

Efficient Progressive Neural Architecture Search*

Juan-Manuel Pérez-Rúa

juanmanuel.perezrua@orange.com

Moez Baccouche

moez.baccouche@orange.com

Stéphane Pateux

stephane.pateux@orange.com

Orange Labs
4 rue Clos Courtel, 35512
Cesson-Sévigné, FR

Abstract

This paper addresses the difficult problem of finding an optimal neural architecture design for a given image classification task. We propose a method that aggregates two main results of the previous state-of-the-art in neural architecture search. These are, appealing to the strong sampling efficiency of a search scheme based on sequential model-based optimization (SMBO) [15], and increasing training efficiency by sharing weights among sampled architectures [18]. Sequential search has previously demonstrated its capabilities to find state-of-the-art neural architectures for image classification. However, its computational cost remains high, even unreachable under modest computational settings. Affording SMBO with weight-sharing alleviates this problem. On the other hand, progressive search with SMBO is inherently greedy, as it leverages a learned surrogate function to predict the validation error of neural architectures. This prediction is directly used to rank the sampled neural architectures. We propose to attenuate the greediness of the original SMBO method by relaxing the role of the surrogate function so it predicts architecture sampling probability instead. We demonstrate with experiments on the CIFAR-10 dataset that our method, denominated Efficient progressive neural architecture search (EP-NAS), leads to increased search efficiency, while retaining competitiveness of found architectures.

1. Introduction

Since the popularization of convolutional neural networks (CNN) for image classification by Krizhevsky *et al.* [12] in 2012, many subsequent works have proposed new hand-designed architectures leading to steady performance improvements over the years (see, for exam-

ple [6, 7, 22, 26, 9], among others). The process leading to all these discoveries is almost always arduous, requiring careful experimentation and the intuition of an expert. Some of these findings seem to point to the fact that intricate neural network designs can offer large gains in performance [7, 9, 26]. However, it is not always obvious what to change in a known design to push performance even further. Evidently, there is a need for efficient yet effective ways to discover new architectures of CNN automatically. This is the problem that we tackle in this paper. Here, we focus on finding optimal CNN architectures for image classification, while keeping in mind that, very often, advances in neural networks for image classification transfer to a large variety of other learning problems.

Current advances in neural architecture optimization can be categorized in roughly three groups: genetic algorithms (GA), reinforcement learning (RL), and surrogate-based optimization (SO). GA-based approaches [20, 25, 28] consist on iteratively mutating, training and evaluating promising architectures, while only top performers on a validation set are selected for further mutation. In RL methods [29, 30, 31], a controller agent generates the description of a neural model, which is then trained and evaluated on a validation set. Validation error is then fed-back to the agent as a reward function, improving the controller and generating better models in future iterations. On the other hand, by SO we refer to methods that rely on learning a sort of surrogate function expressing a relationship between sampled models and validation error or similar. Typical examples are Bayesian optimization [23, 24] and sequential model-based optimization (SMBO) [15].

Recently, progressive neural architecture search (PNAS) [15], one of the surrogate-based search methods, achieved state-of-the-art results on the CIFAR-10 dataset [11]. Their algorithm performs a progressive scan of the neural architecture search space (which is constrained by design according to findings of previous

*To be presented at the *British Machine Vision Conference (BMVC 2018)*

state-of-the-art). The top K best performing architectures are chosen at each step of the algorithm and validation errors are collected by training the selected architectures for several epochs. These errors are then used to train a surrogate function that predicts validation error of subsequent architectures. The surrogate function allows efficient exploration of the search space by decreasing the amount of architectures that are actually trained. Computational cost is nonetheless high, requiring 100 GPUs working for 2 days to achieve their best results. However, PNAS [15] is considerably more efficient than previous methods, which rely on up to 800 GPUs working for a month (*i.e.*, NAS [30]) to achieve their best results.

A recent work based on RL, ENAS [18], proposed to leverage weight-sharing among sampled architectures to increase search efficiency. The motivation behind this idea is the observation that the main bottleneck during neural architecture search is the training of sampled models to completion. Weight-sharing in this context removes the need to train sampled architectures from scratch. This idea improves time efficiency by a factor of 1000 with respect to NAS [30]. We propose in this paper **to investigate the effect on speed and performance of weight-sharing in progressive SMBO-based optimization** for neural architecture search. The purpose of this is to benefit from the simplicity and impressive performance of PNAS [15], while improving on speed thanks to the efficiency of ENAS [18]. Leveraging the gain on efficiency, we also propose **to relax the sampling strategy of PNAS by performing probabilistic sampling of new architectures** based on the prediction of the surrogate function. We shall call this new approach **efficient progressive neural architecture search** (EPNAS).

The remainder of this paper is organized as follows. In Section 2, we give an overview of the related work and state-of-the-art in neural network architecture search. In Section 3 we introduce the search space our method acts on, and in Section 4 we explain EPNAS, our algorithm for neural architecture optimization. Later on, in Section 5, we present experimental results and analysis of the properties of our method. Finally, in Section 6, we give concluding remarks and discuss future work.

2. Related work

Optimization of neural network hyperparameters is a problem that has been tackled since the early nineties [1, 3]. Researchers quickly realized that, due to the extremely high computational demands of the problem, neural topology optimization would need to wait for the advent of higher-order parallel computing hardware and software [3]. Traditionally, up to this point, most of the attempts to tackle the problem were based on **randomized search methods**. A few years later, in 2002, Stanley and Milkkulainen [25] pro-

posed an evolutionary random search approach that solves jointly for optimal weights and network topology. In their work, topology mutation follows structured constraints and it is performed progressively.

Fast forward to recent years, evolutionary and genetic algorithms are still relevant. However, most modern methods focus on evolving the neural topology [16, 19, 20, 28], leaving the optimization of neural weights to gradient-based approaches. These methods, however, are reserved for large hardware set-ups with hundreds of GPUs running for weeks.

Another group of practical approaches for automatically selecting a neural topology relies on **Bayesian optimization** [23, 24]. In this type of methods, the validation error of neural models is modeled as a Gaussian process, leading to optimal selection of candidate configurations. However, these methods cannot handle variable-size and variable-connectivity models.

Very recently, **reinforcement learning** appeared as an alternative to previous search methods. The technique usually consists of a controller recurrent neural network that samples new architectures at each iteration. This controller is trained with policy gradient [30] or Q-learning [29], by feeding it with the validation errors collected by training the sampled networks to completion. A large scale benchmarking work by Real *et al.*, [19], demonstrates that evolutionary approaches either match or surpass performance of RL methods while also reaching that outcome faster. In order to increase efficiency, subsequent works have focused on constraining the search space to modular elements composed of convolutional operations with demonstrated value for image recognition [31]. Similarly to the method from [20], which allows weight inheritance during exploration of the search space, ENAS [18] proposes an interesting idea: sharing weights among sampled architectures. The weight-sharing strategy in RL-based approaches offers competitive results while improving exploration speed by a large margin.

In parallel to RL-based approaches, a different type of methods appeared recently. **Surrogate aided exploration** has also demonstrated accuracy and efficiency for network hyperparameter optimization [15, 17]. Sequential model-based optimization (SMBO) lies at the core of such methods. SMBO originated as a technique for general algorithm parameter optimization. The method itself does not require gradients, but it requires the training of an intermediate function that scores parameter configurations. During exploration, this learned intermediate function or surrogate, is evaluated across a large set of parameter configuration candidates. The top K performing configurations according to the surrogate are chosen to continue sequential exploration for another iteration. In the context of machine learning, this method has been used to automatically con-

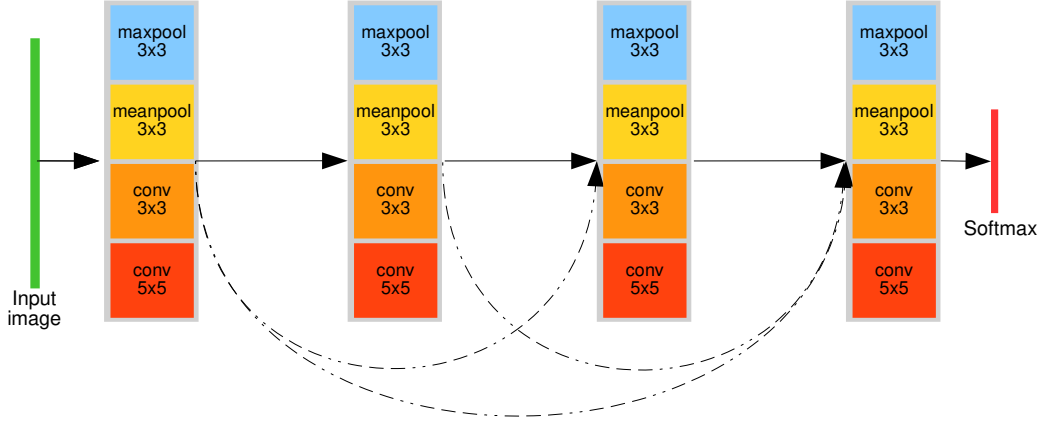


Figure 1. **Structure of the macro search space used in this paper.** In this example, for the sake of simplicity, the number of layers is $L = 4$, and the number of possible operations per-layer is 4. The dotted lines represent the skip connections that are also part of the search space.

struct ensembles of classifiers by Lacoste *et al.*, [13]. On a different line, an interesting surrogate-based method, denominated SMASH [4], displays strong results. The surrogate function of SMASH is not designed to rank tested configurations, but to predict the weights of sampled architectures. Through random search, SMASH is able to find competitive architectures in a number of datasets.

More recently, Liu *et al.*, [15] proposed to use SMBO-based progressive search (PNAS) within a constrained search space as in [31], achieving state-of-the-art results in image classification on CIFAR-10, while also being significantly less computationally intensive than [31]. The architecture hyperparameters learned for CIFAR-10 were successfully transferred to ImageNet with simple modifications. However, the method is still considerably slower (~ 200 times) than ENAS [18]. In this paper we are interested in assessing the inclusion of weight-sharing in SMBO-based approaches. Beyond that, we believe PNAS suffers from over-greediness as it merely chooses the top K architectures at each step of the progression of the method. In this paper we are also interested in checking whether surrogate-based sampling could lead to better results with respect to the greedy choice of configuration candidates.

3. Search space structure

In this section, we describe the search space used by our method. Following [18], we limit the search space of our algorithm to one of two possible structures. First, we consider what they denominate *macro* search space, spanning entire deep convolutional models with skip connections. Later on, we work on a *micro* search space, operating over modular convolutional cells.

Macro search space. First we focus on regular convolutional neural networks with variable skip connections and convolutional operations, namely the *macro* space (see Fig. 1). For this search space we start from a fixed convolutional module to expand the input image from 3 color channels to the fixed number of channels C that will take place along the network, with a maximum number L of convolutional layers. At each layer one of six possible operations is chosen. We follow [18] regarding the considered operations:

- 3×3 max pooling
- 3×3 mean pooling
- 3×3 convolution
- 5×5 convolution
- 3×3 depthwise convolution [5]
- 5×5 depthwise convolution [5]

At the output of the CNN described by a realization of the *macro* search space, a softmax layer processes the channel-wise global average of the last convolutional layer. This idea, introduced by [14], precludes the use of large fully connected layers that might prevent stable training during the architecture search. Overall, we can describe a layer l of a network in the *macro* space with a tuple (O_l, S_l) . Here, $O_l \in \mathcal{O}$, where \mathcal{O} is the set of possible operations. S_l is a variable-length tuple describing what previous layers are used as input to layer l . This allows the search space to contain neural networks with variable amounts of skip connections. Observe then that $S_l \subseteq (H_1, \dots, H_{l-1})$, where H_l is the output of layer l , avoiding unwanted loops in the underlying computational graph. To summarize, a whole network of L layers is fully described by a L -tuple $((O_1, S_1), \dots, (O_l, S_l), \dots, (O_L, S_L))$.

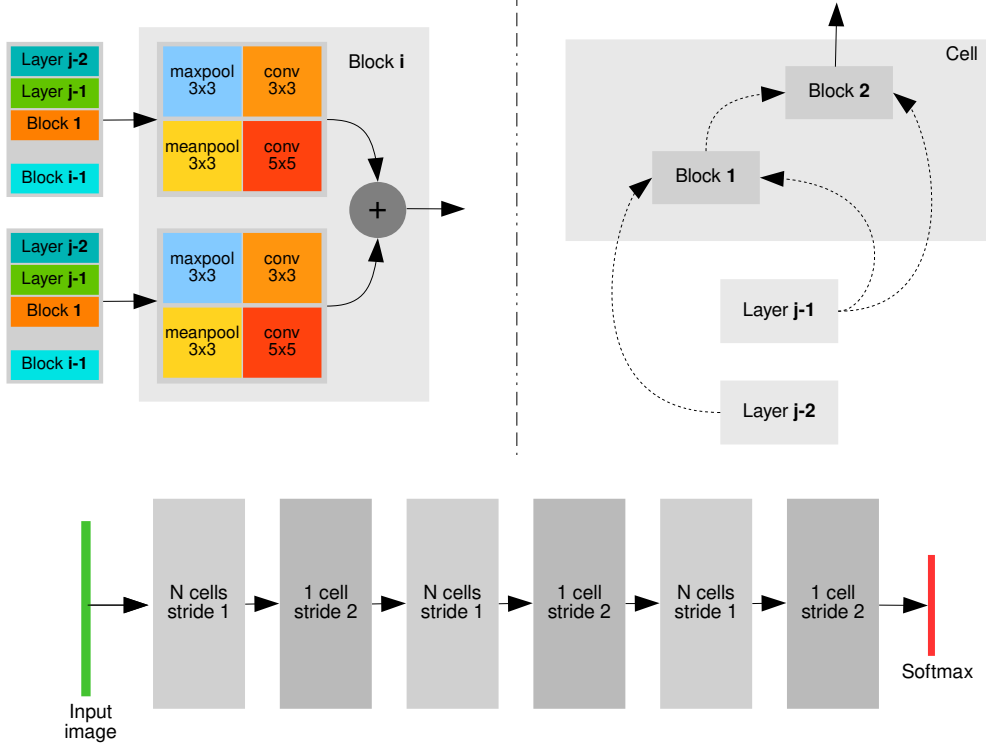


Figure 2. **Structure of the micro search spaces used in this paper.** Top row: Block structure on the left, cell structure on the right. Recall that a cell is formed by one or more blocks. In this example, the number of possible operations per block is 4. Observe that for any given block inside a cell, the possible inputs for the two operations of the block are the outputs of previous two layers and previous blocks of the same cell. Bottom row: a neural network in the *micro* search space is formed by stacks of cells with the same structure.

Micro search space. For the *micro* search space, we keep the fixed input convolution as in the *macro* space and employ, again, global average pooling of the last convolutional layer as input to the final softmax layer. Fig. 2 shows a diagram that explains the *micro* search space in detail. A neural network configuration stemming from a realization of the *micro* search space is formed by stacks of convolutional modules denominated **cells**. According to the network topology that is shown in the bottom row of Fig. 2, a cell is replicated several times. In such a way, a whole network configuration is described only by a cell structure.

As it can be appreciated in the bottom part of Fig. 2, to form a CNN from the micro space, two types of cell are stacked: *normal*, and *reduction*. The only difference between them is that the reduction cells use strides. The objective is to increase the receptive field of deeper layers in the network. Normal cells are stacked N times in between reduction ones.

Each one of these cells, as seen in the top row of Fig. 2 is formed by B blocks. We shall denote the space of possible blocks with \mathcal{B} . As opposite to the depth-only structure of the *macro* space, the block-based structure of the *micro*

space allows networks to go wider as needed. The structure of each block is defined by two convolutional operations, and their corresponding input connection. The possible operations for each block are the same ones than for the *macro* search space plus the identity transformation¹. On the other hand, the possible input connections for a given block in a cell are the outputs of the previous two cells plus the outputs of all previous blocks inside the cell. Finally, the two convolutional features of a block are summed up, as in [15]. This means we can describe cell c by a concatenation of B 4-tuples, one tuple with the form (O_1, O_2, S_1, S_2) for each block b . For the *micro* search space, $O_i \in \mathcal{O}$, and $S_i \in \{H_B^{c-2}, H_B^{c-1}, H_1^c, \dots, H_{b-1}^c\}$. Here, H_i^j is the output of block i at cell j . When constructing the final cell output, we simply concatenate all the outputs of blocks that were not used as input of any other block within the cell.

The two described search spaces end up being extremely large. The cardinality of the solution space of *macro* configuration is $\sim 2 \times 10^{29}$ when $L = 12$. On the other hand, the

¹The identity operation allows blocks to directly combine features from previous layers and blocks in a similar way to ResNets[7].

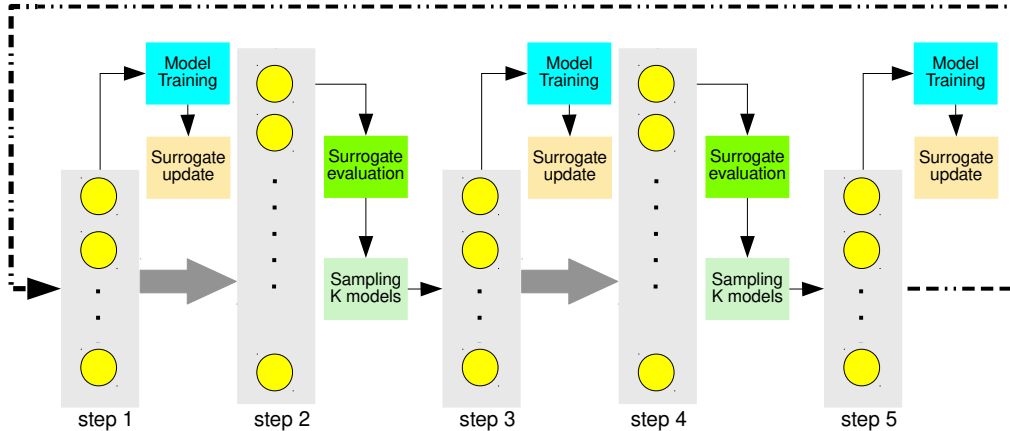


Figure 3. **Illustration of the progressive model-based search procedure used in this paper.** In this example, a maximum number of three blocks (if considering the *micro* space) or layers (*macro* space) are used. **Step 1:** We list all the possible configurations for the simplest complexity level ($B = 1$, or $L = 1$). Each of these models are trained sequentially by sharing-weights among themselves, for a small number of epochs. When validation errors are collected, the surrogate function is trained for the first time. **Step 2:** The second level of complexity is unrolled, leading to a total number of possible configurations of $|\mathcal{B}_1| \times |\mathcal{B}_2|$. Instead of training all these configurations, the surrogate function is used to sample K models. **Step 3:** K configurations are trained, and the surrogate is updated. **Step 4:** The previously sampled K configurations are combined with the new unroll of the architecture complexity. A subset of $\mathcal{B}_{1:3}$ is then evaluated by the surrogate and K new configurations are sampled. **Step 5:** the surrogate is updated again. This whole procedure is repeated for a small number of iterations (dotted arrow) while cooling down the sampling temperature.

solution set of the *micro* space is of cardinality $\sim 5 \times 10^{14}$ when we allow up to $B = 5$ blocks. To put those numbers in context, it can be noted that there are *only* $\sim 2 \times 10^{11}$ galaxies containing $\sim 10^{24}$ stars in the observable universe.

4. Efficient progressive neural architecture search

In this paper we adhere to the notion of PNAS [15] that highlights the difficulty of adequately exploring an exponential search space like ours in a direct manner. A more direct approach to neural architecture exploration as done by NAS [31] or ENAS [18] often requires sampling a large number of configurations. This is the main reason why we adopt a sequential exploration scheme.

Sequential search with shared weights. Under such a paradigm, we explore the search space at increasing levels of model complexity, starting with simple ones first. This means that, in the case of the *macro* search space we start the exploration by considering architectures with $L = 1$, while for the *micro* space we consider architectures with $B = 1$. The sampled neural architectures are then trained for a small number of epochs². This is possible because shared weights are refined further when new sampled archi-

tectures are trained³. The accuracy scores derived from a validation dataset are then used to train a surrogate function that predicts scores of new configurations (details on the surrogate function as described later on).

Subsequently, we take all the possible network configurations at the next level of complexity ($B = 2$ for *micro* and $L = 2$ for *macro*) and combine them with all the configurations of the previous step. At this point, the number of considered architectures is very high. Training all of them with modest hardware setups, even for very few epochs, would be too expensive. Instead, we sample K of them. The probability of architecture i with predicted accuracy s_i to be selected is $p_i = s_i / \sum_j s_j$.

Thus, our surrogate function predicts sampling probability of neural architectures. These K sampled configurations are then trained again for the same number of epochs (three). The newly obtained validation accuracy scores are used to update the surrogate function.

During following exploration steps, all the possible configurations that unfold at subsequent levels of complexity are combined with the currently sampled K architectures. The *sampling, training, and surrogate updating* loop continues until the maximum complexity level is reached. In Fig. 3 we explain in more detail the search procedure of EPNAS.

²While [18] only trains sampled architectures for a single epoch, we found our method to behave better by training for three epochs.

³Two sampled networks share convolutional weights if the same convolutional operations at the same depth are used by the two of them.

Table 1. **Neural architecture search on the CIFAR10 dataset.** We present final validation errors for a number of methods, including our best architectures from the *macro* and *micro* search spaces. For the reported neural search methods, we provide with time duration of the search and the number of used GPUs.

Method	GPUs	Time (days)	Parameters (millions)	Error (%)
ResNet [7]	-	-	1.7	6.43
DenseNet [9]	-	-	25.6	3.46
Super Nets [27]	-	-	-	9.21
ConvFabrics [21]	-	-	21.2	7.43
SMASH [4]	1	1.5	16.0	4.03
QNAS [2]	10	10	11.2	6.92
NAS [30]	800	28	37.4	3.65
ENAS macro [18]	1	0.32	21.3	4.23
EPNAS macro (ours)	1	1.2	5.9	5.14
EPNAS macro (ours + more channels)	1	1.2	38.8	4.01
NASNet micro [31]	450	4	3.3	3.41
ENAS micro [18]	1	0.45	4.6	3.54
PNAS micro [15]	100	1.5	3.2	3.63
EPNAS micro (ours)	1	1.8	1.6	5.69
EPNAS micro (ours + more channels)	1	1.8	6.6	3.71

Observe that, departing from PNAS [15], we only train sampled architectures for a small number of epochs. The noisy estimation of sampled architecture accuracy is attenuated by implementing iterations on the sequential exploration. In our experiments, we repeat the whole process described before (the sequential exploration from simplest to highest level of network complexity) up to five times. More importantly, we adopt the graph-based interpretation of the search space proposed by ENAS [18], allowing us to employ their weight-sharing strategy. In this way, the more an operation at a layer or block is used by sampled architectures, the more its weights are updated. Effectively, search is biased towards architectures with previously trained modules as long as they have been refined more through gradient descent (presenting better classification accuracy scores). This is why in our implementation, probabilistic random sampling is preferred over simply taking the top K performing architectures. Random sampling allows our method to escape local minima. We implement a temperature-driven sampling procedure, so the search space is explored more randomly first, while fully trusting the surrogate function at the end. The final sampling probability of architecture i is given by $p_i^{1/\tau} / \sum_j p_j^{1/\tau}$ with temperature τ decaying quickly to one as the number of iterations increases. In this sense, EPNAS lies in between a greedy progressive approach and purely random search.

Surrogate function. The progressive search used in this paper, together with the type of architectures with variable length and connectivity we deal with, make of *recurrent neural networks* an ideal candidate for implementing our

surrogate function. This is a choice that is common in the literature [31, 29, 18, 15]. In practice, we use a LSTM [8] network.

The actual input to the LSTM is computed with a small linear layer that acts as decoder for the symbol array describing the network configuration. Another linear layer with a sigmoid non-linearity processes the last hidden state of the LSTM. The output is a scalar in $[0, 1]$ representing the accuracy score on the validation dataset of the input neural network described by the input string describing a neural configuration. The input dataloader for the LSTM is implemented with a hash-table so we can keep control of repeated network configurations. The surrogate function is trained with Adam [10] with a $L1$ loss.

5. Experiments

Experimental setup. In this paper, neural architecture search for image classification is evaluated on CIFAR-10 [11], as it is common practice [31, 15]. This dataset contains 50.000 training images and testing 10.000 images. As it is standard for image classification, the images are normalized for training. The dataset is augmented by performing random-cropping and random mirroring.

Implementation details. Since we use shared weights, we need to guarantee that the number of input channels remain the same for operations at the same block or layer. We achieve this by including 1×1 convolutions in between cells and layers, for the *micro* and *macro* search spaces respectively. This structure was originally proposed by [18]. The

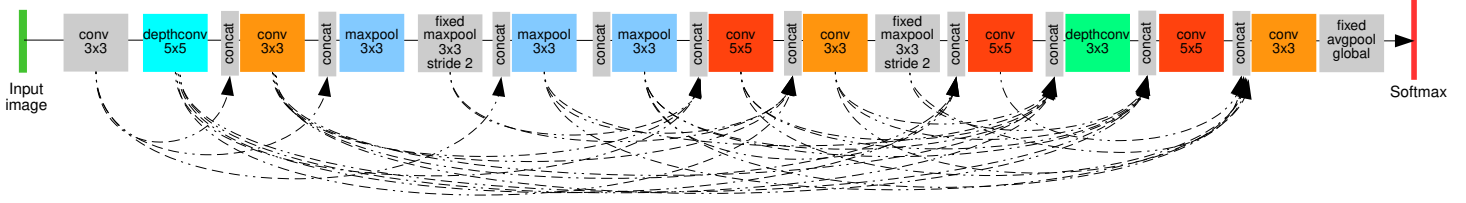


Figure 4. Best found CNN in the *macro* search space.

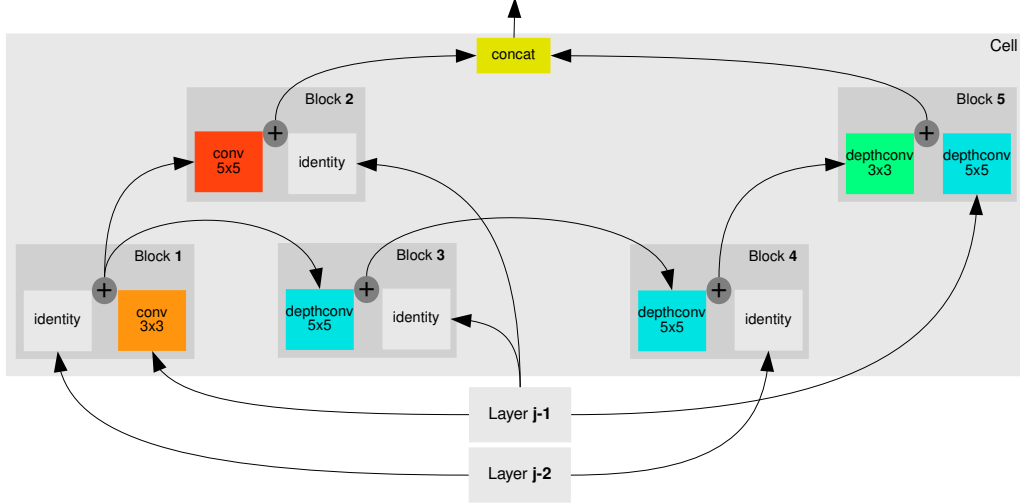


Figure 5. Best found cell in the *micro* search space.

weights of such intermediate convolutions are also shared among sampled architectures.

The surrogate function is a LSTM with 100 hidden units, processing an input feature of size 100. The weights of the linear layers are initialized randomly with a uniform distribution between $[-1, 1]$. During exploration, $K = 25$ network configurations are sampled at each step. These configurations are trained sequentially for three epochs. The number of channels for the convolutional layers is 24 during exploration. For the *micro* search space, the number of stacked normal cells is $N = 2$, while the maximum number of blocks is $B = 5$. During the final training of the found architectures, we let $N = 3$ and the initial number of convolutional channels to 96 for the *micro* space. On the other hand, for the *macro* search space, we set the number of layers to $L = 12$, and the initial number of channels to 200. We also evaluate the found configurations with a larger number of channels (512 for the *macro* space, and 128 for the *micro*). Models with fewer channels are faster but less accurate, while models with more channels turns into higher accuracy at the cost of being slower. The convolutional weights are learned under the same optimizer configuration and learning rate schedule of [31]. Furthermore, we perform three to five iterations of our progressive search. In our experiments, three iterations always performed around

8% better than a one-pass progressive search, even when the sampled models were trained for more epochs.

Discussion. Our results are shown in Table 1 along with several baselines and other state-of-the-art algorithms. The first group in Table 1 presents results of recent hand-designed neural architectures. In particular, we show results by ResNet [7], the work that introduced residual skip connections. Along with ResNets, impressive results for image classification by DenseNets [9] are presented. The second group in Table 1 is conformed by other neural search methods, spanning several different kind of techniques. All of those methods aim at designing full networks, as in our *macro* space. The third group is formed by more direct competitors of our method on the *micro* search space. Indeed, PNAS [15] and NASNet [31] search neural architectures within a very similar search space to ours.

It must be observed that **our method delivers competitive results with respect to PNAS**, while being **close to a hundred times more efficient in terms of GPU/days**. The best found CNN from the *macro* space is represented in Fig. 4, while the best found cell from the *micro* space can be seen in Fig. 5. Moreover, even though NASNet [31] finds architectures that perform better on the validation set, our method is almost 900 times faster. Our results, not be-

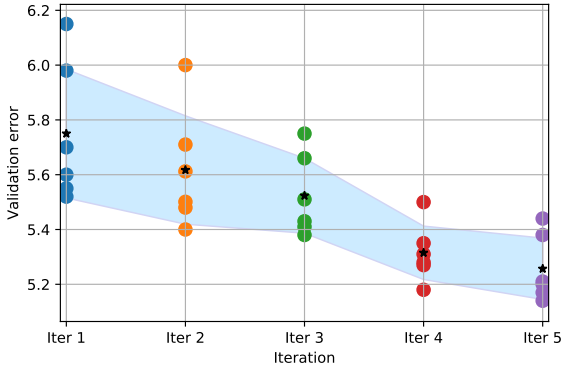


Figure 6. **Behaviour of EPNAS across multiple parallel runs and number of iterations on the macro space.** Dots indicate independently run instances of EPNAS. Black star indicates the mean for runs with the same number of iterations and blue shadow the variance.

ing too far from the absolute top performer, present convenient trade-off between number of parameters and performance, as it can be appreciated by our experiments with more channels. It can also be pointed out that better results are expected with larger K . However, we noticed that the increase on exploration cost are large. With more than one available GPU, one could run several search threads in parallel and choose the best found architecture from all of the them. The behaviour of EPNAS among threads can be seen in Fig 6. Observe that for few iterations the variance of the accuracy of found architectures is relatively high. This can be explained by the increased sampling temperature. As the number of iterations increases and the sampling temperature decreases, the variance is reduced. Observe that different architectures offer relatively similar validation accuracy. This is likely due to the shape of the constrained search space, which increases the likelihood of randomly sampled architectures to perform well. Overall, runs of EPNAS with more iterations seem to perform generally better.

6. Conclusions

In this paper we have introduced a new method for neural architecture search that works well on modest hardware configurations with a single or few GPUs only. Our method is based on a progressive paradigm by means of the *Sequential Model-based Optimization* technique. To enable faster exploration, we adopt a recently-proposed heuristic from the state-of-the-art that consists of sharing weights among sampled architectures during exploration. This allows search methods to circumvent the need for complete training of sampled architectures, a common bottleneck in neural architecture search. The savings on computational

expense allow us to explore modifications to the original sequential approach. We introduce search iterations and surrogate-based probabilistic sampling of network configurations as opposite to a one-shot greedy-selection of top architectures at each exploration step. We demonstrate on the challenging CIFAR-10 dataset that our method is able to find very competitive architectures with results that are not far from the state-of-the-art. With this paper, we also demonstrate the applicability of weight-sharing for neural architecture search methods based on progressive search. Future work includes the exploration of our efficient progressive neural architecture search (EPNAS) for a wider variety of learning problems.

References

- [1] P. J. Angeline, G. M. Saunders, and J. B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(1):54–65, 1994.
- [2] B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning, 2016.
- [3] J. Branke. Evolutionary algorithms for neural network design and training. In *Proceedings of the First Nordic Workshop on Genetic Algorithms and its Applications*. Citeseer, 1995.
- [4] A. Brock, T. Lim, J. M. Ritchie, and N. Weston. Smash: one-shot model architecture search through hypernetworks. In *International Conference on Learning Representations*, 2017.
- [5] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *Computer Vision Pattern Recognition*, 2016.
- [6] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *International Conference on Computer Vision*, pages 1026–1034, 2015.
- [7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Computer Vision Pattern Recognition*, pages 770–778, 2016.
- [8] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [9] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. In *Computer Vision Pattern Recognition*, volume 1, page 3, 2017.
- [10] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [11] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009.
- [12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Conference on Neural Information Processing Systems*, pages 1097–1105, 2012.
- [13] A. Lacoste, H. Larochelle, F. Laviolette, and M. Marchand. Sequential model-based ensemble optimization. *arXiv preprint arXiv:1402.0796*, 2014.
- [14] M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

- [15] C. Liu, B. Zoph, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. *arXiv preprint arXiv:1712.00559*, 2017.
- [16] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, A. Navruzyan, N. Duffy, and B. Hodjat. Evolving deep neural networks. *arXiv preprint arXiv:1703.00548*, 2017.
- [17] R. Negrinho and G. Gordon. Deeparchitect: Automatically designing and training deep architectures. *arXiv preprint arXiv:1704.08792*, 2017.
- [18] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.
- [19] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2018.
- [20] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, Q. Le, and A. Kurakin. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, 2017.
- [21] S. Saxena and J. Verbeek. Convolutional neural fabrics. In *Conference on Neural Information Processing Systems*, pages 4053–4061, 2016.
- [22] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.
- [23] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Conference on Neural Information Processing Systems*, pages 2951–2959, 2012.
- [24] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Prabhat, and R. Adams. Scalable bayesian optimization using deep neural networks. In *International Conference on Machine Learning*, pages 2171–2180, 2015.
- [25] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.
- [26] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, et al. Going deeper with convolutions. In *Computer Vision Pattern Recognition*, 2015.
- [27] T. Veniat and L. Denoyer. Learning time-efficient deep architectures with budgeted super networks. *CoRR*, 2017.
- [28] L. Xie and A. Yuille. Genetic cnn. *International Conference on Computer Vision*, 2017.
- [29] Z. Zhong, J. Yan, and C.-L. Liu. Practical network blocks design with Q-learning. In *AAAI Conference on Artificial Intelligence*, 2018.
- [30] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.
- [31] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Computer Vision Pattern Recognition*, 2018.