PARETO-OPTIMAL PROGRESSIVE NEURAL ARCHITECTURE SEARCH

Danilo Ardagna, Eugenio Lomurno, Matteo Matteucci, Stefano Samele*
Department of Electronics, Information and Bioengineering
Polytechnic University of Milan
Italy

ABSTRACT

Neural Architecture Search (NAS) is the process of automating architecture engineering, searching for the best deep learning configuration. One of the main NAS approaches proposed in the literature, Progressive Neural Architecture Search (PNAS), seeks for the architectures with a sequential model-based optimization strategy: it defines a common recursive structure to generate the networks, whose number of building blocks rises through iterations. However, NAS algorithms are generally designed for an ideal setting without considering the needs and the technical constraints imposed by practical applications. In this paper, we propose a new architecture search named Pareto-Optimal Progressive Neural Architecture Search (POPNAS) that combines the benefits of PNAS to a time-accuracy Pareto optimization problem. POPNAS adds a new time predictor to the existing approach to carry out a joint prediction of time and accuracy for each candidate neural network, searching through the Pareto front. This allows us to reach a trade-off between accuracy and training time, identifying neural network architectures with competitive accuracy in the face of a drastically reduced training time.

Keywords POPNAS · PNAS · NAS · deep learning · machine learning · Pareto optimality

1 Introduction

In the last years, the contribution of machine learning has risen in many fields, increasing the request for intelligent and dynamic solutions. In particular, through the study and development of convolutional neural networks (CNNs), deep learning applications have achieved significant results in image classification and other computer vision tasks Krizhevsky et al. [2012], Simonyan and Zisserman [2014], Xie et al. [2017], He et al. [2016], Szegedy et al. [2015]. One of the most relevant limitations is the process of designing and building deep neural networks; in fact, the handcrafted design remains the primary constraint in terms of time taken and resources spent, and there is no guideline which grants a good intuition into the best network design.

Automatic machine learning (autoML) leads to a considerable acceleration in this sense: it makes it feasible to approach big data problems related to new fields by using black-box models that users can exploit even without being specialized data scientists. The mechanical design of an artificial neural network is a well-known task already addressed in the literature Zoph and Le [2016], Cai et al. [2018], Zoph et al. [2018], Liu et al. [2018]. These works successfully presented several strategies to build networks that achieved and overcame state-of-the-art accuracy on image classification tasks. While early works leveraged upon massive computational resources Zoph and Le [2016], recent works try to relax these requirements and proposed methods working even on a single GPU Liu et al. [2018].

Despite the progress achieved, the computation times of these techniques remain, in most cases, too expensive. In many scenarios, indeed, it is necessary to frequently update deep learning architecture, and the required time can become a fundamental discriminating factor.

In this work, we propose Pareto-Optimal Progressive Neural Architecture Search (POPNAS), a Neural Architecture Search (NAS) method that, starting from the Progressive Neural Architecture Search (PNAS) technique Liu et al.

*Authors contributed equally to this work.

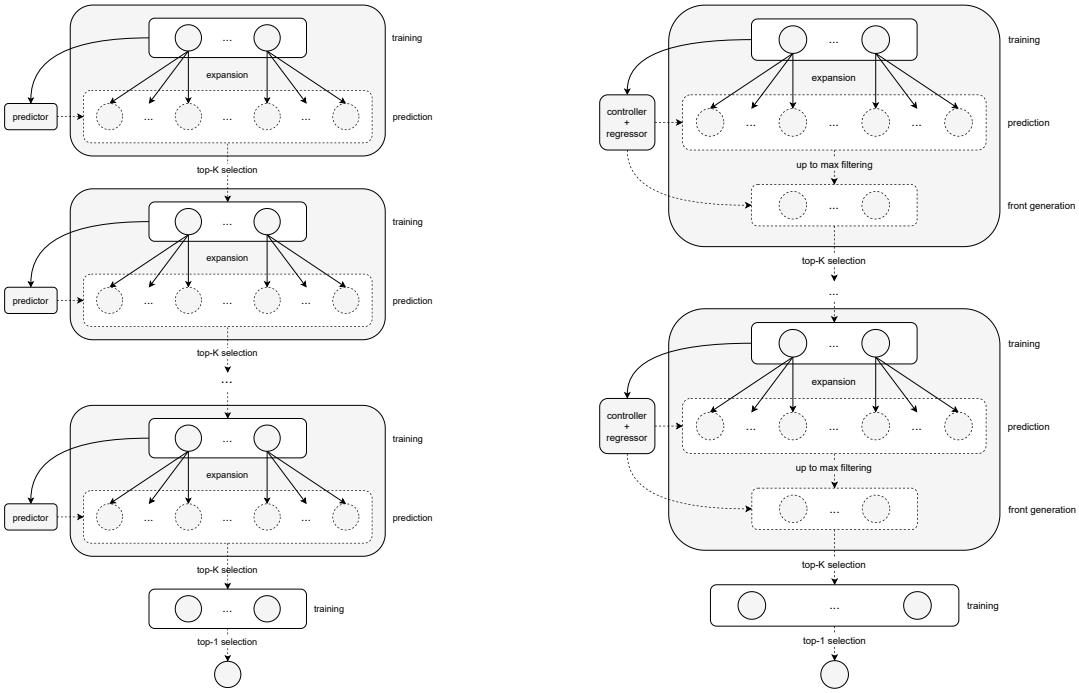

Figure 1: Schemas of PNAS (left) and POPNAS (right) architectures. With respect to the first one, the latter estimates the training time of the predicted architectures and then cuts out the most time expensive ones just before the top-K selection.

[2018], manages the trade-off between time and accuracy via Pareto efficiency. Thanks to the proposed approach, it is possible to obtain competitive performance results and massive reductions in model search time with respect to PNAS. To the best of our knowledge, this is the first work proposing a NAS technique that considers time as a constraint for the optimization problem.

The paper is organized as follows: Section 2 shows an overview of the existing NAS techniques and discusses how these works relate to PNAS. Section 3 presents POPNAS and the new performance and search strategy to include time constraints. Section 4 describes the conducted experiments and their results. It also includes the ablation study and the comparison with the PNAS technique. Finally, Section 5 discusses the novelty of the approach and the possible improvements of this research.

2 Related Works

Most NAS techniques are based on three fundamental steps:

- The definition of a search space, intended as the set of all admissible neural networks we want to build;
- The search strategy: an algorithm to explore the search space;
- The evaluation strategy: a way to evaluate and rank the explored models, such that the approach can address the development of the most promising ones.

While both the search space and the evaluation strategy have essential importance in the performance and computational costs of auto-generated models, the literature is often divided according to the most appropriate exploration strategy to be adopted, i.e., reinforcement learning, gradient-based optimization, evolutionary algorithm, and bayesian optimization.

This work is an extension of the paper *Progressive Neural Architecture Search* (PNAS) Liu et al. [2018], which extended and improved the ideas contained in *Neural Architecture Search* (NAS) Zoph and Le [2016]. NAS was the first attempt to successfully exploit a reinforcement-learning-based algorithm to build up deep learning architectures to overcome human design models in the image classification task through agent training, which is itself a neural network. The agent, also named controller, was a two-layer LSTM Hochreiter and Schmidhuber [1997] that generated the network specification, up to a pre-defined depth, in terms of a sequence of discrete value vectors. The learning process involved Proximal Policy Optimization, Schulman et al. [2017] or the Reinforce algorithm Williams [1992], where the reward

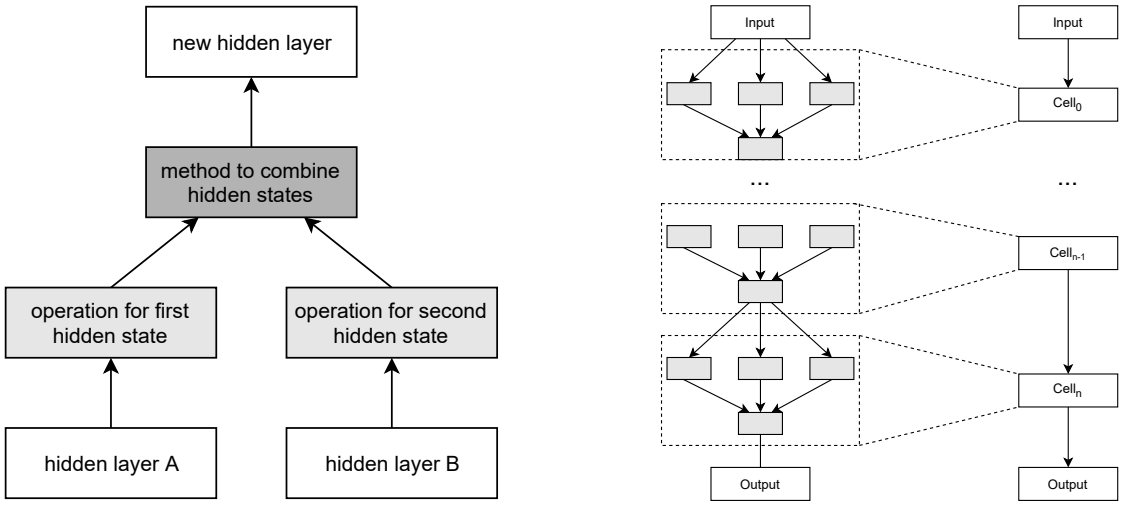

Figure 2: Summary diagrams of the transition from blocks to network topologies. Each block is composed of five parameters (left). After the inner structure is defined, each block is linked to other blocks to define a structure containing B blocks that we refer to as a cell (right). In this chain-structured architecture, each cell receives as input only the previous cell output.

signal was the accuracy of the controller sampled networks over a validation set. For this work, the authors conducted their experiments over the CIFAR-10 dataset Krizhevsky et al.. They also modified the procedure to learn recurrent network structures using the Penn Treebank dataset Marcus et al. [1994], a well-known benchmark for language modelling.

The main drawback of NAS was its computational cost since it required training and evaluating all the sampled set of child networks each time it updated the controller weights. Subsequent works tried to optimize learning by reducing the search space of network configurations or looking for less intensive candidate networks evaluation procedures.

In *Efficient architecture search by network transformation* (EAS) Cai et al. [2018], authors proposed to avoid training from scratch the children networks each time, but to enlarge and to modify an already explored solution adding function-preserving transformations, as in Net2net Chen et al. [2015a]. In *Learning transferable architectures for scalable image recognition* (NASNet) Zoph et al. [2018], instead of training the controller to generate a whole network, the authors reduced the research space to generate basic cells and then stack any of them to compose a network, similar to human-designed ResNet He et al. [2016], Xie et al. [2017] and InceptionNet Szegedy et al. [2016a], Szegedy et al. [2015], Szegedy et al. [2016b]. Their auto-generated network trained on CIFAR-10 was also able to achieve state-of-the-art accuracy over ImageNet Deng et al. [2009] without much inner modification. BlockQNN model Zhong et al. [2018] operated similarly, only using a Meta-Q-learning algorithm instead of the classic reinforce, as done in MetaQNN Baker et al. [2016].

The authors of PNAS proposed a cell-based approach too. Following a sequential model-based optimization approach (SMBO) Hutter et al. [2011], at each step of the learning procedure, their method tried to expand the structure of an existing cell made of a certain amount of blocks by adding a new operation. Then, the algorithm selected the top k cells according to a trained predictor able to anticipate the designed cells scores without actually evaluating them. PNAS predictor is aimed to foretell the accuracy of children networks. The authors tested different predictors and created a network that reached state-of-the-art performance on CIFAR-10, stacking the best-performing found cells. Training the predictor still required to train a subset of neural networks to use their encoding and accuracies as PNAS training set, this time with a gradient-based approach.

The mentioned method is close to other approaches, such as *Stochastic Neural Architecture Search* (SNAS) Xie et al. [2018], that consisted in an all-in-one gradient-based optimization method to update the controller parameters and the child networks, building an end-to-end trainable architecture. To obtain the same end-to-end optimization process, authors of *Differential Architecture Search* (DARTS) Awad et al. [2020] introduced a continuous relaxation of the architecture definition to allow the controller direct optimization over the validation set performance using gradient descent. The work has been further extended in DARTS+ Liang et al. [2019] with an early stopping procedure that prevents generated networks achieving poor performance.

There have been attempts also in using evolutionary algorithms to explore the search space, such as *Hierarchical Neural Architecture Search* (HNAS) Liu et al. [2017]. Authors restricted the search space by imposing a hierarchical network

structure; they also used an evolutionary algorithm based on tournament selection similar to Real et al. [2017]. They built complex architectures using different kinds of previously learned blocks, achieving good results even with random search exploration approaches. Other evolutionary methods specified a neural network structure and interconnection as a connectivity constraint matrix mapped into a bit-string genotype Miller et al. [1989], Suganuma et al. [2017].

Lastly, NAS-based bayesian optimization methods built a probabilistic model based on the objective function to find the most promising neural networks to train Shahriari et al. [2015], Snoek et al. [2012], Bergstra et al. [2012].

3 Proposed Method

The proposed method, named Pareto-Optimal Progressive Neural Architecture Search (POPNAS), is intended to keep all the PNAS algorithm advantages while dealing with time constraints to speed up the whole research and achieving similar accuracy performance. In order to do that, a new time regressor is required, which jointly works with the controller. As shown in Figure 1, at each iteration, after models expansion, one predictor, named *controller*, has to evaluate the accuracy of children architectures, as it is done in PNAS, while another predictor, named *regressor*, has to predict their training time to achieve the Pareto efficiency simultaneously.

3.1 Search Space

The POPNAS search space defines the set of architectures that the search strategy will take into account. As done in Liu et al. [2018], we define a block as a structure that maps two input tensors to one output tensor, considering it as the architecture basic unit. Then we construct a cell as a combination of up to B blocks. However, to build cell structures suitable for time prediction, some minor topology constraints are required.

3.1.1 Cell and Network Topology

A POPNAS cell is a fully convolutional network generated from a graph composed by 1 up to B blocks. A block is specified by the 5-tuple composed by two inputs, two operations, and a concatenation (I_1, I_2, O_1, O_2, C) as shown in left column of Figure 2. The considered operator space O contains the same eight PNAS operations, but in POPNAS, the order in which they appear becomes relevant. Thus, to each operation is associated an index from 1 up to 8, according to the time required to perform it in increasing order:

1. 3x3 average pooling
2. 3x3 max pooling
3. identity
4. 3x3 dilated convolution
5. 3x3 depthwise separable convolution
6. 5x5 depthwise separable convolution
7. 7x7 depthwise separable convolution
8. 1x7 convolution followed by 7x1 convolution

Only the concatenation is considered as combination operator (C) to maintain the same size of the PNAS search space. As in the right column of Figure 2, we encapsulate blocks into cells, where we take into account only a parallel topology, to further simplify the search space. Thus, each input of a block is only coming from previous cell output, and their outputs is concatenated and feed to the successive cell.

Each cell is converted into a CNN stacking it $3N$ times and adding 2 extra cells. N is the number of consecutive unrolled cells, and the extra two 2 cells are simple convolution operations with stride 2 inserted between the groups of N.

3.2 Search Strategy

The aim of POPNAS is to search for the most accurate cell structure among those with the lowest training time, pruning out the cells that take more time but have the same accuracy. We set a maximum time limit of L for the training time of children networks so that the algorithm can automatically exclude all the cells that take too long.

In the beginning, all the cells with only one block are generated, commuting all the possible combinations of two inputs. Then, they are added to a queue. We call the set of those cells b_1. Each cell is now iterated $3N + 2$ times to generate the child network with F initial filters, in case of a convolution operation. It is then trained for E epochs on a prefixed

Table 1: An example of the sliding blocks mechanism managing two cells with $b = 1$ and $B = 3$. After the operators re-index, a first row with the cell in correspondence of the first block is added to the observations, then the cell is shifted to the second block to create the second row and to the third block to create the third row. At this point, the iteration restarts with the next cell.

	1st block		2nd block		3rd block	
training time (sec)	op 1	op 2	op 3	op 4	op 5	op 6
63.145701	5	5	0	0	0	0
63.145701	0	0	5	5	0	0
63.145701	0	0	0	0	5	5
63.789210	6	6	0	0	0	0
63.789210	0	0	6	6	0	0
63.789210	0	0	0	0	6	6

Table 2: An example of the initial performance improvement at the beginning of the first iteration. A special network formed only by the global average pooling and the softmax classification layer is generated and trained. Then, the row is added to the observations as a training time belonging to an empty cell with $b = 0$.

		1st block		2nd block		3rd block	
training time (sec)	blocks	op 1	op 2	op 3	op 4	op 5	op 6
3.043455	0	0	0	0	0	0	0
63.145701	1	5	5	0	0	0	0
63.145701	1	0	0	5	5	0	0
63.145701	1	0	0	0	0	5	5

dataset, split into a train and a validation sets. We refer to the set of all one block cells as C_1 and to the set of just trained networks as M_1. It is important to notice that, once the networks are evaluated on the validation set, we have gathered information about the training times taken by each network, collected in the set T_1, and their accuracies, in the set A_1. The controller, designed to manage the generated architectures qualities, is trained based on the networks measured performance. We refer to the vector of the controller weights after this training step as π.

With reference to Figure 1, for each block dimension b from 2 to B, the set of previously selected cells are expanded adding all possible new blocks definitions, generating a search space subset S_b. For example, for $b = 2$, the one block cells are expanded adding all the possible permutations of the second block. For each expanded cell, the controller determines its accuracy. We call $\hat{A}_b$ the set containing the predicted accuracies of all the cells with b blocks. Besides, we feed the time regressor R_b with all the observations collected so far (C_{b-1}, $T_{b-1}, \ldots$); in the case $b = 2$, the architectures of all the one block cells with the relevant training times. Then, we pass to the time regressor each cell belonging to S_b to predict its training time. We refer to the set of predicted training times as $\hat{T}_b$. Then, firstly, the algorithm cuts back all the cells for which the time prediction is higher than L, if any, generating a subset S'_b of S_b. Secondly, a time-accuracy Pareto front P_b is generated from the predictions so that the most promising K fastest cells under the same accuracy are picked up and added to the queue (S''_b). New child networks are created stacking them. The networks are trained and evaluated on the same dataset as before.

Also in this case, starting from S''_b, M_b is the set of stacked cells with b blocks, C_b and T_b are the sets of the cells with b block cells and their training times, respectively, while A_b is the set of their accuracies. A_b is used to update the controller, obtaining a new vector π composed by its new weights. Then, the process restarts from the K cells expansion into $K' >> K$ new ones with size $b + 1$. At the end of the last step, the best cell in terms of accuracy, among all the trained ones, is returned by the algorithm.

3.3 Performance Estimation Strategy

The POPNAS estimation strategy consists of the combined work carried out by two different predictors: for each cell, an LSTM controller gives the estimated accuracy like in PNAS, while a new time regressor is added to evaluate the estimated training time. This new time regressor, selected among many regression models and representing the main innovation of our work, acts in two places with different purposes. The first time is during the Pareto front generation. Here, the regressor has only to grant a proper models ranking because the algorithm considers only the cells with lower training time under the same accuracy. The second time is during the cells pruning phase, when the algorithm discards all the proposed architectures that exceed the training time limit L.

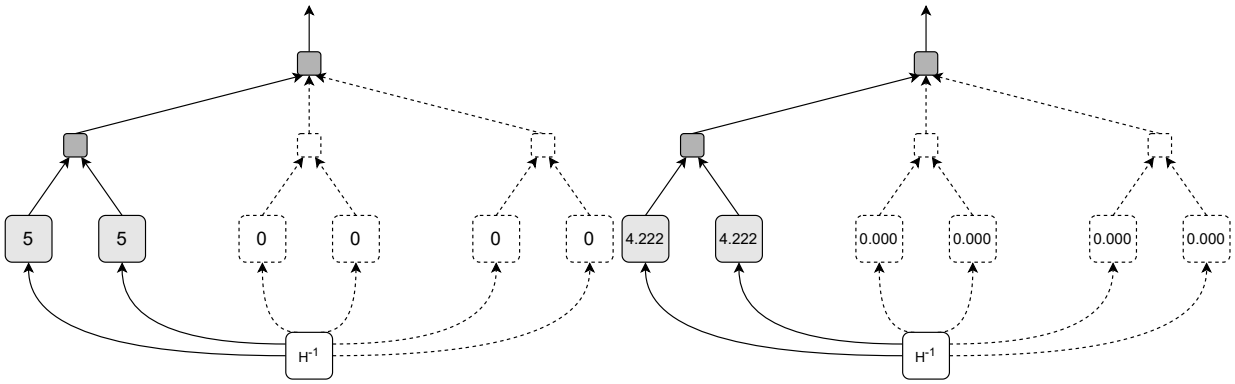

Figure 3: On the left, a representation of how the regressor handles the static re-index procedure of a cell with one block, fixing the overall dimension to B = 3. The two blocks operators are fixed with value equal to 5, corresponding to 3x3 depthwise separable convolution, while all the empty blocks operators are set to 0. On the right, a representation of how the regressor now handles the same one block flat cell of the static re-index example using the dynamic re-index procedure.

3.3.1 Sliding blocks mechanism

At each iteration of the algorithm, the regressor has to predict the training time taken by cells with b+1 blocks having available only observations of cells up to b blocks. This scenario implies that it is required to assign weights to features never seen before. We apply a sliding blocks mechanism to the cell architecture, sliding it over block to block, as represented in Table 1. Considering a cell with dimension b, there are $B - b$ empty blocks: the same training time is passed to the regressor as if the cell occupies the first b blocks, as if it occupies the blocks from the second to the $(b + 1)^{th}$ and so forth. In this way, each feature is evenly distributed over the observations.

3.3.2 Initial trust improvement

The time regressor considers as features the cell architecture and the number of blocks that constitutes it. At the first iteration, the regressor has to predict the training time of the block 2 having only the block 1 available, but in the case of multi-branch architecture topology we need to feed the regressor with the previous two blocks. As for the sliding blocks mechanism, this issue requires an adeguate technique which must be applied at each POPNAS execution.

To deal with it, we add to the observations a special row with b = 0 as an initial thrust, in which the architecture is empty and the network consists only of the final global average pooling followed by the softmax classification layer, whose presence is independent of the b value. After the second iteration, since the predictor will be able to see observations both with b = 1 and b = 2, the initial row will be removed.

3.3.3 Performance Prediction

Different prediction methods have been compared in order to find the best one for POPNAS. In particular, we consider two linear regression methods, a tree boosting system and a heuristic algorithm. At each POPNAS iteration, the chosen regressor has to handle cell architectures from 1 up to b blocks, so it is necessary to have a uniform data dimension, regardless of the number of blocks. Since a row considers the operations and the blocks of a cell, POPNAS extends all the cells encoding to B blocks, considering unused the empty ones and setting their features to default values to standardize the observation length.

The selected time prediction methods are Ridge regression, linear regression with non-negative least squares (NNLS), XGBoost and a heuristic method based on the sum of single block training times. At the $(b + 1)^{th}$ iteration, for each architecture, sum-block predicts the training time as the sum of the time taken by the cell without the $(b + 1)$ block, i.e. the one obtained in the previous iteration, and the training time of the $(b + 1)^{th}$ block, seen as a one-block-cell. The process by which we chose the best regressor for POPNAS is shown in the subsection dedicated to ablation studies in Section 4.

3.3.4 Operators Re-index

Since the operators have categorical values, it becomes essential to find an efficient way to encode cell architecture as a set of observation features to feed the time regressor.

Table 3: Average relative error of the four regressors, evaluated both for each block size and in a progressive way.

	Proper			Progressive		
	B = 2	B = 3	B = 4	B = 2	B = 3	B = 4
Block sum	0.1821	0.2758	0.2390	0.1821	0.2290	0.2323
Ridge	0.3465	0.0524	0.0353	0.3465	0.1995	0.1444
NNLS	0.3660	0.0524	0.0355	0.3660	0.2092	0.1510
XGBoost	0.5778	0.4849	0.3893	0.5778	0.5314	0.4838

Table 4: Average relative error of Ridge and NNLS regression before and after each transformation, evaluated both for each block size and in a progressive way.

	Proper (Ridge)			Progressive (Ridge)			Proper (NNLS)			Progressive (NNLS)		
	B = 2	B = 3	B = 4	B = 2	B = 3	B = 4	B = 2	B = 3	B = 4	B = 2	B = 3	B = 4
Standard	0.3270	0.0524	0.0353	0.3270	0.1897	0.1380	0.3483	0.0524	0.0355	0.3483	0.2003	0.1451
Reordered	0.3126	0.0613	0.0637	0.3126	0.1869	0.1456	0.2580	0.0469	0.0625	0.2580	0.1524	0.1223
w/o inputs	0.4018	0.0522	0.0388	0.4018	0.2270	0.1639	0.3483	0.0522	0.0388	0.3483	0.2002	0.1461
Reord w/o inp	0.3028	0.0470	0.0709	0.3028	0.1749	0.1401	0.2580	0.0469	0.0709	0.2580	0.1524	0.1251

To solve this problem, we propose two different solutions, i.e. static and dynamic re-index techniques. In the first case, each operator is associated to an integer value ranging from 1 to the size of the operator set. In the latter case, we consider a heuristic re-index method that takes into account the distance between indices. To do that, after training the same cells as in the static re-index case, we normalize each value dividing it by the highest observed training time and then we multiply it by the size of the operator set. The formula we apply can be written as follows:

$$index_i = \frac{time_i}{max(T_1)} \cdot size(T_1)$$

where T_1 is the training time set of the symmetric flat cells with $b = 1$ and i ranges from 1 to $size(T1)$. These methods allow to treat an empty block as a normal block with operators indices equal to 0, i.e. empty operation, as shown in Figure 3. The process by which we chose to apply or not a re-index technique for POPNAS is shown in the subsection dedicated to ablation studies in Section 4.

4 Experiments & Results

The first part of the experiments conducted for this work is an ablation study to determine which is the best regressor to estimate cells training time. Since the method we have proposed in the previous section heavily leverages a time limit L, pruning cells with a higher training time, we would like to have a time regressor able to estimate it with high accuracy. The following section directly compares PNAS and POPNAS methodologies in terms of accuracy and training time.

4.1 Ablation Study

POPNAS method selects the Pareto-optimal solutions with respect to time and accuracy, making the process of evaluating the time regressor performance dependent from POPNAS itself. Thus, we conducted the experiments using PNAS method, where, besides training the accuracy controller we also trained and evaluated the time regressor. We have investigated different models through the use of the a-MLLibrary Lattuada [2019]. It is a library for the generation of regression models that allows building the best predictor among a wide range of regression types given a set of observations, using a black-box approach. We conducted our experiments on a subset of CIFAR-10. In particular, since CIFAR-10 has five batches of 10,000 images each, we used a randomly selected single batch, splitting it into a training set of 9,000 images and a validation set of 1,000 images. For the accuracy predictor, i.e., the controller, we used an LSTM as in the PNAS algorithm, with a learning rate fixed to 0.002. For the child networks, we adopted the same learning rate as in the original paper, i.e., 0.01. We evaluated $K = 256$ networks out of the generated ones at each stage during the search. We used a maximum cell depth B equal to 4 to speed up the process. We always used 32 as the initial number of filters, in case of convolutions, iterating the cells for N = 2 times, and training each child network with E = 20, as it is in PNAS.

Experiments were performed on an NVIDIA Tesla V100 SXM2, with 16GB VRAM. As time regressor models, we chose Ridge Regression, XGBoost Chen et al. [2015b], and NNLS (Non-negative least square). Ridge regressor was trained with α set to 0.1. For NNLS, we experimented both with and without the fit_intercept set to True. For XGBoost, we considered 1 and 3 as child-weight thresholds to stop the tree splitting if exceeded; gamma, that regularizes the information across the trees, allowing the node addition only if the associated gain is larger or equal to the given value,

Table 5: Average relative error, evaluated with the norm of the absolute error, maximum relative error and minimum relative error of NNLS, with static re-index and dynamic re-index.

	Avg rel error with abs	Max rel error with abs	Min rel error with abs
Static	0.237973999	0.60435179	0.017062738
Dynamic	0.212871771	0.348979024	0.081329244

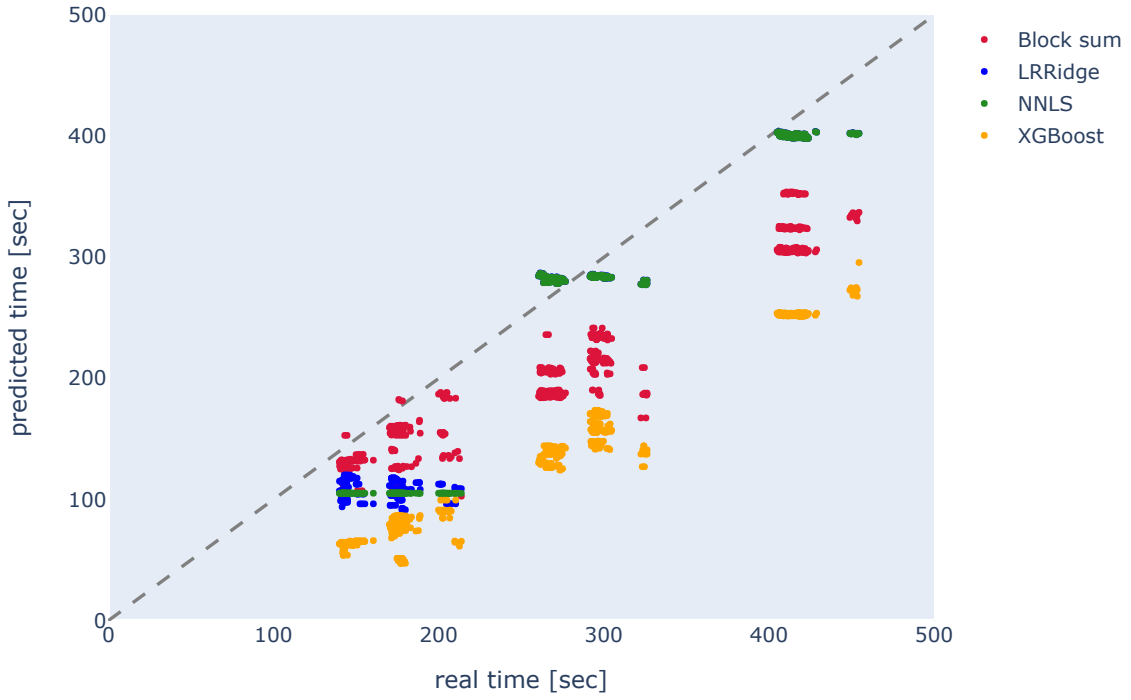

Figure 4: Performance of time-regressors. The identified clusters are related to the blocks iterations.

equal to 0 and 1; numbers of tree regressors: 50, 100, 150 and 250. We selected as possible learning rate 0.01, 0.05, and 0.1. The trees' maximum depth has been chosen equal to 1, 2, 3, 5, 9, and 13. The a-MLLib performed a 5-fold cross-validation to find, through a grid search strategy, the best hyperparameter settings for each regressor. In our experiments, we trained time-regressors on data available at iteration b, and evaluated them at iteration b+1, before the updating procedure. In the ablation study we first compared the selected methods and then we have progressively pruned the less performing approaches, assessing the impact of the proposed optimization strategies: input removal, static re-index, and dynamic re-index.

4.1.1 Time Prediction Performance

In this section, we present the result of the first plain comparison. It can be deduced from Figure 4 that block sum shows better results only at the first iteration, while it tends to gradually underestimate the training time from the second iteration onwards. This consideration allows us to deduce that the training time increase induced by a block addition is not entirely linear, but it introduces a bias dependent on the number of blocks. Table 3 shows that Ridge regression and NNLS are the two most accurate methods; regressors errors are calculated and shown both in a progressive way, i.e. using data from the first to the current block, and in a proper way, i.e., keeping only data from the current block.

4.1.2 Static Re-index and Input Pruning

This section presents the impact of the static re-index methodology as explained in Section 3.3.4, and input information pruning strategy on the NNLS and Ridge regression, i.e., the best models of the previous section. Input information pruning consists on removing the information I_1, I_2, from the feature dimensions regressor input. Table 4 shows the

Table 6: Average relative error of NNLS with static re-index and dynamic re-index, evaluated both for each block size and in a progressive way.

	Proper			Progressive		
	B = 2	B = 3	B = 4	B = 2	B = 3	B = 4
Static	0.2018	0.2450	0.2642	0.2018	0.2248	0.2380
Dynamic	0.1869	0.1958	0.2547	0.1869	0.1916	0.2128

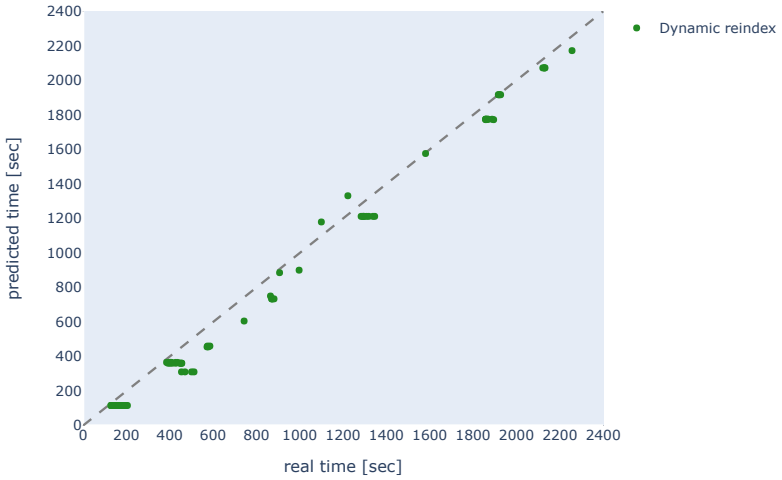

Figure 5: The NNLS predictions on POPNAS, with dynamic re-index.

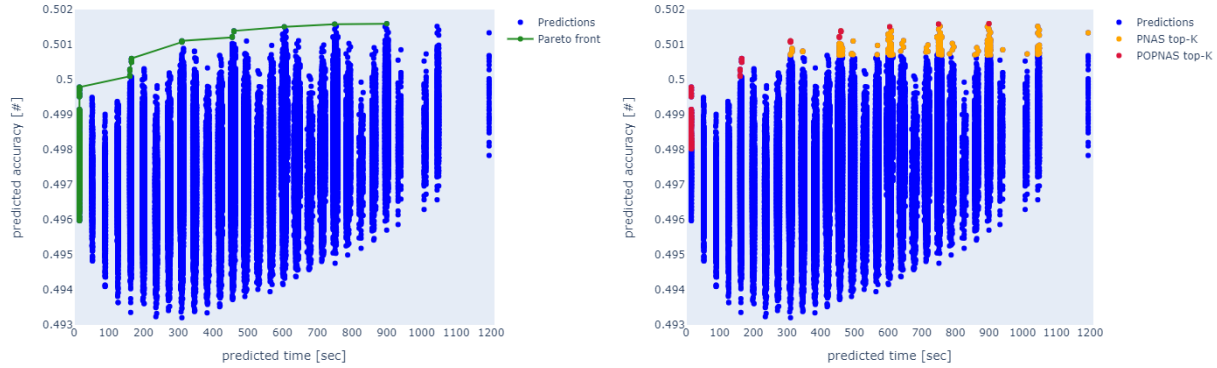

Figure 6: On the left, the Pareto front generation (green), after the prediction of all the architectures with B = 2 (blue). On the right, The top-K selection comparision between PNAS (yellow) and POPNAS (red) with K = 256, above all the architectures with B = 2 (blue).

performance for NNLS and for Ridge regressor. Static re-index helps the predictor to reduce training time error for NNLS, as we can see comparing the first two lines, specifically at Step 2 and 3. Even if error increases slightly at Step 4, the progressive evaluation shows at the last iteration a general improvement. NNLS is also scoring the lowest progressive error. Ridge regression instead does not show improvement concerning the baseline model. Results also show that inputs information is not affecting the performance and can thus be removed.

4.1.3 Dynamic Re-index

This section presents the comparison between static and dynamic re-indexes on NNLS algorithm; we have chosen to explore only the most promising algorithm from previous section experiments. We trained the POPNAS algorithm according to the previous setup choices. The benefits of dynamic re-index can be immediately noticed as shown in Table 5. In fact, with this technique we notice an overall lower average relative error of 0.2129, compared to static

Table 7: The POPNAS networks accuracy from $B = 1$ to $B = 5$. The table considers the accuracy of the best cell, the average of the best 5 cells, the average of the best 25 cells and average of the all the trained cells with the same block size, for each step of the algorithm.

Top	B = 1	B = 2	B = 3	B = 4	B = 5
1	0.694	0.673	0.685	0.701	0.741
5	0.686	0.659	0.672	0.680	0.723
25	0.662	0.457	0.492	0.635	0.675
256	0.541	0.305	0.380	0.488	0.637

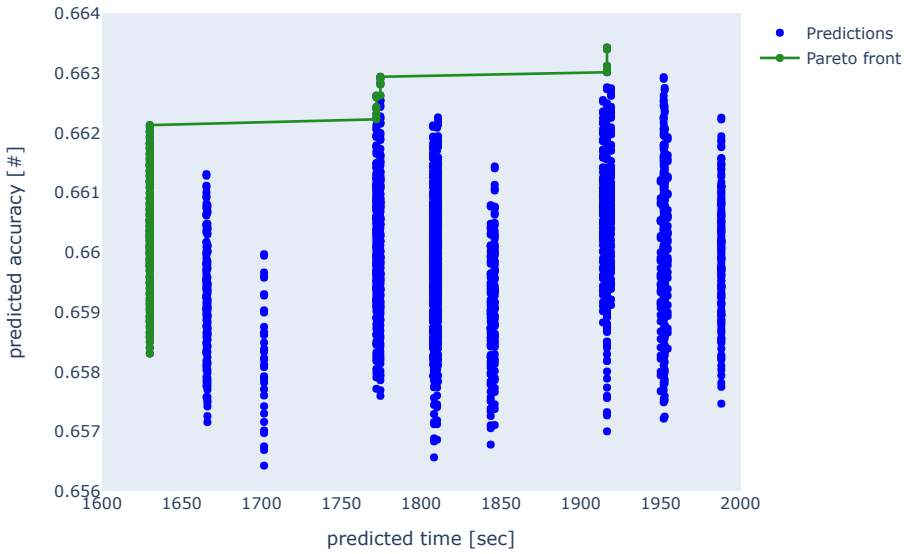

Figure 7: The Pareto front generation (green), after the prediction of all the architectures with B = 5 (blue).

re-index, which instead achieves an error of 0.2380. Moreover, NNLS with static re-index has a limited tendency to underestimate the training time. In Table 5 we can witness the better performance of dynamic re-index through iterations, actually reaching an improvement of 0.05 at the second one, compared to static re-index. Even if the dynamic re-index did not grant an almost perfect accuracy, it seems to halve the maximum relative error, reducing it from 0.6044 to 0.3490, at the expense of a lower minimum relative error increasing, from 0.0171 to 0.0813 (6).

4.2 POPNAS vs PNAS

In the final PNAS-POPNAS comparison, we use NNLS with dynamic re-index and input pruning as our time regressor. We consider cells with lookback depth equal to 1, the complete operator set B = 5, and learning rate equal to 0.01 as in the original paper Liu et al. [2018]. In this case, the Pareto front considered at most 256 best architectures in terms of predicted accuracy and training time. The experiment is performed on an NVIDIA GeForce GTX 1080Ti, with 11 GB VRAM. The PNAS model has been taken from the official tensorflow repository. The architecture has not been modified, but we used the same training settings for a fair comparison.

From Figure 5, we can make a few considerations about the time regressor performance. First of all, the dynamic re-index method confirms that the training time is now less underestimated when compared with the previous approaches. We have concluded that the main reason for its prediction improvement is the uneven presence of the operators in the Pareto front; indeed, we have observed that the algorithm focuses mainly on a narrow subset of the allowable operators, which are considered the best trade-off between a short training time and a good accuracy: 3x3 depthwise separable convolution, 3x3 average pooling and 3x3 max pooling. By this way, time predictions benefit from a smaller operators pool.

We can also observe that a small group of architecture prediction times is overestimated: it is the subset of cells containing at least a 1x7 followed by 7x1 convolution. Since only a few observations contain that operator, the regressor tends to predict a higher training time than the real one in the algorithm intermediate steps. Another essential detail is the absence of widely separated groups of observations, with the variations of B. This phenomenon happens due to the

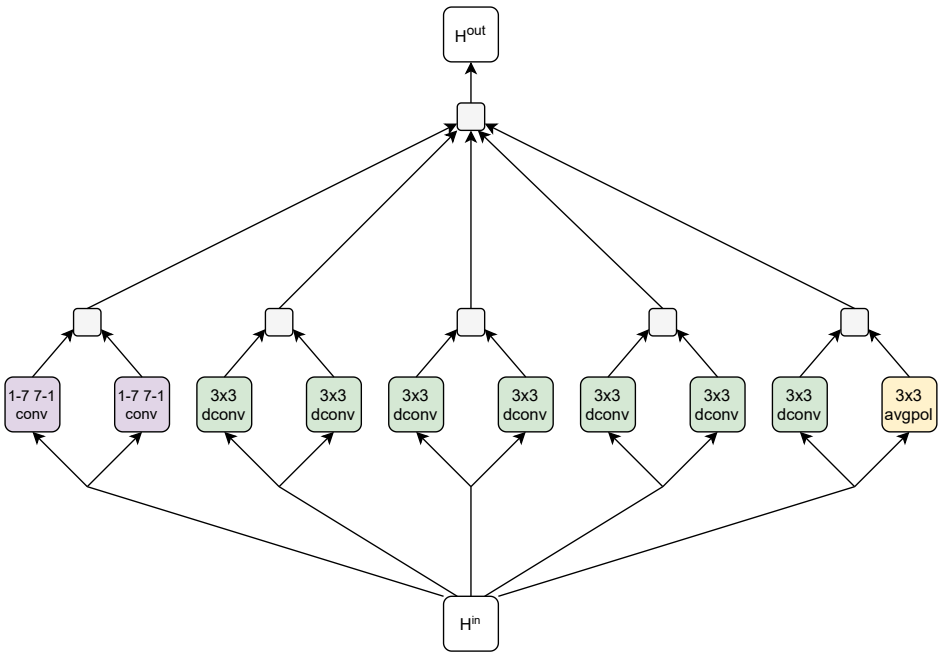

Figure 8: The POPNASNet-5 best cell architecture.

Table 8: Performance comparision between PNASNet-5 and POPNASNet-5.

	Accuracy	Training time
PNASNet-5	0.8223	1 hour 6 min
POPNASNet-5	0.7433	38 min

increased variance of the training time through the Pareto front: a neural network entirely made up of pooling layers has a training time less than half of a network with convolutional layers only.

On the left of Figure 6, we show the Pareto front generation for cells with B = 2, at the end of the first iteration of the algorithm. The right part of Figure 6 illustrates the best 256 architectures in terms of predicted accuracy as selected from the Pareto front. The graph highlights the difference between the PNAS and POPNAS top-K selections: in fact, PNAS takes a total training time for B = 2 of 2 days, 1 hour and 31 minutes, with an average training time of 11 minutes per network, while POPNAS takes a total training time of 4 hours and 29 minutes, with an average training time of only 50 seconds per network. On the contrary, the top-K selected cells average predicted mean accuracy with B = 2 is only minimally reduced from 0.5009 to 0.4989 in POPNAS.

Figure 7 shows the Pareto front generation at the last iteration of the algorithm. This time, the predicted architectures are spread over a smaller time range with higher accuracy. This confirms that the algorithm has developed a relatively narrow subset of architectures through the iterations, composed of the same operators with different permutations: since all the architectures consist mainly of 3x3 depthwise separable convolution, 3x3 average pooling and 3x3 max pooling, the number of columns is lower than in the first iteration. If we finally compare the time required by the architectures trained at the last iteration of our first test with the ones of the current test, the advantages gained with POPNAS are very clear: the total training time falls from 11 days, 15 hours and 38 minutes, with an average training time of 1 hour and 5 minutes per network, to 5 days, 23 hours and 36 minutes, with an average training time of 33 minutes per network only. So, reducing by 13.3% the average accuracy of the 256 trained cells with B = 5, which decreases from 0.770 to 0.637, we obtain a 2x boosting in terms of required training time.

We summarize the accuracy performance obtained on the validation set at each iteration of the algorithm in Table 7. We can notice that the average performance accuracy of the 256 selected cells drastically decreases at the second iteration of the algorithm: the main reason can be seen in Figure 6, where we can observe that most of the architectures picked up by both the algorithms have a much lower predicted accuracy than the remaining part. The benefits of the POPNAS algorithm are clearly visible from the fourth iteration, in which the best trained cells exceed both 0.7 in terms of accuracy. Each row of the last iteration also exceeds the relevant values of the first one: this means that the Pareto

front converges to a subset of architectures competitive with the one found by PNAS, but with a much lower execution time of the entire algorithm.

Due to the computational and time constraints highlighted with the first CIFAR-10 batch, we have conducted the final experiments over the second batch of CIFAR-10 instead of on the entire training set or over different datasets. We are interested in evaluating the improvements of our approach with respect to PNAS over unseen data, which means also smaller amounts of samples are reasonable, given the evidence of such improvements. The results comparison includes the best cell found by POPNAS, (POPNASNet-5, Figure 8), and the one found by PNAS (PNASNet-5), run over our defined above search space. The best POPNAS cell is the architecture of the Pareto front with the best accuracy over the validation set.

The results can be seen in Table 8: with an accuracy reduction from 0.8223 to 0.7433, POPNASNet-5 halves the time required for its training, decreasing from 1 hour and 6 minutes to 38 minutes only. The two values also seem to be consistent with the average training time observed at the relevant algorithms in the last step. According to our expectations, as shown in Figure 6, the accuracy reduction is due to the pruning of some promising block in favour of faster ones, especially during the early stages of the algorithms. From a different point of view, the search for the best solutions over the Pareto front is strictly connected to the threshold chosen to prune expensive time blocks and heavily affect the evolution of the search space. Even if we observed that this effect decreases over the iterations, it is the main reason for the progressive gap with PNAS best cell in terms of both the measured performance. While the proposed approach adopted a greedy pruning step to underline the effort of the proposed technique, it is still possible to leverage a relaxation of time constraints. Nevertheless, we have proven that under strict requests, it is possible to adopt NAS techniques with virtuous trade-offs that reduce computation time and maintain competitive accuracy performance.

5 Conclusion

In this work, we presented POPNAS, a progressive neural architecture search algorithm that considers the trade-off between accuracy and time with the Pareto efficiency property. We have achieved significant computational time improvements by training time regressors over the PNAS starting algorithm while maintaining competitive performance in the found architecture. This paper objective is dual: on the one hand, to propose an efficient solution to the NAS tasks that often require enormous computational resources and turn out to be not feasible without the availability of powerful hardware tools. On the other hand, providing the intuition of carrying out autoML tasks by balancing different factors according to different needs.

With this perspective, there are numerous steps we intend to explore, such as adding new operations and new layer topologies to the search space or using other regressors to improve time estimates further. Another way forward could be studying other trade-offs with different metrics or tasks different from the classification.

Acknowledgements

The European Commission has partially funded this work under the H2020 grant N. 101016577 AI-SPRINT: AI in Secure Privacy-pReserving computINg conTinuum. The authors would also like to thank Matteo Vantadori and Marco Lattuada for their initial effort over the development and experiment activities.

References

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*, pages 19–34, 2018.

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

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). URL `http://www.cs.toronto.edu/~kriz/cifar.html`.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. The penn treebank: Annotating predicate argument structure. In *Proceedings of the Workshop on Human Language Technology*, HLT '94, page 114–119, USA, 1994. Association for Computational Linguistics. ISBN 1558603573. doi:10.3115/1075812.1075835. URL `https://doi.org/10.3115/1075812.1075835`.

Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015a.

Christian Szegedy, Christian Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261*, 2016a.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016b.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2423–2432, 2018.

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer, 2011.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926*, 2018.

Noor Awad, Neeratyoy Mallik, and Frank Hutter. Differential evolution for neural architecture search. *arXiv preprint arXiv:2012.06400*, 2020.

Hanwen Liang, Shifeng Zhang, Jiacheng Sun, Xingqiu He, Weiran Huang, Kechen Zhuang, and Zhenguo Li. Darts+: Improved differentiable architecture search with early stopping. *arXiv preprint arXiv:1909.06035*, 2019.

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

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, pages 2902–2911. PMLR, 2017.

Geoffrey F Miller, Peter M Todd, and Shailesh U Hegde. Designing neural networks using genetic algorithms. In *ICGA*, volume 89, pages 379–384, 1989.

Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the genetic and evolutionary computation conference*, pages 497–504, 2017.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *arXiv preprint arXiv:1206.2944*, 2012.

James Bergstra, Daniel Yamins, and David Cox. Hyperopt: Distributed asynchronous hyper-parameter optimization. *Retrieved May*, 21:2020, 2012.

Marco Lattuada. a-mllibrary. `https://github.com/eubr-atmosphere/a-MLLibrary`, 2019. [Online; accessed 28-March-2020].

Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, et al. Xgboost: extreme gradient boosting. *R package version 0.4-2*, 1(4), 2015b.