected to train the respective predictors. While the accuracy predictor can directly process the cell specifications, i.e. lists of block encodings, the time predictor requires the extrapolation of a designed set of features from the cell structure. A *dynamic reindex map* is generated from t_0 and T_1, providing a numerical measure of the time complexity of each operator $o \in \mathcal{O}$. (more details about the time features and dynamic reindex, provided in Formula (1), are given in Section 3.2.2).

From this point, the search proceeds iteratively, analysing the previous results at each step to identify optimal regions of the search space, and expanding the architectures in those directions with an additional block. At each iteration, the time and accuracy predictors are fitted to the data of all previously trained architectures. The algorithm then lists all possible network expansions $\mathcal{S}'_b$ obtained by adding any single block to a cell specification trained in the previous step ($\mathcal{S}_{b-1}$). During the expansion step, POPNASv3 prohibits the generation of specular block encodings, which would lead to *graph isomorphisms* and thus to computationally equivalent neural networks. Fig. 1(a) illustrates the block isomorphism problem. Identifying graph isomorphisms is critical to avoid wasting resources training the same network multiple times, which would increase search time and reduce network diversity within the Pareto front. The predictors are then used to estimate $\hat{a}_s, \hat{t}_s$ of each architecture $s \in \mathcal{S}'_b$, allowing the construction of a time-accuracy Pareto front $\mathcal{S}_b$ with the architectures that achieve the best trade-off between the two metrics. While constructing the Pareto front, POPNASv3 detects and prunes potential graph isomorphisms between the candidate cells and those already inserted in the Pareto front. Cell isomorphism detection is not performed during cell expansion because graph isomorphism is an NP-complete problem and it is computationally expensive to apply it to sets of large cardinality. Instead, since the Pareto front is limited to K elements, it is feasible to perform the detection on this limited number of networks.

In addition to the Pareto front architectures, an *exploration step* can potentially introduce new architectures to be trained, triggered only if the predictors strongly discourage the use of some operators or inputs. Specifically, the algorithm counts how many times each $o \in \mathcal{O}$ and $i \in \mathcal{I}$ appear in the Pareto front networks. Operators and input values that are used at least 5 times less than the average usage in a perfectly balanced case are inserted into the exploration operator set $\tilde{\mathcal{O}}$ and the exploration input set $\tilde{\mathcal{I}}_b$. If the exploration sets are not empty, an additional Pareto front is constructed with only those architectures that contain at least a fixed number of inputs and operators from the exploration sets, eliminating the architectures already selected in the standard Pareto front and those that are isomorphic. The iteration ends by training all the architectures belonging to the Pareto front or the exploration Pareto front, collecting their results to train the predictors in future iterations. The POPNASv3 search procedure ends when the architectures have been expanded to the target number of blocks B defined in the experiment configuration.

3.4. Post-search procedures

The architectures sampled during the search are only trained for a small number of epochs, so they do not converge to their best results and require additional training. Two additional steps, which we call *model selection* and *final training*, are performed after the search procedure to determine the best architecture after prolonged training and to maximise its performance. While the search process focuses only on the micro-architecture, i.e. finding the best cell structures, the model selection can tune the macro-architecture by testing different configurations. Instead, final training extensively optimises the weights of the best architecture found by model selection. Therefore, a deployable neural network model can be produced by performing the search and these two procedures sequentially.

The accuracy results obtained during the search, even with a short training period, are indicative of the potential of the sampled networks and their ability to generalise. Therefore, we simply select the top K architectures found during the search for the model selection process. The model selection step consists of training the candidate architectures for a significant number of epochs, using the same training and validation dataset splits used in the search, so that we can determine the best architecture for the task. An analogous procedure is also performed in NASNet [14], but in POPNASv3 we limit it to the top 5 architectures by default, as Pareto optimisation ranks the architectures and makes the differences between cells more apparent. If there are specific deployment constraints on metrics such as number of parameters and inference time, it is possible to exclude architectures that do not satisfy the constraints, as these metrics are computed and logged during the search process. During model selection, we introduce additional generalisation techniques, such as scheduled drop path, introduced in NASNet [14] as a modification of the drop path [41] technique whose drop rate scales with the number of iterations, and cutout [42] data augmentation. These techniques are generally not recommended for the proxy training performed during the search, as they slow down the learning of the architectures, but are essential to avoid overfitting when training for a large number of epochs.

The model selection procedure is illustrated in Algorithm 2. Model selection can optionally tune the macro-architecture structure, i.e. the number of motifs M, the number of normal cells N stacked in each motif, and the number of starting filters F. These parameters are fixed and not optimised during the search to avoid exponential growth of the search space cardinality. The architectures are trained using the macro-structure configuration provided during the search, then, given a range of parameters (P_{min}, P_{max}) as an argument, the macro-architecture of each cell is individually tuned using grid search on sets of modifiers. The modifiers are defined by the algorithm as the number of motifs M_{mod} and normal cells per motif N_{mod} to add compared to the search configuration, plus a set of multipliers F_{mod} for the initial filters. We refer to all possible values for the modified macro-hyperparameters as M', N' and F'. For each (cell-encoding, M', N') triplet, the algorithm trains the architecture with the largest number of filters F' that fit the defined parameter range; if no F' value satisfies the parameter constraint, the triplet is discarded.

After model selection, the best architecture configuration is retrained using all available data for an even larger number of epochs, aiming for the best deployment performance. The dataset is no longer split into training and validation, so this last step should only be performed if the architecture generalised well during model selection training, to avoid overfitting. The hyperparameters are set exactly as in the model selection phase, except that the number of epochs is increased to maximise training convergence. At the end of training, the architecture is evaluated on the test set to obtain an estimate of its final accuracy in an actual deployment.

4. Experiments and results

In this section, we describe the experiments performed to evaluate the performance of POPNASv3 and compare the results with PNAS, the method from which our search strategy was designed. We have performed experiments on several Image Classification (IC) and Time Series Classification (TSC) datasets commonly used in the literature, checking the robustness of the algorithm to different input sizes, channels, number of classes and domains. The image classification datasets and their main characteristics are presented in Table 1. For TSC, we first considered the four datasets listed in Table 2 to validate the algorithm design

Table 1
Datasets selected for image classification experiments.

Dataset	Train size	Test size	Input size	Channels	# Classes
CIFAR10 [44]	50000	10000	32×32	3 (RGB)	10
CIFAR100 [44]	50000	10000	32×32	3 (RGB)	100
Fashion MNIST [45]	60000	10000	28×28	1 (Grayscale)	10
EuroSAT[a] [46]	21600	5400	64×64	3 (RGB)	10

[a]: EuroSAT provides only the training set, the test set is reserved with the last 20% of the samples, as done by the EuroSAT authors themselves [46].

Table 2
Datasets selected for the initial time series classification experiments.

Dataset	Train size	Test size	Length	Channels	# Classes
ElectricDevices [26]	8926	7711	96	1 (Univariate)	7
FordA [26]	3601	1320	500	1 (Univariate)	2
FaceDetection [27]	5890	3524	62	144 (Multivariate)	2
LSST [27]	2459	2466	36	6 (Multivariate)	14

and tune the algorithm hyperparameters, and later extended the experiments to a larger selection of datasets from the UCR [26] and UEA [27] archives. In this second set of experiments, we selected all datasets with at least 360 training samples, as neural networks could easily overfit with less data, making accuracy predictions noisier and less reliable. In the latter cases, we argue that NAS is not an adequate solution and that custom training pipelines, together with domain-specific data augmentation techniques, produce better models.

POPNASv3 is a direct extension of the PNAS search algorithm, also using an improved model structure. In order to make a comparison between the two techniques, and since the original PNAS codebase is not publicly available, we have also performed the image classification experiments on the POPNASv3 workflow without the Pareto optimisation and architectural modifications introduced by this work. In this way, we simulate a PNAS execution on the POPNASv3 search space, making it possible to directly compare their results. PNAS was not designed to handle time series classification tasks, so we only provide the POPNASv3 results for these datasets.

4.1. Experiments configuration

POPNASv3 models are implemented using the Keras API provided in TensorFlow 2.7.3 [43]. Most of the experiments were performed on a single NVIDIA A100 GPU partitioned using the MIG configuration 3g.40gb, while model selection and final training were performed on the entire A100 without partitions. To investigate the impact of different hardware on the predictors, we also ran some experiments on TPU and multi-GPU settings, which are reported in Appendix A.

We use a similar configuration for all experiments, mainly adapting the search space and the target number of epochs in post-search procedures between image classification and time series classification tasks. Table 3 gives an overview of the main hyperparameters used in each step of the algorithm. The operator set $\mathcal{O}$ is adapted depending on the input type. For all image classification tasks we use $\mathcal{O}$ = { *identity, 3×3 dconv, 5×5 dconv, 7×7 dconv, 1×3-3×1 conv, 1×5-5×1 conv, 1×7-7×1 conv, 1×1 conv, 3×3 conv, 5×5 conv, 2×2 maxpool, 2×2 avgpool* }; instead for the time series classification we use $\mathcal{O}$ = { *identity, 7 dconv, 13 dconv, 21 dconv, 7 conv, 13 conv, 21 conv, 7:2dr conv, 7:4dr conv, 2 maxpool, 2 avgpool, lstm, gru* }. The maximum lookback distance is set to 2 for all experiments. The search continues until the cells are expanded to $B = 5$ blocks, retaining a maximum of $K = 128$ architectures in each step, plus up to $J =$

Table 3
Summary of hyperparameters used during the neural architecture search procedure and the post-search steps.

Hyperparameter	Neural architecture search	Model selection	Final training
epochs	21	200[a], 100[b]	600[a], 200[b]
learning rate	0.01	0.01	0.01
weight decay	0.0005[a], 0.001[b]	0.0005[a], 0.001[b]	0.0005[a], 0.001[b]
optimiser	AdamW	AdamW	AdamW
scheduler	cosine decay restart	cosine decay	cosine decay
drop path rate	0.2[b]	0.4[a], 0.6[b]	0.4[a], 0.6[b]
label smoothing		0.1	0.1
motifs	3	tuned	from model selection
normal cells per motif	2	tuned	from model selection
filters	24	tuned	from model selection
secondary exit		✓	✓
validation set size	10%	10%	
data augmentation[a]	✓	✓	✓
cutout[a]		✓	✓
z-normalisation[b]	✓	✓	✓

[a]: Image classification only.
[b]: Time series classification only.

16 additional architectures if the exploration step is triggered. During the search, the hyperparameters related to the macro-architecture are fixed for all models, building architectures with $M = 3$ motifs, $N = 2$ normal cells per motif, and $F = 24$ starting filters. The neural network models are trained for $E = 21$ epochs, using the AdamW [36] optimiser with cosine decay restart scheduler to adjust the learning rate and weight decay during training. The learning rate is initially set to 0.01 and the weight decay to 0.0005 for IC tasks, while it is increased to 0.001 for TSC tasks. This configuration allows for fast convergence with minimal time investment, providing good estimates of the capabilities of each sampled architecture. The IC tasks use random shifting and random horizontal flipping for data augmentation, while the TSC datasets are only pre-processed by z-normalising the data and do not use any data augmentation techniques.

The accuracy predictor is an ensemble of five LSTM-based models with attention, with the structure defined in Section 3.2.1. The ensemble is trained using K-folds, with $K = 5$, where each model is trained on a different fold. The models share the same hyperparameters, using a learning rate of 0.004 with the Adam [47] optimiser and no scheduler. Weight regularisation is applied with a factor of 0.00001. The Catboost regressor, used as a time predictor, performs a random search hyperparameter optimisation at runtime, searching for the best parameters within this search space:

- learning_rate: uniform(0.02, 0.2)
- depth: randint(3, 7)
- l2_leaf_reg: uniform(0.1, 5)
- random_strength: uniform(0.3, 3)
- bagging_temperature: uniform(0.3, 3)

The training procedure uses K-folds, with $K = 5$, training each model for 2500 iterations and using early-stopping with patience set to 50 iterations.

After the search, the top 5 architectures are tuned in the model selection phase using as modifiers: $M_{mod} = [0, 1]$, $N_{mod} = [0, 1, 2]$ and $F_{mod} = [0.85, 1, 1.5, 1.75, 2]$. Both the model selection and the final training phases use the hyperparameters used during the search as a baseline, overriding some of them in order to maximise performance in extended training. The networks found on IC datasets are trained for 200 epochs during the model selection phase and for 600 epochs during the final training of the best architecture, while for TSC tasks the number of epochs is reduced to 100 and 200, respectively, since they converge much faster. The scheduler is changed to cosine decay, the scheduled drop path is enabled with a rate of 0.4 in IC and 0.6 in TSC tasks, and label smoothing [48] is set in the loss function with a value of 0.1. A secondary output is added at 2/3 of the total architecture length to improve generalisation, and the loss weights are set to 0.75 for the main output and 0.25 for the secondary output. For image classification tasks, we also enable a random cutout [42] of size (8, 8) to further improve generalisation.

4.2. Predictor results and pareto front analysis

In this section, we evaluate the performance of the POPNASv3 predictors for both image classification and time series classification tasks. The construction of the Pareto front depends on the correct ranking of the candidate architectures, so it is fundamental to evaluate the ranking quality of the predictions at each step. To analyse the ranking and the accuracy we use the Spearman's rank correlation coefficient (ρ) and the mean average percentage error (MAPE) respectively. The results of the accuracy predictor are given in Table 4, while the results of the time predictor are given in Table 5. We also performed some experiments on the CIFAR10 dataset using different hardware configurations to check if the predictors are robust to different distribution strategies and training devices. The results of the hardware experiments are reported in Appendix A.1.

The accuracy predictions made by the LSTM ensemble generally have small errors, with some exceptions. We note that most of the inaccuracies occur during the $b = 2$ step, in datasets that are difficult to fit in a short number of epochs. In these cases, the predictor tends to overestimate the gain obtained by adding the second block, but this behaviour is corrected in the later iterations. The ranking provided by the predictor is quite accurate for image classification tasks, contributing significantly to the accuracy of the Pareto front. For time series classification tasks, the results tend to be noisier, but this behaviour is to be expected given the smaller amount of data available and the lack of generic data augmentation techniques.

In terms of time prediction, the MAPE is significantly higher in the $b = 2$ step. The error is due to features related to structural changes in multi-block cells that are constant for mono-block architectures, making it impossible for the time predictor to learn the importance of these features without pre-training. Although this is a limitation of our feature set, the error is a common bias for most architectures and therefore does not affect the ranking. The MAPE tends to decrease in later steps as the predictor can learn these behaviours specific to more complex cells. Nevertheless, the ranking is optimal in time predictions, with most Spearman coefficients tending towards 1. The quality of the ranking compensates for the suboptimal MAPE in the early

Table 4
The results of POPNASv3 accuracy predictor for each evaluated dataset.

Dataset	MAPE(%)				Spearman(ρ)			
	$b = 2$	$b = 3$	$b = 4$	$b = 5$	$b = 2$	$b = 3$	$b = 4$	$b = 5$
CIFAR10	1.967	1.304	0.927	0.777	0.888	0.832	0.882	0.960
CIFAR100	13.492	1.635	2.354	2.766	0.909	0.820	0.947	0.945
Fashion MNIST	1.743	0.779	1.063	0.940	0.663	0.531	0.773	0.935
EuroSAT	1.325	0.874	0.615	0.597	0.605	0.840	0.810	0.872
ElectricDevices	1.934	1.321	1.144	0.595	0.830	0.945	0.818	0.817
FordA	2.842	0.717	0.652	0.822	0.633	0.568	0.524	0.688
FaceDetection	7.375	2.943	2.743	1.952	0.750	0.551	0.247	0.477
LSST	7.504	9.208	15.393	14.716	0.827	0.822	0.736	0.775

Table 5
The results of POPNASv3 time predictor for each evaluated dataset.

Dataset	MAPE(%)				Spearman(ρ)			
	$b = 2$	$b = 3$	$b = 4$	$b = 5$	$b = 2$	$b = 3$	$b = 4$	$b = 5$
CIFAR10	17.219	10.134	5.487	8.709	0.960	0.979	0.991	0.979
CIFAR100	21.830	6.459	6.334	6.599	0.982	0.995	0.990	0.994
Fashion MNIST	23.912	19.755	11.614	11.512	0.965	0.962	0.964	0.922
EuroSAT	17.852	7.815	5.839	4.707	0.976	0.995	0.996	0.994
ElectricDevices	27.792	12.390	11.198	6.377	0.920	0.970	0.979	0.987
FordA	26.059	14.740	6.802	3.827	0.879	0.958	0.896	0.919
FaceDetection	13.781	11.214	7.479	13.054	0.646	0.832	0.922	0.947
LSST	16.508	22.587	33.984	10.300	0.717	0.966	0.836	0.940

stages, as Pareto front correctness is the key to increasing search efficiency without discarding potentially optimal solutions.

The plots shown in Fig. 4 give an overview of how Pareto optimisation is effective for fast exploration of large search spaces. POPNASv3 is constrained to select at most K architectures in each expansion step, but the cardinality of the possible cell expansions is orders of magnitude larger than K. From Fig. 4(a) it is clear that there is a high density of architectures with promising accuracy, but it is a waste of resources to consider them all, as done in PNAS, since they are often associated with long training times and the estimated accuracy gap between them is negligible. In addition, the time-accuracy Pareto front often contains a number of architectures below K, which results in fewer architectures to sample and further speeds up the search. Although more costly, sampling the networks using only their estimated accuracy, as done by PNAS, is more error-proof for finding high-quality cells, as the predictors may produce some outliers and noisy estimates. In this respect, the POPNASv3 exploration step introduces in each iteration some cells that use inputs and operators that are rarely selected in Pareto front architectures. The results of these peculiar cells help the predictor to select more diversified networks when their results compete with those selected in the Pareto front. Thus, the exploration step mitigates possible noise in the predictions due to biases introduced by the simplest architectures and stabilises the convergence to the optimal regions of the search space.

4.3. Image classification results

For the image classification datasets, we ran the experiments on both POPNASv3 and PNAS, using the same hyperparameter configuration, in order to make a fair comparison between the two methods. Apart from the absence of the Pareto front and the exploration step from the procedure, PNAS does not use residual units at the cell outputs and reshapes the lookbacks with a pointwise convolution before generating the next cell if their shape is not the same. Other improvements introduced by POPNASv3 that do not change the search strategy and macro architecture, such as the use of Swish as the activation function and the detection of graph isomorphisms, are instead integrated into our PNAS implementation.

Table 6 shows the search results obtained with the two methods. Considering that POPNASv3 and PNAS bootstrap the search in the same way and use the same search space in our experiments, the first 301 sampled cell specifications are equivalent in the two methods. In the next computations, we exclude these architectures from the network count and search time to better estimate the effect of Pareto optimality on search efficiency. On average, POPNASv3 trains 56% fewer architectures than the target population size K fixed in the configuration, which is instead always saturated by PNAS in each step, given its search strategy definition. The restricted cardinality of the Pareto front contributes significantly to the search time reduction, but the time-accuracy optimisation also reduces the average training time of these networks, achieving much better speed-ups than the estimated 2×factor due to architecture reduction. From the results, POPNASv3 speeds up the search procedure on average by a 4.8×factor when considering only $b >= 2$ steps, while the improvement is still a significant 3.4×factor when considering the entire search procedure. In terms of accuracy achieved after proxy training, the top architectures found by POPNASv3 and PNAS do not differ significantly, except for the CIFAR100 dataset where POPNASv3 models perform better. This latter difference is due to the residual connections added to the output of POPNASv3 cells, which improve training convergence by facilitating gradient propagation, thereby increasing accuracy in datasets where the results after proxy training are far from optimal. As POPNASv3 samples fewer networks, we check whether our algorithm can find multiple solutions with similar performance. The average accuracy of the top five architectures found on each dataset is close to the accuracy of the best one, indicating that Pareto optimisation does not introduce large accuracy gaps between the most promising architectures. Post-search procedures can therefore train these architectures more extensively and investigate which configuration performs better at convergence based on possible deployment constraints.

After searching, we performed model selection on the top-5 cells found, using a parameter range of $(2.8 \times 10^6, 3.5 \times 10^6)$ for

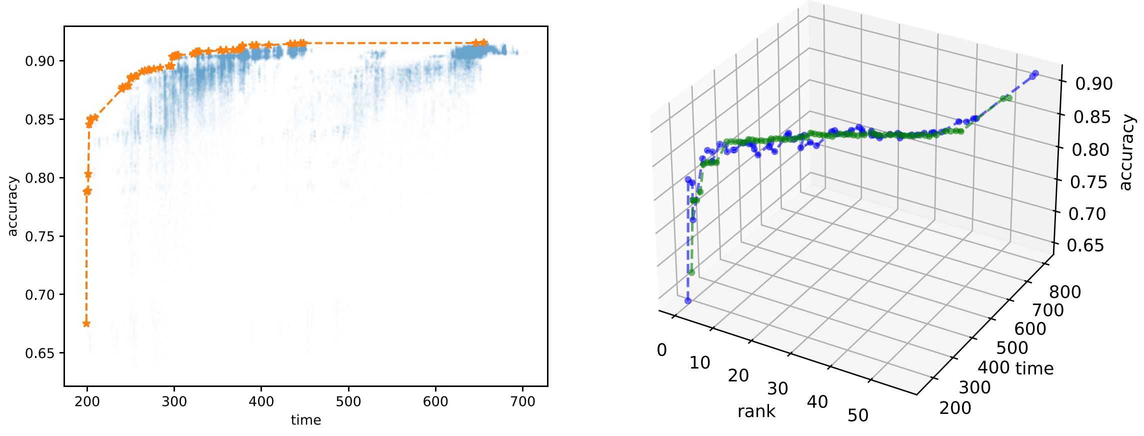


Fig. 4. Pareto plots extrapolated from the CIFAR10 experiment on architectures with cells composed of 5 blocks. (a) The scatter plot of the predictors' estimates for each candidate network evaluated during the extension from 4 to 5 blocks. Each blue semi-transparent pixel represents a different cell specification. Opaque point clouds indicate a large number of architectures with similar results. POPNASv3 selects for training only those architectures that belong to the Pareto front, here represented by the orange star symbols. (b) A comparison of the real and estimated Pareto fronts. The blue line shows the real accuracy and training time obtained after proxy training, while the green line shows the estimated values given by the predictors.

Table 6
Comparison between POPNASv3 and PNAS search performance on the evaluated IC datasets. The accuracies shown are those obtained on the validation set after proxy training. The number of networks and the search time in brackets refer to steps with $b > 2$, excluding the steps $b <= 1$, which are deterministic and equivalent between the two methods.

Dataset	Method	# Networks	Top Cell Accuracy	Top-5 Cells Accuracy	Search time
CIFAR10	POPNASv3	**520** (219)	0.918	0.917	**46h 48m** (29h 19m)
	PNAS	813 (512)	**0.922**	**0.920**	176h 30m (158h 12m)
CIFAR100	POPNASv3	**539** (238)	**0.706**	**0.701**	**55h 5m** (37h 49m)
	PNAS	813 (512)	0.680	0.679	161h 47m (143h 7m)
Fashion MNIST	POPNASv3	**474** (173)	**0.952**	**0.950**	**44h 0m** (26h 3m)
	PNAS	813 (512)	0.946	0.945	174h 53m (155h 23m)
EuroSAT	POPNASv3	**570** (269)	**0.968**	**0.967**	**83h 53m** (53h 35m)
	PNAS	813 (512)	0.967	0.964	246h 14m (217h 45m)

Table 7
The comparison between POPNASv3 and PNAS top-1 networks test performance over the evaluated IC datasets.

Dataset	Method	B	M	N	F	Params	Accuracy	Training time
CIFAR10	POPNASv3	4	3	2	36	3.46M	**0.961**	**5h 22m**
	PNAS	5	3	3	24	**3.25M**	**0.961**	11h 25m
CIFAR100	POPNASv3	5	3	2	24	4.08M	**0.799**	**5h 13m**
	PNAS	5	3	2	24	**4.00M**	0.794	5h 24m
Fashion MNIST	POPNASv3	2	3	2	36	**3.11M**	**0.957**	**2h 38m**
	PNAS	4	3	2	24	3.14M	0.956	6h 3m
EuroSAT	POPNASv3	4	3	3	20	3.37M	0.985	**9h 41m**
	PNAS	5	3	2	24	**3.34M**	**0.987**	13h 35m

macro-architecture tuning to facilitate comparison with similar architectures designed for mobile environments. Finally, the best combination of cell and macro configuration obtained during model selection was trained to convergence. The final training results and their macro-architecture settings are summarised in Table 7, while the cell structures are illustrated in Fig. B.1. The training time required for POPNASv3 architectures is always lower than for PNAS, even when the number of parameters is almost identical. This result is not surprising, since the training time is optimised by POPNAS during the search, but not by PNAS. The overall training results of the entire search show that parameters and FLOPs have a strong correlation with training time, but architectures with similar parameters and FLOPs can actually have remarkably different training times. Some operators, in particular depthwise-separable and spatial-separable convolutions, appear to be less optimised for GPU parallelisation because they consist of two separate convolutions executed sequentially. These types of convolutions are therefore slower to perform than normal convolutions, even though they have fewer parameters and FLOPs for the same kernel size. This behaviour could vary depending on the training hardware, but was consistent in all our experiments on A100 GPUs. The accuracy of the architectures found by PNAS and POPNASv3 are almost identical, meaning that the speed-ups

Table 8
POPNASv3 search performance over the evaluated TSC datasets. The listed accuracies are the ones obtained on the validation set after the proxy training.

Dataset	# Networks	Top Cell Accuracy	Top-5 Cells Accuracy	Search time
ElecticDevices	574	0.919	0.917	17h 42m
FordA	439	0.970	0.968	7h 4m
FaceDetection	453	0.791	0.790	6h 50m
LSST	481	0.663	0.645	5h 15m

Table 9
POPNASv3 top-1 networks performance over the default test split of the evaluated TSC datasets.

Dataset	B	M	N	F	Params	Accuracy	Training time
ElectricDevices	4	3	4	24	1.41M	0.753	37m 4s
FordA	4	4	2	20	1.85M	0.958	6m 20s
FaceDetection	3	3	4	24	1.01M	0.676	9m 32s
LSST	4	3	4	24	1.80M	0.721	11m 40s

Table 10
Accuracy comparison between POPNASv3 and state-of-the-art methods on TSC datasets, using the original train-test split.

Dataset	HIVE-COTEv2 [25]	InceptionTime [28]	ROCKET [29]	POPNASv3
ElectricDevices	**0.758**	0.722	0.726	0.753
FordA	0.954	**0.964**	0.946	0.958
FaceDetection	0.660	–	0.644	**0.676**
LSST	0.643	0.612	0.637	**0.721**

of our method are not offset by performance losses in operational models.

4.4. Time series classification results

Time series classification experiments were only performed using the POPNASv3 search strategy, as PNAS was not designed to work on time series. In Section 4.4.1, we analyse the results on the datasets considered for the preliminary tests that we carried out to calibrate the POPNASv3 hyperparameters and to validate the architecture design with respect to the new RNN operators introduced for the TSC task. The results of the experiment on the extended selection of datasets from the UCR and UEA archives are instead reported in Section 4.4.2.

4.4.1. Preliminary experiments

The datasets selected for the preliminary tests, listed in Table 2, have different data types (i.e. electricity consumption, audio, electroencephalograms, astronomical data) and are characterised by a large number of samples with different dimensionality. The large availability of data guarantees more stability in the final results, while the consideration of different data domains and input resolutions favours the evaluation of the generalisation capabilities of the POPNASv3 procedure to different TSC tasks. Table 8 lists the results of the search process. Similar to what we observed in the image classification cases, POPNASv3 uses Pareto optimisation to train a limited number of architectures and quickly converges to optimal regions of the search space. On average, there is minimal variance between the accuracy of the top 5 cell specifications, meaning that POPNASv3 found several competitive architectures for the model selection step. The search time required for the time series classification experiments is almost an order of magnitude lower than for the image classification tasks, which is justified by the fact that these datasets have fewer samples and generally less data to process per sample. In our opinion, POPNASv3 is a valuable technique for exploring different operator sets and architectures for time series classification tasks, given the appealing computational time to run the experiments.

Also for this task, we finalise the search process by performing model selection, using a parameter range of $(1 \times 10^6, 2 \times 10^6)$, and training the best cell and macro-architecture combination to convergence. The results are presented in Table 9, while the cell specifications are illustrated in Fig. B.2. From the accuracy results, it is clear that the best validation accuracies found over the 90%–10% train-validation split are overly optimistic compared to the real capabilities of the networks. Indeed, the accuracy found on the test sets after prolonged training tends to be lower than the validation accuracy obtained after proxy training. This behaviour suggests that the models can converge even during proxy training, but the validation accuracy could be misleading due to overfitting on the small sample size or strong correlations between training and validation samples, e.g. when the samples are windows of the same signal, possibly overlapping. Expanding the validation split could lead to more accurate estimates at search time, but reducing the number of training samples in small datasets could negatively affect the learning process. An alternative approach is to perform k-folds, as in NAS-T [30], at the cost of multiplying the search time by the number of folds used.

Regardless of the validation-test discrepancy, the final accuracy results obtained on the default train–test split of these datasets are remarkable and competitive with state-of-the-art algorithms endorsed by the TSC community such as InceptionTime [28], HIVE-COTEv2 [25] and ROCKET [29]. The comparison with these algorithms is shown in Table 10. POPNASv3 achieves the best accuracy on the two multivariate datasets considered, but while the gap is not significant on FaceDetection, we achieve far better results on LSST compared to the other methods. RNN operators seem to be the key to LSST performance; in fact, the top 30% of cells sampled during the search use LSTM or GRU as an operator at least once, with GRU often outperforming LSTM. Therefore, the combination of RNN cells with 1D convolutional operators appears to be beneficial for improving model performance on some TSC problems. In the literature, this type of architecture goes under the name of Convolutional Recurrent Neural Networks (CRNN), and it has seen application in many TSC works [49,50]. However, CRNNs typically structure the network to extract features using only convolutions and aggregate them in the final layers with the RNN units. Applying the RNN units directly to the input sequences and within the architecture, as done in POPNASv3 cells, works well in our experiments, suggesting that more flexible CRNN structures should also be considered for these tasks.

Dilated convolutions perform well in all datasets except ElectricDevices, but this behaviour can be explained by analysing the samples contained in this dataset. The ROCKET authors provided interesting insights into how dilated convolutions can be understood as a way of sampling the signal at different frequencies. This behaviour can be advantageous for finding patterns over a wide receptive range on slowly evolving signals, and the contribution of these operators has been statistically shown to increase the average performance of the ROCKET classifier. This is also the case for POPNASv3, but here the architectures are adapted to use the best operators for the data. ElectricDevices contains data on the power consumption of household appliances. Several classes of appliances have long flat sections at 0 in the signal, presumably because they are unused and unconnected during these time windows, and are therefore characterised by short peaks of varying amplitude. By sampling the ElectricDevices dataset at lower frequencies, these short peaks are not detected, so POPNASv3 discards dilated convolutions and uses only non-dilated ones. These considerations show how using NAS on large search spaces with different operators can lead to interesting architectures for the given problem, in a completely black-box fashion. We argue that the purpose of NAS should not be to focus only on achieving the best possible accuracy for the task, but to design algorithms that have the freedom to find unconventional architectures that may go beyond the current knowledge we have on neural network architecture design.

The best cells found on these datasets are shown in Fig. B.2 of Appendix B. All cells tend to structure the computational graph in multiple layers, processing some of the intermediate block results with other operators to rapidly increase the receptive field and aggregate the features extracted by previous layers. This structural behaviour is not observed in the image classification cells, where "flat" cells perform well simply by applying all operators in parallel. Furthermore, all the TSC cells discovered use only −2 lookback, and therefore the architectures stack fewer cells than the target number defined in the POPNASv3 configuration, indicating that the potential performance gain from adding more cells is negligible compared to the increased computational cost. The operators selected in these cells are highly dependent on the characteristics of the dataset, as we have analysed previously, and they are related to each other without any discernible patterns.

4.4.2. *Experiments on the UCR and UEA archives*

As mentioned above, we extended the experiments to a larger selection of datasets from the UCR and UEA archives, filtering out the datasets with fewer than 360 samples. We also include in this second analysis the four datasets used for design validation. The full results for all these datasets are presented in Appendix B.2. The base configuration used for these experiments is that given in Table 3, but the batch size and validation size are automatically tuned for each dataset. The fraction of training samples reserved for validation is adjusted to have at least 60 samples in the validation split. In this way, we reduce the variance of the validation accuracy in smaller datasets without excessively reducing the training size. The batch size is assigned based on the number of samples in the training split, as a maximum multiplier of 16 in the interval $[32, 128]$, so that each epoch processes at least 10 batches of data. This setting strikes a balance between training time and convergence stability. Larger batch sizes improve training efficiency, but reduce the number of weight updates, which also affects the smoothness of the learning rate scheduler and the scheduled drop path, potentially leading to underfitting.

Table B.1 summarises the results of POPNASv3 searches on the extended selection of datasets. Most datasets complete the search process in less than 10 h, sampling between 400 and 500 architectures. The average validation accuracy of the 5 best cells found on each dataset is usually close to the top architecture. Considering the higher expected variance in the results due to the lower data availability in the validation split, this observation further confirms the stability of POPNASv3 in finding multiple viable solutions. The final architectures obtained after model selection with params range $(1 \times 10^6, 3 \times 10^6)$, listed in Table B.2, are quite heterogeneous in both structure and operator usage, confirming the observations made during the preliminary tests. In summary, 29 of the best architectures found on the 49 datasets considered use only skip connections as lookback, stacking a smaller number of cells, but POPNASv3 often reintroduces some of them by increasing the M and N hyperparameters during model selection. Only two groups of datasets, namely (Adiac, EthanolLevel) and (FordB, EOGVerticalSignal, UWaveGestureLibraryZ) share the same cells, while other datasets have unique architectures.

The comparison between the test accuracy found by POPNASv3 and the state-of-the-art methods considered in the TSC literature is presented in Table B.3. On all four multivariate datasets, POPNASv3 is able to find better solutions than the other methods, indicating that the neural networks found by our algorithms are well suited to extract correlations from multiple time series. In addition to LSST, which we have analysed in previous experiments, our method also shows a significant improvement on the PhonemeSpectra dataset over the other models considered, achieving a 32% relative gain in accuracy over HIVE-COTEv2. The architecture found on this complex dataset, shown in Fig. 5, is particularly interesting, as POPNASv3 mixes seven different operators in a convoluted DAG structure, connecting blocks in multiple paths. Multi-branched asymmetric architectural patterns are never explored in hand-crafted models, but these particular solutions could be interesting in various applications, and are easier to explore with NAS.

On univariate datasets, the performance of POPNASv3 deteriorates, often finding single-block architectures that are not always competitive with state-of-the-art solutions, even on the simplest datasets. We aggregate the results on all datasets with the mean rank of each technique considered together with POPNASv3, represented with a critical difference diagram [51] using the Wilcoxon signed-rank test with Holm alpha correction [52,53]($\alpha = 0.05$), shown in Fig. 6.[2] The null hypothesis is set to "the algorithms perform equally well". The Wilcoxon signed-rank test is used to compare the performance of each pair of methods, calculating their differences in accuracy for each dataset and ranking them based on the absolute values of these differences. The test ignores the absolute magnitude of the accuracy gap between the two methods, making it resistant to outliers. A thick black line, i.e. a clique, indicates a group of methods whose results are not statistically different for the Wilcoxon signed-rank test, i.e. the null hypothesis is not rejected between each pair of algorithms connected by the clique. POPNASv3 achieves the second best average rank, with similar results to InceptionTime and ROCKET. The Wilcoxon signed-rank test shows that the results of POPNASv3 are statistically competitive with HIVE-COTEv2 (p-value $= 0.127$), even if HIVE-COTEv2 has a significantly better ranking on the whole selection.

As indicated above, POPNASv3 finds monoblock cells in about half of the considered datasets, which is not inherently bad for easy-to-learn problems, but we suspect that the algorithm did not converge to optimal solutions in some cases. In fact, the limited number of samples contained in these datasets may hinder search convergence, making the process prone to significant randomness. The validation size on small datasets is not sufficient to properly estimate the quality of a network: when only dozens of samples are available for validation, repeating the proxy training of the same architecture several times can give very different accuracy results. The uncertainty of these measurements strongly affects the accuracy predictor, which becomes unable to discriminate architectures within the uncertainty range, negatively affecting the Pareto front extracted by the algorithm. This is a problem common to all machine learning techniques, but it is particularly difficult to address in end-to-end solutions such as NAS, and we recognise that POPNASv3 is not well suited for general use on small datasets. On datasets with larger numbers of samples and on multivariate datasets, POPNASv3 architectures have competitive results, even significantly better in multivariate cases compared to state-of-the-art methods, and are characterised by interesting internal motifs.

[2] The figure was generated using the open-source code provided in https://github.com/hfawaz/cd-diagram [54].

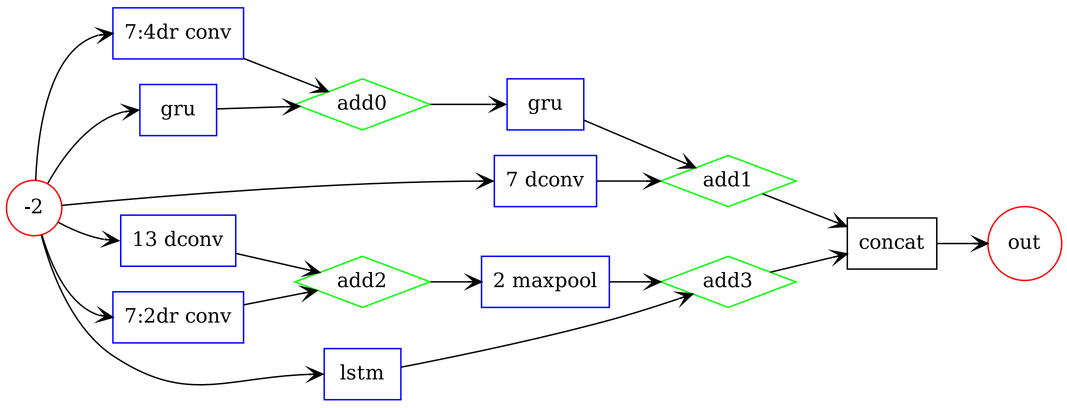


Fig. 5. Cell structure found on PhonemeSpectra dataset.

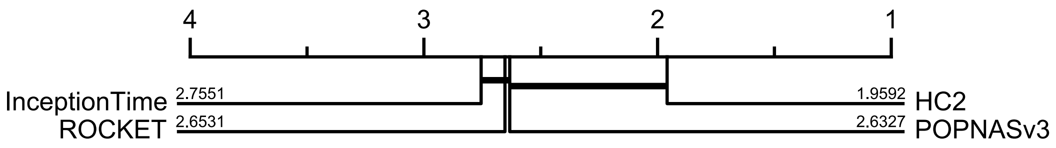


Fig. 6. Critical difference diagram on the extended TSC dataset selection. The number associated with each method represents the mean rank obtained over the dataset selection.

5. Conclusions

In this article, we introduced POPNASv3, a sequential model-based optimisation method that can work with potentially unlimited sets of operators, while maintaining competitive search time and accuracy results with other state-of-the-art methods. The POPNASv3 search procedure is flexible in terms of hardware environments and computational budgets, as the complexity of the search space and the macro-architecture of the models can be adapted to the hardware capabilities. By starting the search by training the simplest models in the search space, the algorithm can identify the most promising operators and estimate their computational efficiency on the training device. The predictors use the training results of previously trained networks and guide the search process to sample the most promising cells in subsequent iterations. Our approach restricts the search to architectures belonging to the Pareto front, providing a significant efficiency gain over similar methods and allowing exploration of large search spaces in a short time. Our method can address image and time series classification tasks without modification, as the operators are adapted to the dimensionality of the input and reshaped to build consistent architectures composed of different operator types. Thus, we demonstrate that our methodology can be extended to other tasks by simply adapting the search space, while keeping the search strategy and overall training procedure unchanged.

The experiments show that POPNASv3 can determine the most appropriate operators to process each dataset and converge to the optimal regions of the search space by sampling a minimal number of architectures. By exploring large assorted operator sets on the time series classification datasets, POPNASv3 was able to find innovative architectures that mix convolutions, dilated convolutions, pooling and recurrent neural network units. After the search, the macro-architecture of the best model can be automatically fine-tuned and retrained from scratch to maximise its performance for final deployment. The performance of these models is competitive with the state-of-the-art, confirming the potential of POPNASv3 to discover unconventional motifs not often considered in the literature, and its ability to enable rapid model prototyping in industry.

CRediT authorship contribution statement

Andrea Falanti: Conceptualization, Methodology, Software, Visualization, Writing – original draft. **Eugenio Lomurno:** Conceptualization, Writing – review & editing. **Danilo Ardagna:** Project administration, Resources, Writing – review & editing. **Matteo Matteucci:** Project administration, Resources, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The code and the data are available at the Zenodo link inside the footnote of the first page of the paper

Acknowledgements

The European Commission has partially funded this work under the H2020 grant N. 101016577 AI-SPRINT: AI in Secure Privacy-pReserving computINg conTinuum.

Appendix A

A.1. Predictors accuracy on different hardware

The POPNASv3 algorithm supports different hardware configurations for training the candidate neural network models. The underlying training procedure is performed using the distribution strategies implemented in the TensorFlow API, allowing our

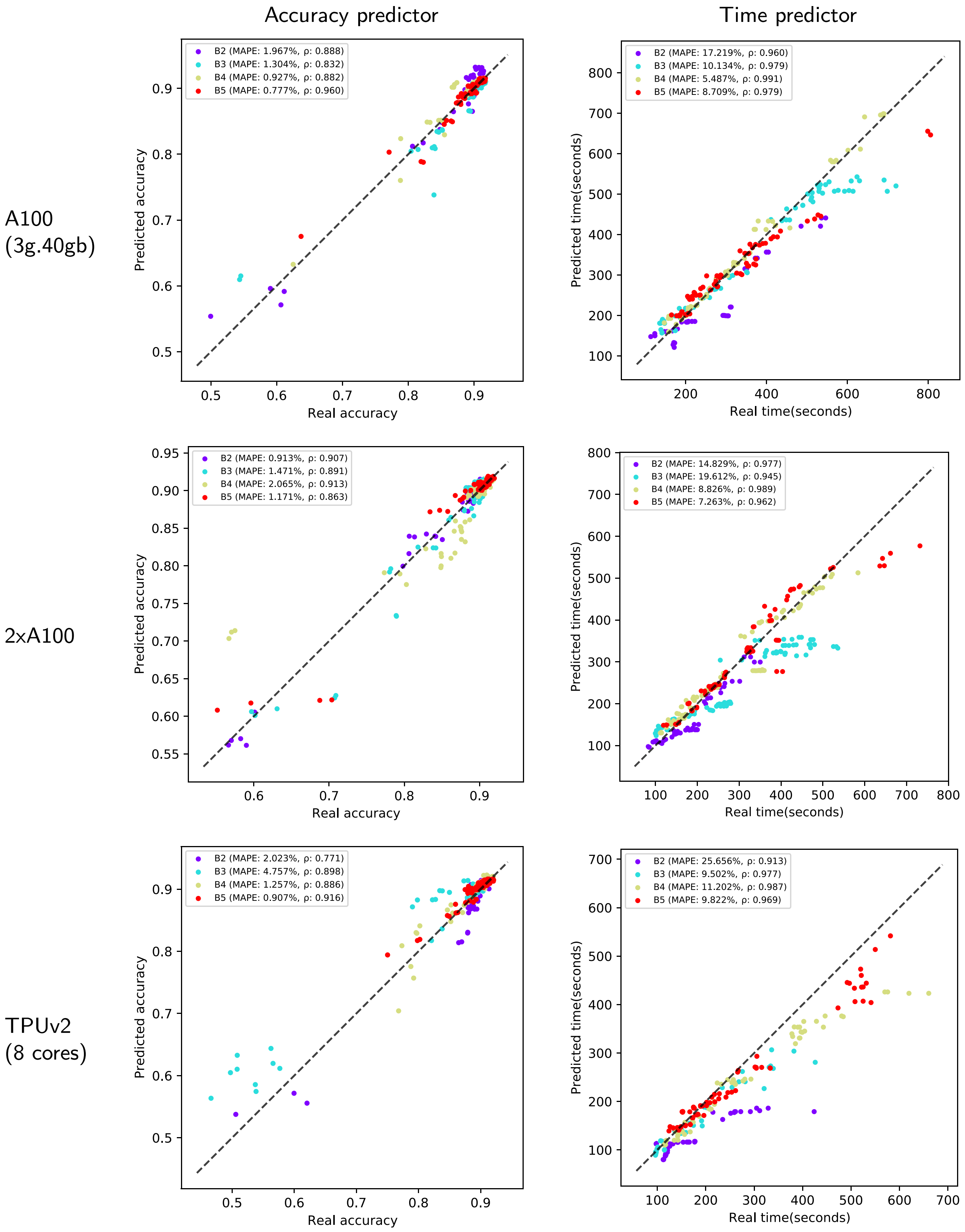


Fig. A.1. Scatter plots representing the predicted accuracy and training time compared to the real ones retrieved after training, for each hardware-related experiment.

Table A.1
Comparison of POPNASv3 accuracy predictor results on CIFAR10 experiments running on different hardware environments.

Device	MAPE(%)				Spearman(ρ)			
	$b=2$	$b=3$	$b=4$	$b=5$	$b=2$	$b=3$	$b=4$	$b=5$
A100 80GB (MIG 3g.40gb)	1.967	1.304	0.927	0.777	0.888	0.832	0.882	0.960
2xA100 80GB	0.913	1.471	2.065	1.171	0.907	0.891	0.913	0.863
TPU v2–8	2.023	4.757	1.257	0.907	0.771	0.898	0.886	0.916

Table A.2
Comparison of POPNASv3 time predictor results on CIFAR10 experiments running on different hardware environments.

Device	MAPE(%)				Spearman(ρ)			
	$b=2$	$b=3$	$b=4$	$b=5$	$b=2$	$b=3$	$b=4$	$b=5$
A100 80GB (MIG 3g.40gb)	17.219	10.134	5.487	8.709	0.960	0.979	0.991	0.979
2xA100 80GB	14.829	19.612	8.826	7.263	0.977	0.945	0.989	0.962
TPU v2–8	25.656	9.502	11.202	9.822	0.913	0.977	0.987	0.969

algorithm to train the architectures on a single GPU, multiple GPUs, or even TPU devices. The training time of the models can vary in a non-linear way with the available computational power of the hardware, so we performed some experiments to study the robustness of our time predictor and feature set to different hardware environments. The predictors are fundamental in guiding the algorithms to explore the most promising regions of the search space, since the Pareto front depends on the correctness of the ranking of the evaluated architectures. Therefore, the predictors must be able to correctly rank the results not only for different tasks, but also for different training techniques.

We performed three experiments on the CIFAR10 dataset, training the networks on different hardware configurations and adjusting the learning rate and batch size according to the device. The first experiment is the one described in Section 4.3, performed on a single A100 MIG partition (3g.40gb profile) using the default hyperparameters listed in Table 3. The other two experiments are performed on 2 A100 GPUs and an 8-core TPUv2 machine, respectively, with the batch size increased to 256 in both experiments to make better use of the available devices. We observed in a previous ablation study on the TPU that increasing the batch size to 512 or more leads to generally worse accuracy in all networks, so we limit it to 256 even though TPUs can exploit much higher parallelisation. The initial learning rate is set to 0.02, following the heuristic of multiplying the learning rate by the same factor as the batch size.

The results of the predictors for these experiments are presented in Tables A.1 and A.2. Fig. A.1 provides a visual comparison of the results in the form of scatter plots showing the predicted values compared to the actual values. The accuracy achieved by the networks should not depend on the hardware used for training, but may be slightly affected by changes in batch size and learning rate. The accuracy predictor has a similar performance in the three experiments due to the low variance of the accuracy achieved by the architectures, which is more due to the weight initialisation and non-deterministic operations performed in parallel architectures than to the hardware configuration itself. The MAPE per step is negligible and the Spearman's rank correlation coefficient obtained is significantly high in all configurations, indicating that the accuracy predictor is suitable for the task.

Instead, the results of the time predictor are more relevant, since the training time speed-up could be different for each architecture and therefore the predictor needs to adapt to the hardware environment. If we analyse the architectures with single block cells, which are fixed by our training procedure and therefore the same in all experiments, we find an average speed-up of 1.47 ± 0.09 between using a partitioned A100 and two A100s with replicated strategy, while using TPU increases this factor to a speed-up of 1.69 ± 0.26. In both cases, the standard deviation of the speed-ups is not negligible, which means that the time predictor must adapt to a non-linear speed-up between architectures. The dynamic reindex formula is tuned to the results obtained at runtime, allowing the predictor to learn the training time discrepancies between operators in different hardware environments. The results show that the time predictor achieves a Spearman rank coefficient close to 1 in every step for all experiments. The training time ranking is therefore optimal even on different training devices, demonstrating that our method can work well for heterogeneous hardware and training strategies.

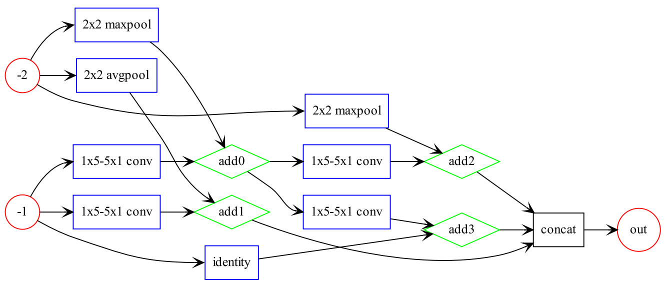
Appendix B

B.1. POPNASv3 top cells

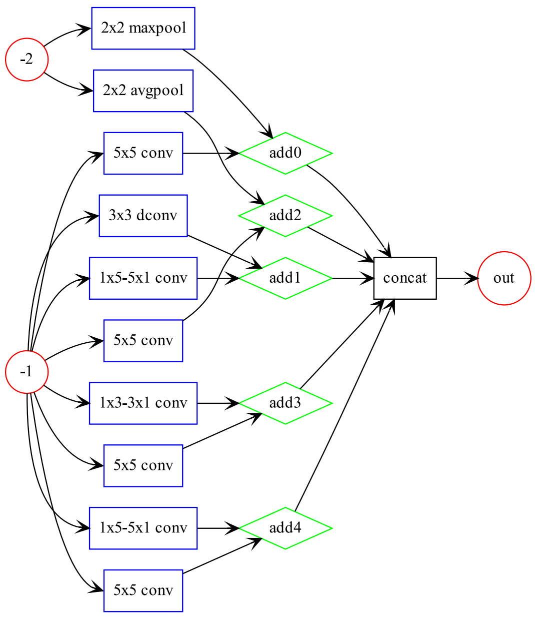
Here we report an illustration of all the top-1 cell structures found by POPNASv3 on the datasets considered in the experiments section.

B.2. Extended results on UCR/UEA archives

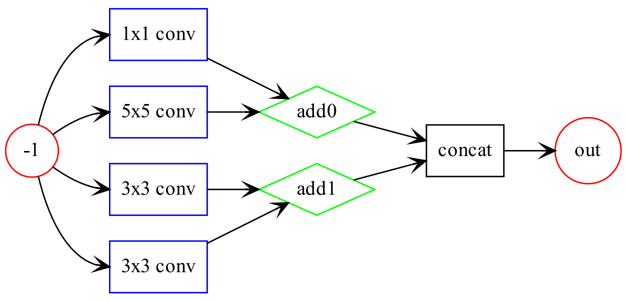
This section presents the results of the experiments performed on the extended selection of time series classification datasets, composed of 45 univariate datasets and 4 multivariate ones. These datasets contain at least 360 samples in the training set, and their samples have equal lengths. The results have been analysed in Section 4.4, here we just report a summary of the results under multiple tables. Multivariate datasets are separated from the univariate datasets and presented at the bottom of each table.



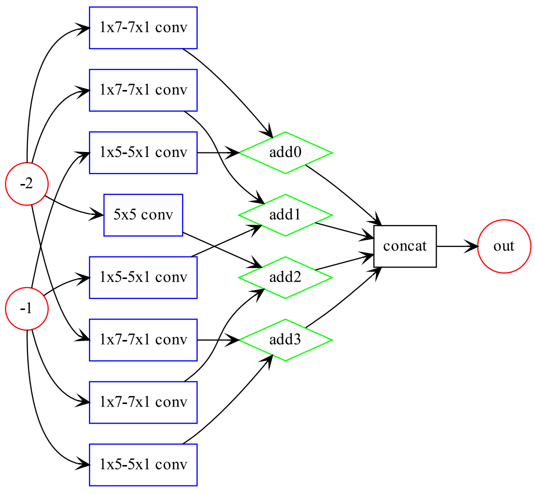
(a) CIFAR10



(b) CIFAR100

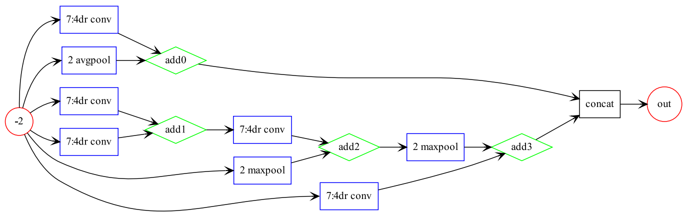


(c) Fashion MNIST

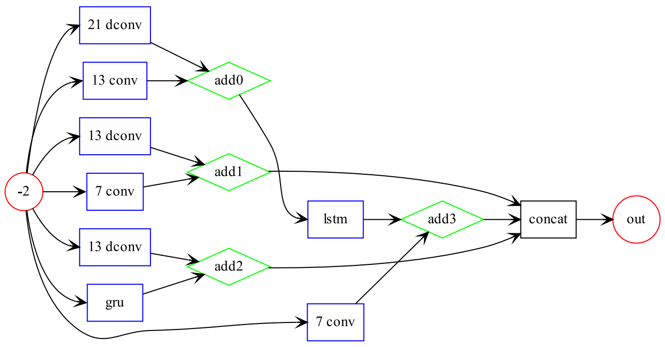


(d) EuroSAT

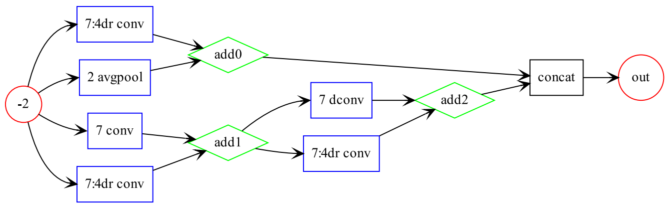
Fig. B.1. Cell motifs used in the best architectures found on image classification datasets.



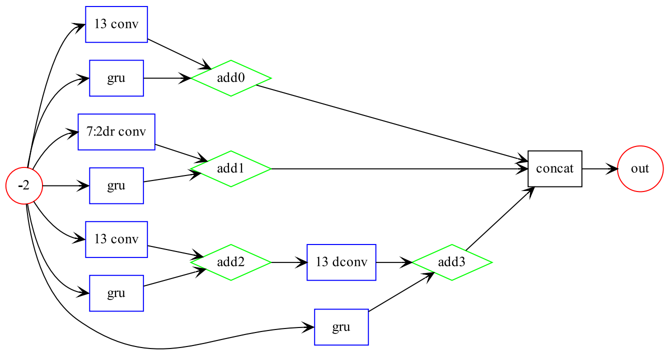
(a) Ford A



(b) ElectricDevices



(c) FaceDetection



(d) LSST

Fig. B.2. Cell motifs used in the best architectures found on time series classification datasets of the preliminary tests.

Table B.1
Search results and top validation accuracies found on the extended UCR/UEA datasets selection.

Dataset	# Networks	Top accuracy	Top-5 Cells Accuracy	Search time
Adiac	417	0.556	0.530	3h 10m
ChlorineConcentration	404	0.639	0.630	3h 23m
CricketX	478	0.825	0.816	4h 18m
CricketY	501	0.857	0.838	5h 1m
CricketZ	445	0.778	0.752	3h 46m
Crop	496	0.775	0.772	8h 9m
DistalPhalanxOutlineAgeGroup	428	0.900	0.883	3h 28m
DistalPhalanxOutlineCorrect	420	0.950	0.937	3h 12m
DistalPhalanxTW	417	0.850	0.840	3h 14m
ECG5000	432	0.983	0.970	3h 59m
EOGHorizontalSignal	509	0.919	0.877	13h 28m
EOGVerticalSignal	525	0.677	0.665	14h 21m
ElectricDevices	507	0.920	0.918	13h 55m
EthanolLevel	446	0.525	0.502	6h 37m
FaceAll	447	1.000	1.000	3h 35m
FiftyWords	451	0.844	0.819	4h 13m
FordA	475	0.970	0.966	7h 38m
FordB	467	0.967	0.965	7h 19m
HandOutlines	482	0.920	0.904	13h 29m
LargeKitchenAppliances	487	1.000	0.987	7h 46m
MedicalImages	477	0.787	0.780	4h 18m
MiddlePhalanxOutlineAgeGroup	413	0.833	0.827	3h 9m
MiddlePhalanxOutlineCorrect	425	0.867	0.850	3h 31m
MiddlePhalanxTW	405	0.734	0.728	3h 8m
NonInvasiveFetalECGThorax1	504	0.811	0.759	6h 35m
NonInvasiveFetalECGThorax2	456	0.861	0.853	5h 48m
PhalangesOutlinesCorrect	426	0.878	0.859	3h 28m
ProximalPhalanxOutlineAgeGroup	435	0.933	0.933	3h 37m
ProximalPhalanxOutlineCorrect	430	0.933	0.923	3h 19m
ProximalPhalanxTW	473	0.867	0.853	4h 26m
RefrigerationDevices	438	0.783	0.763	5h 52m
ScreenType	428	0.583	0.567	5h 41m
SemgHandMovementCh2	509	0.656	0.641	11h 9m
SemgHandSubjectCh2	516	0.859	0.850	8h 36m
ShapesAll	509	0.883	0.857	6h 44m
SmallKitchenAppliances	446	0.850	0.830	6h 25m
StarLightCurves	511	0.980	0.980	9h 1m
Strawberry	452	0.952	0.948	4h 19m
SwedishLeaf	475	0.933	0.917	4h 27m
TwoPatterns	484	1.000	1.000	4h 42m
UWaveGestureLibraryAll	567	0.978	0.951	9h 4m
UWaveGestureLibraryX	491	0.844	0.827	4h 40m
UWaveGestureLibraryY	488	0.800	0.793	4h 38m
UWaveGestureLibraryZ	421	0.789	0.782	3h 43m
Wafer	465	1.000	1.000	4h 7m
FaceDetection	467	0.800	0.794	7h 25m
LSST	471	0.663	0.647	5h 15m
PenDigits	469	0.999	0.999	7h 56m
PhonemeSpectra	572	0.358	0.351	11h 9m

Table B.2
Final training results on the extended UCR/UEA datasets selection. The macro-architecture parameters and the cell specifications are listed together with the accuracy reached on the default test split.

Dataset	B	M	N	F	Params(M)	Acc.	Cell specification
Adiac	1	3	2	42	2.47	0.821	[(−2, ’gru’, −1, ’21 conv’)]
ChlorineConcentration	1	3	4	48	1.56	0.810	[(−2, ’13 dconv’, −2, ’13 conv’)]
CricketX	4	3	2	48	2.17	0.821	[(−2, ’21 dconv’, −2, ’7:4dr conv’);(−2, ’21 dconv’, −2, ’7:4dr conv’);(−2, ’13 conv’, −2, ’2 avgpool’);(2, ’13 dconv’, 2, ’2 avgpool’)]
CricketY	3	4	2	24	2.30	0.862	[(−2, ’identity’, −2, ’7:4dr conv’);(−2, ’2 maxpool’, 0, ’7:4dr conv’);(0, ’gru’, 1, ’7:4dr conv’)]
CricketZ	1	3	2	24	0.38	0.831	[(−2, ’7:4dr conv’, −1, ’7:4dr conv’)]
Crop	1	3	2	24	0.25	0.782	[(−2, ’21 dconv’, −1, ’7:4dr conv’)]
DistalPhalanxOutlineAgeGroup	1	3	2	24	0.14	0.691	[(−2, ’13 dconv’, −2, ’gru’)]
DistalPhalanxOutlineCorrect	3	4	4	48	2.78	0.793	[(−2, ’identity’, −1, ’2 maxpool’);(−2, ’2 maxpool’, −2, ’2 maxpool’);(1, ’identity’, 1, ’2 maxpool’)]
DistalPhalanxTW	1	3	4	48	1.56	0.712	[(−1, ’7:4dr conv’, −1, ’2 avgpool’)]
ECG5000	1	3	2	24	0.20	0.927	[(−2, ’gru’, −1, ’2 avgpool’)]
EOGHorizontalSignal	5	3	3	42	2.89	0.655	[(−2, ’2 maxpool’, −2, ’2 maxpool’);(0, ’2 maxpool’, 0, ’2 avgpool’);(0, ’7:4dr conv’, 1, ’7:4dr conv’);(2, ’21 dconv’, 2, ’2 maxpool’);(−2, ’21 conv’, −2, ’gru’)]
EOGVerticalSignal	1	3	2	48	1.65	0.591	[(−1, ’7:4dr conv’, −1, ’gru’)]
ElectricDevices	4	3	2	24	2.15	0.744	[(−1, ’7 conv’, −1, ’13 conv’);(−1, ’13 conv’, −1, ’13 conv’);(−1, ’7 conv’, −1, ’gru’);(1, ’7 dconv’, 1, ’gru’)]
EthanolLevel	1	4	2	20	2.31	0.850	[(−2, ’gru’, −1, ’21 conv’)]
FaceAll	1	3	2	24	0.08	0.956	[(−2, ’7 dconv’, −1, ’21 dconv’)]
FiftyWords	3	4	2	24	1.38	0.862	[(−2, ’identity’, −2, ’2 avgpool’);(−2, ’identity’, −2, ’2 avgpool’);(0, ’7:4dr conv’, 1, ’7:4dr conv’)]
FordA	2	4	2	36	2.58	0.953	[(−2, ’identity’, −2, ’7:4dr conv’);(−2, ’identity’, −2, ’7:4dr conv’)]
FordB	1	4	3	24	2.34	0.843	[(−1, ’7:4dr conv’, −1, ’gru’)]
HandOutlines	4	4	2	48	2.63	0.927	1[(−2, ’identity’, −2, ’2 avgpool’);(−2, ’identity’, −2, ’2 avgpool’);(0, ’2 avgpool’, 1, ’gru’);(0, ’identity’, 2, ’identity’)]
LargeKitchenAppliances	5	3	2	48	3.01	0.904	[(−2, ’identity’, −2, ’identity’);(−2, ’identity’, −2, ’identity’);(−2, ’7 conv’, 1, ’7:4dr conv’);(−2, ’21 conv’, 1, ’13 dconv’);(−2, ’21 dconv’, 1, ’13 dconv’)]
MedicalImages	3	3	2	24	0.82	0.784	[(−2, ’7:4dr conv’, −2, ’7:4dr conv’);(−2, ’7 dconv’, 0, ’identity’);(0, ’21 conv’, 1, ’gru’)]
MiddlePhalanxOutlineAgeGroup	1	3	2	24	0.14	0.565	[(−2, ’21 dconv’, −2, ’gru’)]
MiddlePhalanxOutlineCorrect	1	4	2	20	2.27	0.777	[(−2, ’7:4dr conv’, −1, ’21 conv’)]
MiddlePhalanxTW	1	3	2	24	0.42	0.519	[(−2, ’21 dconv’, −1, ’13 conv’)]
NonInvasiveFetalECGThorax1	4	3	2	36	2.97	0.943	[(−2, ’21 conv’, −2, ’2 maxpool’);(−2, ’21 conv’, −2, ’2 maxpool’);(0, ’21 conv’, 0, ’lstm’);(0, ’2 avgpool’, 2, ’2 maxpool’)]
NonInvasiveFetalECGThorax2	4	3	4	24	1.64	0.945	[(−2, ’identity’, −2, ’21 conv’);(−2, ’7:4dr conv’, 0, ’7:4dr conv’);(−2, ’21 dconv’, 0, ’7:4dr conv’);(2, ’21 dconv’, 2, ’7:4dr conv’)]
PhalangesOutlinesCorrect	4	3	2	24	0.26	0.843	[(−2, ’2 maxpool’, −2, ’2 maxpool’);(−2, ’7 conv’, 0, ’2 maxpool’);(0, ’identity’, 1, ’2 maxpool’);(−2, ’7:4dr conv’, 2, ’13 dconv’)]
ProximalPhalanxOutlineAgeGroup	1	4	2	42	3.01	0.849	[(−2, ’13 dconv’, −1, ’7:4dr conv’)]
ProximalPhalanxOutlineCorrect	1	4	2	24	1.81	0.876	[(−2, ’21 conv’, −2, ’7:2dr conv’)]
ProximalPhalanxTW	2	4	2	24	1.61	0.776	[(−2, ’7 dconv’, −2, ’7:2dr conv’);(−2, ’7:2dr conv’, 0, ’7:2dr conv’)]
RefrigerationDevices	1	3	4	48	2.07	0.520	[(−2, ’13 dconv’, −1, ’lstm’)]
ScreenType	1	3	3	48	1.03	0.549	[(−2, ’7 conv’, −2, ’7:4dr conv’)]
SemgHandMovementCh2	2	3	2	24	0.26	0.771	[(−2, ’2 avgpool’, −2, ’2 avgpool’);(−2, ’7:2dr conv’, 0, ’gru’)]
SemgHandSubjectCh2	3	4	2	24	2.13	0.878	[(−2, ’7:2dr conv’, −2, ’2 maxpool’);(−2, ’2 avgpool’, 0, ’7:4dr conv’);(−2, ’7:2dr conv’, 1, ’gru’)]
ShapesAll	4	3	4	20	2.49	0.897	[(−2, ’21 conv’, −1, ’gru’);(−2, ’21 dconv’, −1, ’21 dconv’);(−2, ’21 conv’, 1, ’gru’);(−2, ’13 dconv’, −1, ’lstm’)]
SmallKitchenAppliances	1	3	2	24	0.57	0.803	[(−2, ’gru’, −1, ’13 conv’)]
StarLightCurves	1	3	2	24	0.24	0.977	[(−2, ’21 dconv’, −1, ’gru’)]
Strawberry	5	3	4	24	1.45	0.970	[(−2, ’7:4dr conv’, −2, ’7:4dr conv’);(−2, ’7:2dr conv’, −2, ’gru’);(0, ’2 maxpool’, 0, ’gru’);(0, ’identity’, 0, ’gru’);(2, ’2 maxpool’, 3, ’gru’)]
SwedishLeaf	3	3	4	24	2.67	0.962	[(−2, ’13 conv’, −2, ’21 conv’);(−2, ’13 conv’, −2, ’21 conv’);(−2, ’13 conv’, −2, ’13 conv’)]
TwoPatterns	1	3	2	24	0.03	1.000	[(−2, ’identity’, −2, ’7 dconv’)]
UWaveGestureLibraryAll	3	3	2	24	0.91	0.970	[(−2, ’7 conv’, −2, ’lstm’);(−2, ’7:4dr conv’, 0, ’7:4dr conv’);(−2, ’13 conv’, 0, ’7:4dr conv’)]
UWaveGestureLibraryX	4	3	2	24	0.56	0.824	[(−2, ’identity’, −2, ’2 maxpool’);(0, ’2 maxpool’, 0, ’2 avgpool’);(−2, ’7:4dr conv’, 1, ’7:4dr conv’);(−2, ’7:4dr conv’, 0, ’gru’)]
UWaveGestureLibraryY	5	3	2	36	2.36	0.767	[(−2, ’2 maxpool’, −2, ’2 maxpool’);(−2, ’2 avgpool’, 0, ’7:4dr conv’);(−2, ’7:4dr conv’, 0, ’7:4dr conv’);(−2, ’7:4dr conv’, 1, ’7:4dr conv’);(−2, ’21 conv’, 3, ’2 maxpool’)]
UWaveGestureLibraryZ	1	3	4	48	2.91	0.745	[(−1, ’7:4dr conv’, −1, ’gru’)]
Wafer	1	3	2	24	0.03	0.998	[(−2, ’identity’, −2, ’13 dconv’)]
FaceDetection	4	3	2	42	2.36	0.675	[(−2, ’7:4dr conv’, −2, ’2 avgpool’);(−2, ’7:4dr conv’, 0, ’2 avgpool’);(−2, ’7:4dr conv’, 1, ’7 dconv’);(−2, ’13 conv’, 2, ’7:4dr conv’)]
LSST	3	4	2	24	2.65	0.723	[(−2, ’7:4dr conv’, −2, ’gru’);(−2, ’7:4dr conv’, −2, ’gru’);(−2, ’13 dconv’, −2, ’gru’)]
PenDigits	1	3	3	48	1.27	0.989	[(−2, ’13 dconv’, −1, ’gru’)]
PhonemeSpectra	4	4	3	20	2.26	0.382	[(−2, ’7:4dr conv’, −2, ’gru’);(−2, ’7 dconv’, 0, ’gru’);(−2, ’13 dconv’, −2, ’7:2dr conv’);(−2, ’lstm’, 2, ’2 maxpool’)]

Table B.3
Comparison between POPNASv3 accuracy and state-of-the-art methods on the extended UCR/UEA datasets selection. The test accuracy is computed on the default test split. The rank achieved by POPNASv3 on each dataset is reported in brackets after the accuracy value.

Dataset	HC2	InceptionTime	ROCKET	POPNASv3
Adiac	0.813	**0.849**	0.785	0.821 (2)
ChlorineConcentration	0.771	**0.876**	0.812	0.810 (3)
CricketX	0.826	**0.846**	0.828	0.821 (4)
CricketY	0.851	0.849	0.856	**0.862** (1)
CricketZ	**0.864**	0.854	0.854	0.831 (4)
Crop	0.765	**0.797**	0.751	0.782 (2)
DistalPhalanxOutlineAgeGroup	**0.748**	0.734	**0.748**	0.691 (4)
DistalPhalanxOutlineCorrect	0.775	**0.793**	0.772	**0.793** (1)
DistalPhalanxTW	0.705	0.669	**0.719**	0.712 (2)
ECG5000	**0.947**	0.940	0.946	0.927 (4)
EOGHorizontalSignal	**0.657**	0.633	0.649	0.655 (2)
EOGVerticalSignal	0.569	0.511	0.555	**0.591** (1)
ElectricDevices	**0.758**	0.722	0.726	0.744 (2)
EthanolLevel	0.696	0.842	0.578	**0.850** (1)
FaceAll	0.868	0.796	0.940	**0.956** (1)
FiftyWords	0.833	0.848	0.835	**0.862** (1)
FordA	0.954	**0.964**	0.946	0.953 (3)
FordB	0.831	**0.857**	0.806	0.843 (2)
HandOutlines	**0.938**	–	–	0.927 (2)
LargeKitchenAppliances	**0.907**	0.899	0.893	0.904 (2)
MedicalImages	**0.808**	0.795	0.799	0.784 (4)
MiddlePhalanxOutlineAgeGroup	**0.597**	0.558	0.584	0.565 (3)
MiddlePhalanxOutlineCorrect	**0.852**	0.828	0.828	0.777 (4)
MiddlePhalanxTW	**0.558**	0.506	**0.558**	0.519 (3)
NonInvasiveFetalECGThorax1	**0.947**	–	–	0.943 (2)
NonInvasiveFetalECGThorax2	**0.967**	–	–	0.945 (2)
PhalangesOutlinesCorrect	0.831	**0.857**	0.831	0.843 (2)
ProximalPhalanxOutlineAgeGroup	**0.854**	0.849	**0.854**	0.849 (3)
ProximalPhalanxOutlineCorrect	0.900	**0.931**	0.900	0.876 (4)
ProximalPhalanxTW	**0.829**	0.771	0.805	0.776 (3)
RefrigerationDevices	0.539	0.528	**0.544**	0.520 (4)
ScreenType	0.579	**0.581**	0.499	0.549 (3)
SemgHandMovementCh2	**0.856**	0.569	0.653	0.771 (2)
SemgHandSubjectCh2	**0.902**	0.764	0.869	0.878 (2)
ShapesAll	0.922	**0.928**	0.910	0.897 (4)
SmallKitchenAppliances	**0.837**	0.773	0.813	0.803 (3)
StarLightCurves	**0.982**	0.978	0.980	0.977 (4)
Strawberry	0.976	**0.984**	0.981	0.970 (4)
SwedishLeaf	0.965	**0.971**	0.970	0.962 (4)
TwoPatterns	**1.000**	**1.000**	**1.000**	**1.000** (1)
UWaveGestureLibraryAll	0.974	0.950	**0.975**	0.970 (3)
UWaveGestureLibraryX	**0.855**	0.824	0.854	0.824 (3)
UWaveGestureLibraryY	**0.775**	0.767	0.772	0.767 (3)
UWaveGestureLibraryZ	**0.797**	0.768	0.796	0.745 (4)
Wafer	**1.000**	0.999	0.998	0.998 (3)
FaceDetection	0.660	–	0.644	**0.675** (1)
LSST	0.643	0.612	0.637	**0.723** (1)
PenDigits	0.979	0.988	0.983	**0.989** (1)
PhonemeSpectra	0.290	–	0.276	**0.382** (1)

References

[1] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, 2016.

[2] L. Alzubaidi, J. Zhang, A.J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M.A. Fadhel, M. Al-Amidie, L. Farhan, Review of deep learning: concepts, CNN architectures, challenges, applications, future directions, J. Big Data 8 (1) (2021) 53.

[3] S. Dargan, M. Kumar, M.R. Ayyagari, G. Kumar, A survey of deep learning and its applications: A new paradigm to machine learning, Arch. Comput. Methods Eng. 27 (4) (2020) 1071–1092.

[4] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, in: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, NIPS '12, Curran Associates Inc., Red Hook, NY, USA, 2012, pp. 1097–1105.

[5] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, 2014, http://dx.doi.org/10.48550/ARXIV.1409.1556, URL: https://arxiv.org/abs/1409.1556.

[6] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.

[7] M. Zöller, M.F. Huber, Survey on automated machine learning, CoRR abs/1904.12054, 2019.

[8] B. Zoph, Q.V. Le, Neural architecture search with reinforcement learning, 2017, arXiv:1611.01578.

[9] F. Hutter, H.H. Hoos, K. Leyton-Brown, Sequential model-based optimization for general algorithm configuration, in: Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5, Springer, 2011, pp. 507–523.

[10] E. Lomurno, S. Samele, M. Matteucci, D. Ardagna, Pareto-optimal progressive neural architecture search, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, Association for Computing Machinery, New York, NY, USA, 2021, pp. 1726–1734, http://dx.doi.org/10.1145/3449726.3463146.

[11] A. Falanti, E. Lomurno, S. Samele, D. Ardagna, M. Matteucci, POPNASv2: An efficient multi-objective neural architecture search technique, in: 2022 International Joint Conference on Neural Networks, IJCNN, 2022, pp. 1–8, http://dx.doi.org/10.1109/IJCNN55064.2022.9892073.

[12] T. Elsken, J.H. Metzen, F. Hutter, Neural architecture search: A survey, J. Mach. Learn. Res. 20 (1) (2019) 1997–2017.

[13] E. Real, A. Aggarwal, Y. Huang, Q.V. Le, Regularized evolution for image classifier architecture search, in: Proceedings of the Aaai Conference on Artificial Intelligence, Vol. 33, 2019, pp. 4780–4789.

[14] B. Zoph, V. Vasudevan, J. Shlens, Q.V. Le, Learning transferable architectures for scalable image recognition, 2018, arXiv:1707.07012.

[15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1–9.

[16] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, K. Murphy, Progressive neural architecture search, in: Proceedings of the European Conference on Computer Vision, ECCV, 2018, pp. 19–34.

[17] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, W. Banzhaf, Nsga-net: neural architecture search using multi-objective genetic algorithm, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2019, pp. 419–427.

[18] H. Pham, M. Guan, B. Zoph, Q. Le, J. Dean, Efficient neural architecture search via parameters sharing, in: International Conference on Machine Learning, PMLR, 2018, pp. 4095–4104.

[19] T. Elsken, J.H. Metzen, F. Hutter, Efficient multi-objective neural architecture search via lamarckian evolution, 2018, arXiv preprint arXiv:1804.09081.

[20] C.-H. Hsu, S.-H. Chang, J.-H. Liang, H.-P. Chou, C.-H. Liu, S.-C. Chang, J.-Y. Pan, Y.-T. Chen, W. Wei, D.-C. Juan, Monas: Multi-objective neural architecture search using reinforcement learning, 2018, arXiv preprint arXiv:1806.10332.

[21] H. Liu, K. Simonyan, Y. Yang, DARTS: Differentiable architecture search, 2019, arXiv:1806.09055.

[22] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, J. Sun, Single path one-shot neural architecture search with uniform sampling, in: Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVI 16, Springer, 2020, pp. 544–560.

[23] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, Q. Le, Understanding and simplifying one-shot architecture search, in: J. Dy, A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning, in: Proceedings of Machine Learning Research, vol. 80, PMLR, 2018, pp. 550–559, URL: https://proceedings.mlr.press/v80/bender18a.html.

[24] H. Cai, C. Gan, T. Wang, Z. Zhang, S. Han, Once for all: Train one network and specialize it for efficient deployment, in: International Conference on Learning Representations, 2020, URL: https://arxiv.org/pdf/1908.09791.pdf.

[25] M. Middlehurst, J. Large, M. Flynn, J. Lines, A. Bostrom, A. Bagnall, HIVE-COTE 2.0: a new meta ensemble for time series classification, Mach. Learn. 110 (11) (2021) 3211–3243.

[26] H.A. Dau, E. Keogh, K. Kamgar, C.-C.M. Yeh, Y. Zhu, S. Gharghabi, C.A. Ratanamahatana, Yanping, B. Hu, N. Begum, A. Bagnall, A. Mueen, G. Batista, Hexagon-ML, The UCR time series classification archive, 2018, https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.

[27] A. Bagnall, H.A. Dau, J. Lines, M. Flynn, J. Large, A. Bostrom, P. Southam, E. Keogh, The UEA multivariate time series classification archive, 2018, http://dx.doi.org/10.48550/ARXIV.1811.00075, URL: https://arxiv.org/abs/1811.00075.

[28] H. Ismail Fawaz, B. Lucas, G. Forestier, C. Pelletier, D.F. Schmidt, J. Weber, G.I. Webb, L. Idoumghar, P.-A. Muller, F. Petitjean, InceptionTime: Finding AlexNet for time series classification, Data Min. Knowl. Discov. 34 (6) (2020) 1936–1962.

[29] A. Dempster, F. Petitjean, G.I. Webb, ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels, Data Min. Knowl. Discov. 34 (5) (2020) 1454–1495.

[30] H. Rakhshani, H. Ismail Fawaz, L. Idoumghar, G. Forestier, J. Lepagnot, J. Weber, M. Brévilliers, P.-A. Muller, Neural architecture search for time series classification, in: 2020 International Joint Conference on Neural Networks, IJCNN, 2020, pp. 1–8, http://dx.doi.org/10.1109/IJCNN48605.2020.9206721.

[31] Z. Wang, W. Yan, T. Oates, Time series classification from scratch with deep neural networks: A strong baseline, in: 2017 International Joint Conference on Neural Networks, IJCNN, IEEE, 2017, pp. 1578–1585.

[32] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.

[33] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN encoder-decoder for statistical machine translation, 2014, arXiv preprint arXiv:1406.1078.

[34] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: International Conference on Machine Learning, pmlr, 2015, pp. 448–456.

[35] P. Ramachandran, B. Zoph, Q.V. Le, Searching for activation functions, 2017, arXiv:1710.05941.

[36] I. Loshchilov, F. Hutter, Decoupled weight decay regularization, 2019, arXiv:1711.05101.

[37] I. Loshchilov, F. Hutter, Sgdr: Stochastic gradient descent with warm restarts, 2016, arXiv preprint arXiv:1608.03983.

[38] J. Cheng, L. Dong, M. Lapata, Long short-term memory-networks for machine reading, 2016, http://dx.doi.org/10.48550/ARXIV.1601.06733, URL: https://arxiv.org/abs/1601.06733.

[39] L. Prokhorenkova, G. Gusev, A. Vorobev, A.V. Dorogush, A. Gulin, CatBoost: unbiased boosting with categorical features, Adv. Neural Inf. Process. Syst. 31 (2018).

[40] S.M. Lundberg, S.-I. Lee, A unified approach to interpreting model predictions, in: I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems 30, Curran Associates, Inc., 2017, pp. 4765–4774, URL: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

[41] G. Larsson, M. Maire, G. Shakhnarovich, FractalNet: Ultra-deep neural networks without residuals, 2016, http://dx.doi.org/10.48550/ARXIV.1605.07648, URL: https://arxiv.org/abs/1605.07648.

[42] T. DeVries, G.W. Taylor, Improved regularization of convolutional neural networks with cutout, 2017, http://dx.doi.org/10.48550/ARXIV.1708.04552, URL: https://arxiv.org/abs/1708.04552.

[43] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, 2015, URL: https://www.tensorflow.org/. Software available from tensorflow.org.

[44] A. Krizhevsky, Learning Multiple Layers of Features from Tiny Images, Technical Report, University of Toronto, 2009.

[45] H. Xiao, K. Rasul, R. Vollgraf, Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms, 2017, arXiv:1708.07747.

[46] P. Helber, B. Bischke, A. Dengel, D. Borth, Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification, IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens. 12 (7) (2019) 2217–2226.

[47] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2017, arXiv:1412.6980.

[48] R. Müller, S. Kornblith, G.E. Hinton, When does label smoothing help? Adv. Neural Inf. Process. Syst. 32 (2019).
[49] K. Choi, G. Fazekas, M. Sandler, K. Cho, Convolutional recurrent neural networks for music classification, in: 2017 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP, 2017, pp. 2392–2396, http://dx.doi.org/10.1109/ICASSP.2017.7952585.
[50] M. Zihlmann, D. Perekrestenko, M. Tschannen, Convolutional recurrent neural networks for electrocardiogram classification, in: 2017 Computing in Cardiology, CinC, 2017, pp. 1–4, http://dx.doi.org/10.22489/CinC.2017.070-060.
[51] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (1) (2006) 1–30.
[52] S. García, F. Herrera, An extension on ‘‘statistical comparisons of classifiers over multiple data sets’’ for all pairwise comparisons, J. Mach. Learn. Res. 9 (89) (2008) 2677–2694.
[53] A. Benavoli, G. Corani, F. Mangili, Should we really use post-hoc tests based on mean-ranks? J. Mach. Learn. Res. 17 (5) (2016) 1–10.
[54] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, P.-A. Muller, Deep learning for time series classification: a review, Data Min. Knowl. Discov. 33 (4) (2019) 917–963.