Improving the Efficient Neural Architecture Search via Rewarding Modifications

Ignazio Gallo, Gabriele Magistrali, Nicola Landro and Riccardo La Grassa
Department of Theoretical and Applied Science
Varese, Italy
{ignazio.gallo, nlandro, rlagrassa}@uninsubria.it

***Abstract*—Nowadays, a challenge for the scientific community concerning deep learning is to design architectural models to obtain the best performance on specific data sets. Building effective models is not a trivial task and it can be very time-consuming if done manually. Neural Architecture Search (NAS) has achieved remarkable results in deep learning applications in the past few years. It involves training a recurrent neural network (RNN) controller using Reinforcement Learning (RL) to automatically generate architectures. Efficient Neural Architecture Search (ENAS) was created to address the prohibitively expensive computational complexity of NAS using weight sharing. In this paper we propose Improved-ENAS (I-ENAS), a further improvement of ENAS that augments the reinforcement learning training method by modifying the reward of each tested architecture according to the results obtained in previously tested architectures. We have conducted many experiments on different public domain datasets and demonstrated that I-ENAS, in the worst-case reaches the performance of ENAS, but in many other cases it overcomes ENAS in terms of convergence time needed to achieve better accuracies.**

***Index Terms*—Neural Architecture Search, Reinforcement Learning, ENAS, Auto-ML, Classification**

I. Introduction

Despite widespread success in difficult tasks related to image, speech and text recognition, neural networks face a major challenge in the creation of an architecture capable of achieving high accuracy. The effort needed to design the architecture by hand is well-known and it is very time-consuming. Recent studies use sophisticated methods, called neural architecture search (NAS) from its first implementation [1], seeking to find the optimal architecture in a designated search space starting from scratch, while training each designed architecture on the real data to get a validation performance and lead exploration [2] [3] [4] [1] [5] [6] [7] [8]. In literature it exists a wide categorized area of NAS, that automatically finds effective models capable to be competitive or overcome the models built ad-hoc by humans, based on neuro-evolution [9] [10], Bayesian optimization [11] [12], Monte Carlo Tree Search [13] and Reinforcement Learning. (RL) [1] [14] [15] [16]. Many improvements to this method were proposed later, mainly dealing with improving its performance. A major result of this effort is found by ENAS (Efficient NAS) [8], which uses transfer learning [17] to allow newly generated networks to be initialized with the trained weights from previously trained networks, reducing the average training time needed to find a good candidate for the final architecture. In this paper, we propose an improvement of ENAS [8] called I-ENAS (Improved-ENAS). Reinforcement learning treats the validation set performance as the reward, in our approach we modify the reward with a function built ad-hoc, by penalizing or rewarding all the architectures that offers worse/better performance in terms of accuracy when compared to previous architectures.

978-1-7281-8579-8/20/$31.00 ©2020 IEEE

II. Related Work

Our work is based on ENAS [8], which relies on transfer learning to improve on NAS [1]. In ENAS [8] the main actor is the controller LSTM network, which has the task of finding the best architectures by searching for the optimal sub-graph inside a larger directed acyclic graph (DAG) representing the entire search space.

The learning process of the controller over epochs can be divided into two phase.

In the first phase, the controller samples a ”child” architecture by generating a sequence of numbers that describe the child model. Then, the child is trained on a batch of training data. This process is repeated for the entire training data. The main advantage of ENAS over NAS comes into play during this phase, because every time a child architecture is trained, the weights for each individual component are saved and later reused to initialize the weights of future child architectures, assuming any components are shared between them. In the second phase, the controller extracts a child architecture in the same fashion and then tests it on a batch of random validation data. The accuracy obtained from validation test is then used as the reward to train the controller itself, resulting in better architectures being extracted over time.

Rather than designing an entire network, ENAS designs only a ”cell”. ENAS designs two different kinds of cells, convolutional and reduction, which are stacked to form the final architecture. This technique (see Fig. 1) is called ”micro search” and allows to find deep networks without needing a very large search space and cutting down on search times.

There are systems which took similar approaches to NAS, but rather than focusing on lower training times like ENAS did they looked for a different method of generating the architectures. These systems tend to increase the size of the

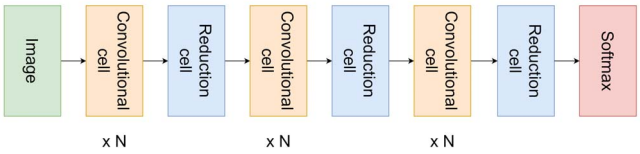

Fig. 1. How ENAS [8] composes a deeper network from the two kinds of cells it designs.

search space and give to the generator more freedom in how to arrange the connections between components.

In [2], authors introduce Randomly Wired Neural Networks, which use classical random graph models to find randomly wired graphs representing the networks. Their work suggests that focusing on a better networks generator may provide new breakthroughs in found architectures, thanks to less constrained search spaces. Similarly to architecture search methods that use reinforcement learning, in [4], authors uses a different gradient-based search method called DARTS(Differentiable ARchiTecture Search). This uses an approach built on a continuous relaxation of the architecture representation as if it were an operation on a direct acyclic graph (DAG), placing multiple candidate operations on each edge. At the end of the search, the final architecture is built from the set operations which ended up with the highest weight among every edge from the DAG architecture representation.

SGAS [3] (Sequential Greedy Architecture Search) begins with a search space comparable to the ones used by DARTS [4], NAS and ENAS, but over the training, at fixed intervals, it has a phase where it computes a score for each edge based on its importance and its entropy. Once found the edge with the highest score, this gets its operation with the maximum weight, fixing it for the rest of the training. This means that as the training proceeds, the authors adopt a pruning methodology to cut off some of the edges obtaining at the end a final architecture.

In PNAS [5] (Progressive Neural Architecture Search) the authors try to find structures identical to ENAS (repeating cells containing blocks of two operations), but choose a different route in their search strategy. They propose a heuristic search, which starts with simple and shallow models and progresses to more complex ones, removing less-performing structures as it goes. This progressive approach allows the authors to achieve higher efficiency than NAS while discovering state-of-the-art networks on Cifar-10 and ImageNet.

An interesting approach is shown in Neural Architecture Optimization [18] (NAO). Rather than looking at architectures in a discrete way like previously described works, NAO [18] has its basis in continuous optimization. It has three major components: an encoder which map the neural architectures to a continuous space; a predictor which takes this continuous representation and predicts its accuracy; a decoder which takes the continuous representation and translates it back to an architecture. With these parts the authors are able to perform gradient-based optimization in the continuous space, potentially finding a new representation with higher accuracy which gets translated back into an architecture. Their tests were able to confirm the effectiveness of their solution, reaching state-of-the-art results in less than a GPU day.

There have been several efforts [19] [20] to adapt evolutionary algorithms to neural architecture search, through an approach called *neuro-evolution of topologies*. However these attempts never reached the accuracy of human-designed networks. AmoebaNet-A [21] is a recent successful attempt at this, even achieving state-of-the-art performance on ImageNet. While searching through the NAS search space, they propose a change to the *tournament selection* evolutionary algorithm to make it show bias to younger genotypes. Eventually this results in a high quality model, but carries the high computational requirements of other evolutionary algorithms and finds itself surpassed by many of the other works cited previously.

III. Proposed Approach

In I-ENAS we take the Reinforcement Learning approach used in NAS and ENAS and add a system which changes the reward given to the LSTM controller depending on results achieved by previous networks.

Our code [22] is based on the ENAS implementation in TensorFlow provided by Minguk Kang [23], which in turn is a modified version of the original code by Melody Guan [24].

It works (as explained in algorithm 1 and shown in Fig. 3) by comparing the accuracy acc of the current network with the moving average avg of the previous 10 networks accuracies to understand how well the current architecture is doing and modify the reward r accordingly. The accuracy acc of the current network is normalized by the average accuracies avg and assigned to s, which is compared with the threshold t. If s is lower than t, then the reward is set to 0.01 to penalize the network for the controller LSTM; if s is higher than t, the reward r is set to the current accuracy multiplied by the factor s, obtaining a decrease for accuracies below the average avg and an increase for accuracies above it. Formally, we define the following reward function τ:

$$\tau(s,t) = \begin{cases} 0.01 & \text{if } s \leq t \\ acc \cdot s & \textit{otherwise} \end{cases} \tag{1}$$

where $s = \frac{acc}{avg}$ and $avg = \overline{accs}$.

In algorithm 1, there are a few functions which are meant to represent normal operations in ENAS which we kept very abstract to focus the algorithm on our additions. These are:

- controller.ExtractArc(): asks the controller LSTM to sample a sequence to form a child architecture.
- childArc.TrainWeights(): trains the child architecture and saves the weights, allowing for weight sharing with future child architectures.
- childArc.Test(): tests the child architecture on a batch of the validation set, obtaining its accuracy.

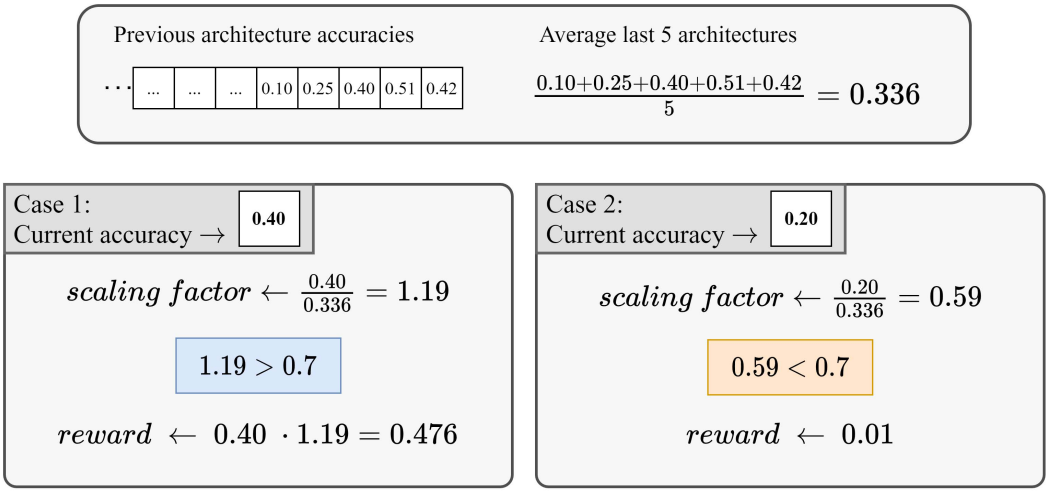

Fig. 2. An example of how two networks with differing accuracies (bottom left and bottom right) result in very different rewards from the same previous situation (top)

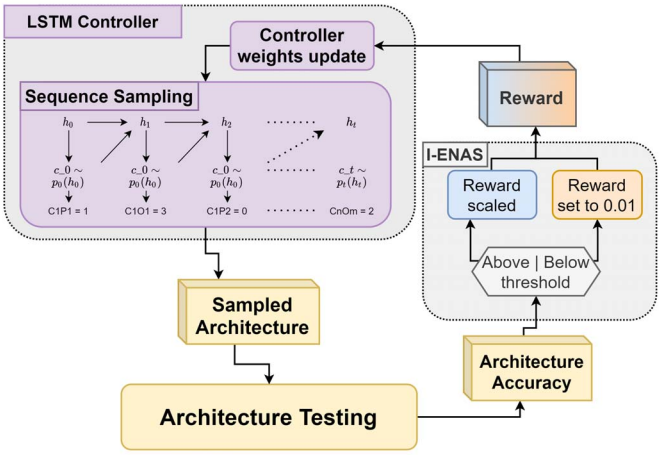

Fig. 3. The loop performed during controller training for I-ENAS.

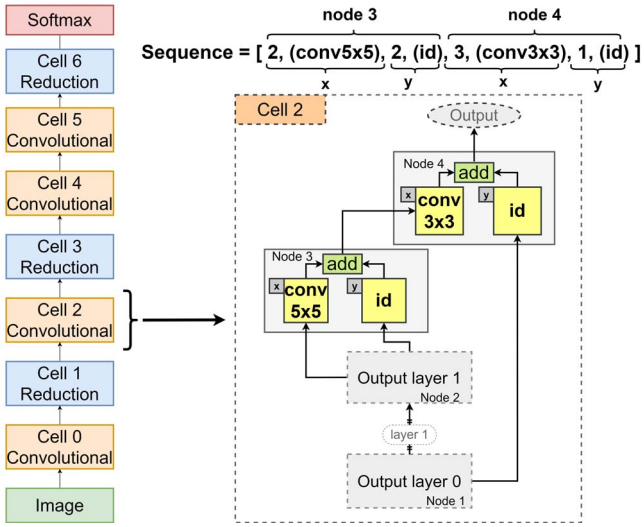

Fig. 4. Simplified example of the translation of a generated sequence into a convolutional cell. In the actual code the operations are also represented by numbers, rather than the text used here for convenience.

IV. Datasets

To evaluate the improvements, we tested on three well-known benchmarks:

- MNIST [25], which is composed of 60,000 28×28 gray-scale images of the 10 digits and a test set of 10,000 images. Due to inner workings of ENAS [8] itself, the images had to be resized to 32x32 by padding them with copies of the border pixels.
- Fashion-MNIST [26], which has identical composition as MNIST but features 10 classes of items of clothing rather than 10 digits. The same resize operation applies.
- Cifar10 [27], a dataset of 50,000 32×32 RGB images with 10,000 test images, labeled in 10 categories.

In ENAS beside the train-validation split we have an additional split for the datasets, where 30% of the validation data is put in a test set. The test set is used between epochs to give an estimation of the current accuracy on data never seen by either child architectures or the controller.

V. Experiments

During our experiments we created networks made up of 6 convolutional and 3 reduction cells, each one containing a sub-network of 5 blocks with 2 operations per block. An example of this can be seen in Fig. 5.

The possible operations for ENAS are: convolutions with a 5×5 or 3×3 kernel sizes, average pooling with a 3×3 kernel size, max pooling with a 3×3 kernel size or identity (output is equal to the input). Our training parameters are taken from the ones that ENAS uses: we trained the shared parameters ω

Input : accs = empty list
childSteps = $\lceil datasetSize/batchSize \rceil$
controllerSteps = 30
epochs = 500
t = 0.7
$upperBound = 10$
Output: A good architecture for a classification problem
for $i \leftarrow 0$ **to** $epochs$ **do**
  for $cStep \leftarrow 0$ **to** $childSteps$ **do**
    $childArc \leftarrow controller.ExtractArc()$;
    $childArc.TrainWeights()$;
  end
  for $contStep \leftarrow 0$ **to** $controllerSteps$ **do**
    $childArc \leftarrow controller.ExtractArc()$;
    $acc \leftarrow childArc.Test()$;
    $r \leftarrow acc$;
    if $accs.size > 0$ **then**
      $avg \leftarrow \overline{accs}$;
      $s \leftarrow \frac{acc}{avg}$;
      if $s \leq t$ **then**
        $r \leftarrow 0.01$;
      else
        $r \leftarrow acc \cdot s$;
      end
      if $accs.size == upperBound$ **then**
        $accs.RemoveOldest()$;
      end
    end
    $accs.push(acc)$;
    $controller.train(r)$;
  end
end

Algorithm 1: The proposed Improved-ENAS (I-ENAS).

of the child networks with Nesterov momentum [28] and we use the Adam optimizer [29] with learning rate of 0.0035, a cosine function useful to learning decay and a weight decay $l_2 = 0.0001$, while the weights ω are initialized as [30]. The controller parameters θ are instead initialized uniformly in $[-0.1, 0.1]$. Finally we apply a temperature of 5.0 and a tanh constant of 2.5 to the output of the controller and add 0.0001 of its entropy to the reward.

To better understand the plots 6, 7 and 8: the data-points represent the accuracies achieved by the networks extracted during controller training, with the orange ones being I-ENAS's and the grey ones being ENAS's. The two colored lines are moving averages over 100 data points, to provide a quick comparison of the two methods.

Another aspect we wanted to highlight is how quickly each system reached its maximum in terms of accuracy, which is done through a marked data-points for comparison between the two systems as in Fig.6, 7 and 8.

ENAS (and thus I-ENAS) does not perform hyper-parameters optimization as part of its algorithm but leaves

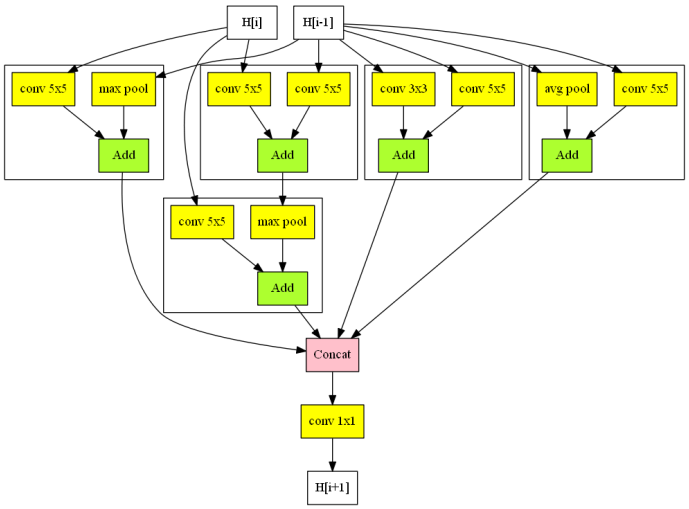

Fig. 5. The reduction block generated by I-ENAS during the Cifar10 experiment V-C, which formed the highest accuracy network found.

it as a followup task once a suitable network is found. For this reason we decided to treat each dataset as if it were a completely unknown classification task and we use the same hyper-parameters for all the tests. This is why the values reached are, except for MNIST, not quite at the state-of-the-art that ENAS can reach after hyper-parameter optimization, but it helps to show that I-ENAS's work can provide solid architectures faster than ENAS, helping the following training and optimization phases.

A. MNIST

In MNIST (Fig. 6) both systems show a very rapid convergence to their maximum accuracy due to the simplicity of the classification task itself. However even in this case it is possible to notice the difference between them, with I-ENAS consistently offering better results and reaching its maximum before ENAS. The 100% accuracy reached is caused by testing the model only on a random batch of 128 data (validation set). However, we emphasize that for each extracted architecture we use a different random batch to evaluate them independently of each other.

B. FashionMNIST

Despite its similarities to the previous dataset, FashionMNIST (Fig. 7) proved to be a far more complicated task and it shows in the results. Despite I-ENAS achieving slightly superior results and hitting accuracies that are close to the state-of-the-art, there isn't such a noticeable difference in results, but it helps show that I-ENAS keeps a performance level at least on par with ENAS. Table I then shows that despite the much closer results than other methods, I-ENAS is still showing a slight edge over ENAS.

C. Cifar10

In Cifar10 (Fig.8) I-ENAS proved to be able to reach a higher maximum accuracy than ENAS and provide consistently better architectures throughout its execution, reaching

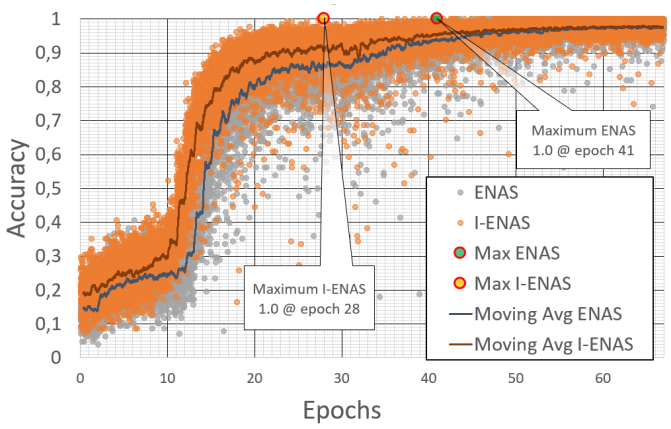

Fig. 6. Comparison of ENAS [8] and I-ENAS on the MNIST dataset after 67 epochs

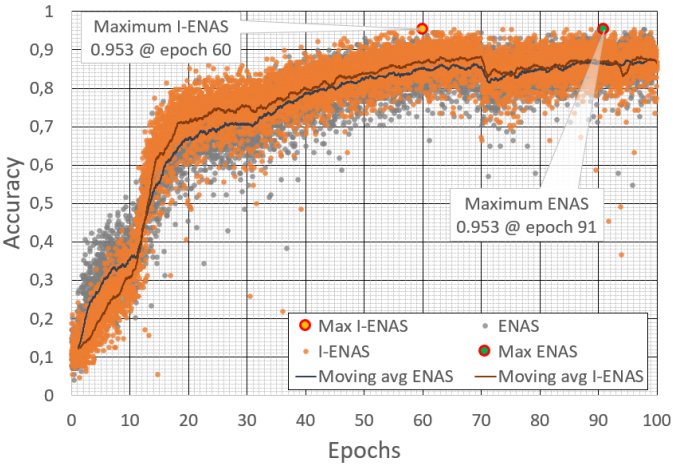

Fig. 7. Comparison of ENAS [8] and I-ENAS on the FashionMNIST dataset after 100 epochs

an accuracy of 95.3% over the maximum of 91.4% reached by ENAS. The occasional drops that can be seen are attributed to the learning rate schedule inherited from ENAS.

D. Comparative analysis of accuracy on extracted architectures

In this experiment, we analyse and compare I-ENAS's and ENAS's convergence in terms of accuracy of all architectures found at a fixed epoch using our reward function τ. We show interesting results as we report in Table I, where we calculated the average accuracy and the standard deviation of the accuracies during a single epoch for each of the experiments. Comparing the two approaches we observe an interesting results in terms of average accuracy and in variance. Clearly, we observe a high capability to extract better architectures from our proposal rather than the architectures extracted by ENAS. The low standard deviation reported using I-ENAS demonstrates a higher capability to extract architectures greatly

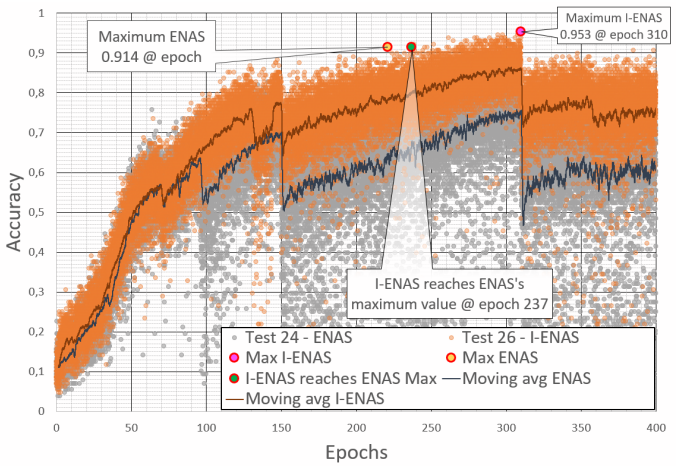

Fig. 8. Comparison of ENAS [8] and I-ENAS on the Cifar10 dataset after 400 epochs

TABLE I
ACCURACY AND STANDARD DEVIATION FOR THE THREE DATASETS AT THE EPOCHS WHERE I-ENAS FIRST REACHED ITS MAXIMUM

		MNIST ep.28	FMNIST ep.60	Cifar10 ep.310
Avg	I-ENAS	**0.915**	**0.866**	**0.856**
	ENAS	0.860	0.852	0.754
STD	I-ENAS	**0.044**	**0.031**	**0.041**
	ENAS	0.066	0.034	0.080

performing in terms of accuracy than those obtained by NAS methodology.

VI. CONCLUSION

As we reported in our experiments, I-ENAS has the capability to achieve better results in terms of accuracy showing a high convergence accuracy between all architectures extracted than the architectures obtained by ENAS, allowing for a quicker transition to the following training and optimization tasks needed to obtain the desired network. Our approach leverages a simple weighted function that assigns a modified reward to the controller responsible to extract good architectures using the accuracies of the last N tested networks. This suggests that a system which modifies the controller's reward can end up being a valuable asset to improve the reinforcement learning methods based on NAS. There are still several avenues of research on this topic, for instance, it is possible to use any other function to assign a reward or using other approaches to handle automatically the way in which the system can attribute that reward. Future work can focus on a dynamic methodology to adapt the policy of reward in run-time and verify if it is possible to obtain an even faster convergence in terms of accuracy.

REFERENCES

[1] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2016.
[2] S. Xie, A. Kirillov, R. Girshick, and K. He, "Exploring randomly wired neural networks for image recognition," 2019.

[3] G. Li, G. Qian, I. C. Delgadillo, M. Mller, A. Thabet, and B. Ghanem, "Sgas: Sequential greedy architecture search," 2019.

[4] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," 2018.

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

[6] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," *arXiv preprint arXiv:1707.04873*, 2017.

[7] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proceedings of the aaai conference on artificial intelligence*, vol. 33, pp. 4780–4789, 2019.

[8] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," 2018.

[9] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, and A. Kurakin, "Large-scale evolution of image classifiers," *arXiv preprint arXiv:1703.01041*, 2017.

[10] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," *arXiv preprint arXiv:1711.00436*, 2017.

[11] T. Domhan, J. T. Springenberg, and F. Hutter, "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[12] H. Mendoza, A. Klein, M. Feurer, J. T. Springenberg, and F. Hutter, "Towards automatically-tuned neural networks," in *Workshop on Automatic Machine Learning*, pp. 58–65, 2016.

[13] R. Negrinho and G. Gordon, "Deeparchitect: Automatically designing and training deep architectures," 2017.

[14] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710, 2018.

[15] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," *arXiv preprint arXiv:1611.02167*, 2016.

[16] Z. Zhong, J. Yan, and C.-L. Liu, "Practical network blocks design with q-learning," *arXiv preprint arXiv:1708.05552*, vol. 1, no. 2, p. 5, 2017.

[17] B. Zoph, D. Yuret, J. May, and K. Knight, "Transfer learning for low-resource neural machine translation," 2016.

[18] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, "Neural architecture optimization," 2018.

[19] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, and B. Hodjat, "Evolving deep neural networks," 2017.

[20] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," 2017.

[21] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," 2018.

[22] G. Magistrali, "Improved enas," 2020. https://gitlab.com/g.magistrali/improved-enas.

[23] M. Kang, "Enas-tensorflow," 2018. https://github.com/MINGUKKANG/ENAS-Tensorflow.

[24] M. Guan, "Efficient neural architecture search via parameter sharing," 2018. https://github.com/melodyguan/enas.

[25] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[26] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[27] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," *Technical Report TR-2009, University of Toronto, Toronto*, 2009.

[28] Y. E. Nesterov, "A method for solving the convex programming problem with convergence rate $o(1/k^2)$.," *Soviet Mathematics Doklady*, vol. 27, pp. 372–376, 1983.

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," 2015.