Final project literature review

MGU - hyper-parameter self-learning using Multi Gated Units in deep neural graphs

XCS229ii Autumn 2020

Andrei Ionut Damian
Lummetry.AI
Bucharest, Romania
andrei@lummetry.ai

Laurentiu Piciu
Lummetry.AI
Bucharest, Romania
laurentiu@lummetry.ai

Abstract

Training deep neural networks requires knowledge and ample experimental time, as well as computational resources due to the search process of the architectural design. In this work, we review the current main research directions in the area of network architecture and finally propose in contrast a novel architecture, namely *MultiGatedUnit*, that uses directly learnable self-gating mechanisms for automated graph topology generation.

1 Introduction and Problem Definition

Training deep neural networks is not as straightforward as just simply adding computational elements within the network. Finding the optimal architecture, i.e. hyperparameter optimization, is an active research topic and is also a major time consumer in practice, requiring vast knowledge, ample experimental time and a lot of computation resources. Although there are proposed methods for avoiding exhaustive search and even advanced ones such as AutoML, this drawback is far from solved from the perspective of computational efficiency and "green" Deep Learning. The second issue is related to the "copycat" approach within each individual section of a neural graph due to the fact that the same module is repeated a certain number of times within a neural graph section. This architectural constraint implies a clear limitation to the class of hypotheses. Considering the possibility that the optimal hypothesis might need unique structures for each individual module throughout the whole deep neural graph, this ideal hypothesis will never be found.

In this work, we propose a novel architecture (called *MultiGatedUnit - MGU*) that encapsulates any kind of neural computational layer (linear, convolutional, etc.) that is able to self-learn the optimal modules configuration for the encapsulated layer through self-gating mechanisms. We argue that the proposed architecture drastically eliminates the need for grid search approaches required to find the optimal architecture of a deep neural graph. Using just one end-to-end optimization process, the deep neural graph using *MGUs* will learn a unique individual module structure for all modules within the graph. Beside finding a potentially ideal architecture for the proposed task we dramatically reduce the carbon footprint of the whole model generation process by eliminating the need for multiple end-to-end optimizations.

The area of neural graph topology architecturing is a long researched subject and it is still a hot one. Early related work approaches network topology generation based on genetic algorithms as an alternative to the classic direct hyper-parameter space search (Miller, G.F.; Todd, P.M.; Hegde (1989), Stanley and Miikkulainen (2002), Wierstra et al. (2005), Floreano et al. (2008)). More recent work in the area of learning network topologies include advanced evolutionary algorithms that minimize the process of searching within the hyperparameter space and generate successful architectures by treating the validation performance as fitness. One such published research is the one that generated the successful deep vision model *AmoebaNet-A* by Real et al. (2019) from Google Brain, but there exist other modern evolutionary research: Suganuma et al. (2018).

Another approach for the intelligent and automated search in hyper-parameter space is based on viewing the hyper-parameter choosing as an action performed by an actor resulting in an architectural state of the network and finally in a reward based on the architecture's validation accuracy/performance (Baker et al. (2017), Zoph and Le (2017), Zoph et al. (2018), Pham et al. (2018), Zhong et al. (2018)).

Besides evolutionary and RL-based approaches, there exist other methods that treat the architecture search as a black-box optimization problem

over a discrete domain, which leads to a large number of architecture evaluations required (random search [Bergstra and Bengio (2012)], MCTS [Negrinho and Gordon (2017)] or Bayesian Optimization [Snoek et al. (2012), Kandasamy et al. (2018)]).

Finally one of the most important approaches forgoes the need to optimize the hyper-parameter search space analysis and uses gradient descent optimization (Brock et al. (2017), Liu et al. (2018)). A particular class of such methods allow the model to learn self-gating mechanisms that might shortcut unwanted layers and allow the information to flow through optimization-objective dictated pathways. These approaches lowers the computational burden of both heuristic grid-search based methods as well as learnable or genetic architectural search. Such approaches start from initial paper by Srivastava et al. (2015) and continue with the important research of Hu et al. (2019).

In section 2, we will concentrate on presenting the best current state-of-the-art directions used for designing neural graphs architectures in order to have a well defined starting point for our contrast analysis. The final two sections of this paper clearly state our position by contrasting the reviewed methods with our proposed current and future work. The main contrasting related work is based on the following sources:

1. Reinforcement Learning methods
   (a) Neural Architecture Search with Reinforcement Learning (Zoph and Le, 2017)
   (b) Learning Transferable Architectures for Scalable Image Recognition (Zoph et al., 2018)
2. Evolutionary methods
   (a) Regularized Evolution for Image Classifier Architecture Search (Real et al., 2019)
3. Architecture search through optimization
   (a) SMASH: One-Shot Model Architecture Search through HyperNetworks (Brock et al., 2017)
   (b) DARTS: Differentiable Architecture Search (Liu et al., 2018)
4. Gating mechanisms
   (a) Highway Networks (Srivastava et al., 2015)
   (b) Squeeze-and-Excitation Networks (Hu et al., 2019)

2 Papers Summaries

2.1 Reinforcement Learning methods

2.1.1 Neural Architecture Search with Reinforcement Learning (*NAS*)

The area of Neural Architecture Search - or NAS in short - is a very active research and experimentation area that provided in the last years good results particularly in the area of Deep Computer Vision both for academia and industry applications. Our proposed work is directly related to the general area of deep neural graph architectures search. However, as mentioned before, in this area there are multiple directions of research and within the multitude of approaches a particular sub-direction is dedicated to applying self-learning algorithms based on reinforcement learning for the objective *intelligent* search within the hyper-parameters spaces.

Intuitively, Zoph and Le (2017) propose an agent that simulates a recurrent process of predicting sequential tokens representing layer parameters that, when put together, generate a functional design of a sequential neural network. In a simple example the authors consider the usual case of deep vision models where NAS based on reinforcement learning is applied: at each step of the unrolled recurrent neural network - called *controller* - generates a certain type of convolutional neural network hyper-parameter such as number of filters, filter width, stride width, etc. The authors use as reinforcement reward for the agent controller a accuracy-like metric - such as accuracy, recall, etc - obtained after training the controller-generated candidate. Finally, the reward is used in conjunction with the REINFORCE algorithm in order to optimize the recurrent sequence generator controller model.

One of the inherent issues of the above approach is that each proposed candidate must be restricted to pure sequential operations without branching or even skip or residual connections that any modern graph architecture relies on. This constraint is imposed by the nature of the controller agent that is based on token-after-token and thus layer-after-layer generation. In order to enable the controller agent to generate complex non-sequential graph structures - such as skip connection - the authors add attention based mechanisms that allow for each

timestep-generated layer to be connected to previous generated layers. The intuition is that for each new predicted token (layer) the attention mechanisms can pinpoint to a previous layer that should be added as input to the current layer input and thus create learnable skip (or residual for that matter) connections.

Finally, in terms of efficiency the authors focus mainly on reducing the time-to-optimal-solution introducing multi-agent training with multiple parallel parameter servers.

2.1.2 Learning Transferable Architectures for Scalable Image Recognition (*NASNet*)

The method proposed by Zoph et al. (2018) is focused on using a small proxy dataset (CIFAR-10) to find the best convolutional architecture that is also fit to be successfully applied on a larger dataset (ImageNet). They achieve this transferability by designing a search space (called "NASNet search space") which is independent of the input image size and of the depth of the network. Mainly, they found in the NASNet search space the best convolutional layer ("cell") that is stacked multiple times (resulting in a feed-forward stack of Inception-like modules called "cells"). In order to achieve independence, they design 2 types of convolutional cells (*Normal Cells* - return feature maps of the same dimension and *Reduction Cells* - return feature maps whose dimensions (height, width) are reduced by a factor of 2). They use NAS methodology to search the structure of these 2 convolutional cell types in the NASNet search space. The search space has some restrictions: each cell receives two inputs (namely the outputs of two cells in previous two lower layers or the input image - i.e. a direct input and a skip input), the operations applied on these 2 inputs are sampled using the softmax output of the NAS controller RNN from a standard set of 10 operations that are widely used in the CNN literature (e.g. identity, 3x3 average pooling, 5x5 max pooling, 1x1 convolution etc), the combination of the 2 results after applying an operation are combined either using element-wise addition or concatenation along the filter dimension. Once finalized the search process, the result will be a single "Normal Cell" architecture and a single "Reduction Cell" architecture that will be used multiple times in the entire convolutional architecture.

They also drop the NAS methodology and successfully use random search to sample the operations from a uniform distribution. This aspect demonstrates that NASNet search space is well constructed and even random grid search also performs reasonably well.

The main result of this work is that the best convolutional architecture found on CIFAR-10 generates the NASNet-A neural network that achieves state-of-the-art accuracy (82.7% top-1 accuracy) on ImageNet without much modification. However, the search process lasts over 4 days using a pool of workers consisting of 500 GPUs.

A by-product of their work is the discovery of *ScheduledDropPath*, a modified version of DropPath (Larsson et al., 2017). Basically, both methods drop (stochastically or not) each path in the convolutional cell.

2.2 Evolutionary methods

2.2.1 Regularized Evolution for Image Classifier Architecture Search (*AmoebaNet*)

Another work biased towards image classification is that proposed by Real et al. (2019). They argue that their image classifier called *AmoebaNet-A* surpasses human-crafted convolutional topologies for the first time (obtaining a 83.9% top-1 ImageNet accuracy). Their work is 100% dependent on the *NASNet search space* (Zoph et al., 2018) ("All experiments use the *NASNet search space*"). However, they replace the RL setting for finding the best "cell" with an evolutionary algorithm that modifies the tournament selection evolutionary algorithm by introducing an age property to favor the younger genotypes. This evolutionary setting, compared to the RL one, allows to obtain faster results with the same hardware.

2.3 Architecture search through optimization

2.3.1 SMASH: One-Shot Model Architecture Search through HyperNetworks

In *SMASH* (Brock et al., 2017), the focus is to accelerate the architecture selection process by learning an auxiliary HyperNet (Ha et al., 2017) that generates the weights of a main model conditioned on the model's architecture. They propose a variant of Dynamic HyperNet in which the parameters W of the main model are generated based on the encoding of the main model's architecture ($W = H(c)$). The main outcome is that multiple architectures can be tested in terms of validation loss / accuracy at the cost of a single HyperNet training. Their validation

is done only through HyperNets for Convolutional Neural Networks. The method achieves competitive but no state-of-the-art results on datasets such as CIFAR-10, CIFAR-100, STL-10, ModelNet10 and ImageNet32x32.

2.3.2 DARTS: Differentiable Architecture Search

This work (Liu et al., 2018) addresses the drawbacks of the best existing architecture search algorithms (either RL-based - Zoph and Le (2017) and Zoph et al. (2018), or evolutionary - Real et al. (2019)), which are very computationally demanding, because they search over a discrete set of candidates. To encounter this problem, *DARTS* relaxes the search space to be continuous, so that the resulting model can be directly optimized via gradient descent. *DARTS* is not restricted to a particular architecture family, being applicable both to convolutional and recurrent networks.

In the same manner as in Zoph et al. (2018) and Real et al. (2019), *DARTS* searches for a computational "cell" (either convolutional or recurrent) that will be multiplied (stacked or recursively connected) in order to form the final network. DARTS uses most of the intuition behind the *NASNet search space*, designing the inputs in the computational "cell" to be the outputs of two cells in previous two lower layers or the input. However, they make the possible operations search space continuous, relaxing the categorical choice of a particular operation to a softmax over all possible operations. The final goal is to jointly learn the architecture pattern and the weights within the resulting network. Therefore, they employ a bilevel optimization algorithm where the best architecture pattern is found by minimizing the validation loss and the weights within the resulting network are obtained by minimizing the train loss. Finally, their method achieves competitive results on CIFAR-10 and outperforms the state-of-the-art on Penn Treebank, having the big advantage of being optimizable.

2.4 Gating mechanisms

2.4.1 Highway Networks

One of the most influencing papers for our work, Srivastava et al. (2015), proposes a straight-forward layer-wise self-gating mechanism for deep neural networks that intuitively allows the graph to self-prune modules that are obsolete. The research team directly references the 1995 Long Short Term Memory recurrent neural network module inception paper (Hochreiter and Schmidhuber, 1997) as an inspiration for the proposed self-gating mechanism, Schmidhuber being one of the former paper authors. The researchers argue that the proposed approach can alleviate the forward as well as backward information flow attenuation in very deep neural networks as well as construct information flow critical paths namely information highways.

The main intuition of the proposed *Highway Network* deep neural graph module can be analyzed starting from a simple case where we have a single linear layer followed by an activation function of our choosing. At this point we add a duplicate linear layer parallel to the initial one as well as a squash activation function such as sigmoid. This additional layer basically uses the input feature information in order to compute the gate values for each individual unit of the initial layer. Finally using the below equation we use the self-gating mechanism to "allow" initial layer information to pass forward (as well as backward) or just bypass it all together.

$$g_{\theta_G}(x) = \frac{1}{1 + e^{\theta_G^T x}} \tag{1}$$

$$h_{\theta_L}(x) = f(\theta_L^T x) \tag{2}$$

$$h_H(x) = g_{\theta_G}(x) * h_{\theta_L}(x) + (1 - g_{\theta_G}(x)) * x \tag{3}$$

In the above equations, x denotes the input features of the layer, θ_G and θ_L are the weight matrices of the self-gating and initial processing, $f(z)$ denotes the activation function of the initial simple layer and finally $h_H(x)$ denotes the operation proposed by Srivastava et al. (2015).

2.4.2 Squeeze-and-Excitation Networks

The area of deep computer vision based on convolutional neural networks has seen a tremendous advancement in the last period. One of the drivers of this advancement has been that of employing advanced auto-ML approaches as well as Network Architecture Search self-learning. One of the most prominent researches in recent years that relates to the area of network architecture designing is the work of Hu et al. (2019) that approaches deep convolutional graphs topology from a similar perspective as Srivastava et al. (2015) with their highway-network architecture.

The main intuition behind *squeeze-and-excite* resides in the idea that a Bernoulli variable could

be computed and applied to a whole channel as a weighting mechanism. Basically, a small fully connected sequential network learns to compute a channel-wise Bernoulli gate, then this vector-gate is applied to the volume (channels) resulting from a convolutional operation - including its linear feature normalization and unit activation. The authors argue that the condensed 1D representation of the 3D feature-maps volume can be seen as an embedding of the whole volume or as an importance-weighting vector of each channel. While in the *highway networks* paper the authors propose a method that learns the weights of gate-layers which in turn are used to choose between information pass-through and processed-information, in this case the *squeeze-and-excite* mechanism acts as a direct and learnable information filter. The sigmoid operation applied to the "squeeze" embedding vector can lead to actually zero-ing entire channels similar to spatial dropout or can "allow" various degrees of information passing through based on the "excite" operation.

3 Compare & Contrast

The work of Zoph and Le (2017) (presented in 2.1.1) is arguably one of the most influential in the area of Neural Architecture Search and a multitude of derived incremental approaches have been developed in past years. Nevertheless, the proposed approach of using reinforcement learning agents as deep neural graph designers does not drastically reduce the computational burden and the architecture search carbon footprint. Although with our proposed approach we do not aim to self-learn and bypass the need for identifying hyper-parameters such as learning rates, weight initialization strategies and such, we argue that focusing on self-learning network topologies - consisting in using *MultiGatedUnits* in conjunction with *graph pruning* post-processing - can drastically reduce the computational costs and the underlying carbon footprint.

As already specified, using reinforcement learning agents as deep neural graph designers does not drastically reduce the computational burden and the architecture search carbon footprint. Zoph et al. (2018) designed the "NASNet search space" (presented in 2.1.2) such that searching (using reinforcement learning) for a convolutional network for image classification reduces to searching in a set of manual predefined operations that are widely used in the CNN literature. In other words, *NASNet* can be understood as a method of narrowing the search space for convolutional "cells". This is a main difference between *MultiGatedUnit* and *NASNet*, because we do not restrict the applicability to one type of computational element. In *NASNet*, once found the best convolutional "cell", it is stacked multiple times in an Inception-like architecture. Considering the possibility that the optimal hypothesis might need unique structures for each individual module throughout the whole deep neural graph, this ideal hypothesis will never be found. In contrast, the proposed *MultiGatedUnit* allows each computational element in the neural graph to have a unique design. *MultiGatedUnit* also has a set of operations that can be applied on a computational element (e.g. normalization after the activations or before [Ioffe and Szegedy (2015), Ba et al. (2016)], adding or not additional skip/residual connections [He et al. (2016)], dropout [Srivastava et al. (2014)] etc), but the flow between the operations is completely optimizable (together with the weights of the resulting computational "modules") using gradient descent. In other words, *MultiGatedUnit* does not sample operations from a discrete set, but it allows one-shot to find (via optimization) the best combination of all possible operations which results in. It is worth mentioning that *NASNet* does not resolve the problem of "arranging" the computational "cells" (or computational "modules" in our case) in the main graph. The search for the depth of the graph is still a necessity, whereas *MGU* allows to select the information flow that maximizes the objective.

Switching to evolutionary methods for designing neural architectures, Real et al. (2019) proposed *AmoebaNet-A* (presented in 2.2.1) which is nothing else but a neural network resulted after searching in the "NASNet search space" using an evolutionary algorithm. Thus, they propose a new searching algorithm through the *NASNet* discrete space that is more computational efficient than reinforcement learning. Therefore, there is no other similarity or difference between the proposed *MultiGatedUnit* and their search methodology that created *AmoebaNet-A*.

There is another class of methods that use optimization for designing architectures, which are more similar to our work from this perspective. The work of Brock et al. (2017) (presented in 2.3.1) presents a *Dynamic HyperNet* that is able to gener-

ate the main network's weights, given their architecture. Once generated multiple networks, they can be compared in terms of validation loss / accuracy. However, the architectures are still manually designed, unlike *MultiGatedUnit* which determines automatically all the feature processors that will be applied to a computational element. The main drawback of their approach is the need of enormous architectural "sketches" which must be thought of by humans. But in the end, one single *HyperNet* is required to train, as in *MGU*. Even though they present their methodology only with convolutional neural networks, as *MultiGatedUnit* it can be scaled to other types of networks. Another work in this area is that of Liu et al. (2018) (presented in 2.3.2) which uses the "NASNet search space" for convolutional neural networks and adapts it for recurrent neural networks. To encounter the problem of searching over a discrete set of candidates, *DARTS* relaxes the search space to be continuous, so that the resulting model can be directly optimized via gradient descent. However, using the "NASNet search space" imposes some restrictions (repeated computational "cell" throughout the network, depth of the graph) which were already debated.

While the work of Srivastava et al. (2015) (presented in 2.4.1) can be considered as the main influential idea behind the inception of our paper we went several several steps forward, some of which have also been hinted in the paper - learning to route information through deep neural networks and the self-learning ability of *highway networks* for bypass almost entirely some layers. Based on their experiments the authors conclude that *highway networks* utilize the self-gating mechanisms to pass information almost unchanged through many of the layers. Finally, our work aims to further develop more complex approaches both to information routing in deep neural graphs with the purpose of enabling self-learning architectures as well as generating automated and advanced heuristics to analyse the actual information flows and regenerate pruned graphs based on inferred critical paths.

The *squeeze-and-excite* technique (presented in 2.4.2) along with *highway networks* propose similar yet different methods for self-learning graph topologies. From the perspective of our work the most important take-away from the *squeeze-end-excite* approach is that of computing compressed representation of the information rather than simply mirroring the features as gating units. Nevertheless, the main issue here is that of generalization: while for self-gating mechanisms one has only to mirror the linear computation followed by a squash activation function such as sigmoid when computing actual embeddings it all comes down to the way information is represented. For convolutional layers a channel-wise gating mechanism based on an embedding representation of each individual channel makes a lot of sense, however for other types of layers this procedure is unclear. In comparison to *squeeze-end-excite* layers, the *MultiGatedUnit* can be applied to almost any type of layer. Nevertheless, the idea of compressing the gating information and forcing the gating layer to learn lower dimensional representations of either the input information or the in-layer processed output is part of our research pipeline.

4 Future Work

In terms of current ongoing research we experimented on image classification tasks with classic datasets such as MNIST and we are currently experimenting with CIFAR-10 and more importantly with the CERVIGRAM (Xu et al., 2015) dataset, as well as employ the *MGU* in predictive analytics experiments on private business data. An important note is that our source code implementation will be made publicly available on GitHub to facilitate further research and development.

Furthermore, we will also explore through research the idea of prune-convert-optimize the *MGUs* following the end-to-end training in order to reduce the inference time as well as energy consumption required for the numerical graph computations. The challenge is to find a heuristic approach of determining the gating thresholds that allow the decision of eliminating gating mechanisms such as for the obvious case when a gate is always opened or closed no matter the input. This heuristic will allow us to "copy" the structure and weights from the *MGU* module to the new "pruned" modules and thus eliminate the need for the multiple self-gating operations.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization.

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. 2017. Designing neural network architectures using reinforcement learning. In *5th Inter-*

national Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings.

James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*.

Andrew Brock, Theodore Lim, J. M. Ritchie, and Nick Weston. 2017. Smash: One-shot model architecture search through hypernetworks.

Dario Floreano, Peter Dürr, and Claudio Mattiussi. 2008. Neuroevolution: From architectures to learning.

David Ha, Andrew M. Dai, and Quoc V. Le. 2017. Hyper networks. In *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation*.

Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. 2019. Squeeze-and-excitation networks.

Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *32nd International Conference on Machine Learning, ICML 2015*.

Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabás Póczos, and Eric P. Xing. 2018. Neural architecture search with Bayesian optimisation and optimal transport. In *Advances in Neural Information Processing Systems*.

Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. 2017. Fractalnet: Ultra-deep neural networks without residuals.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search.

S.U. Miller, G.F.; Todd, P.M.; Hegde. 1989. Designing Neural Networks using Genetic Algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms, George Mason University, Fairfax, Virginia, USA*.

Renato Negrinho and Geoff Gordon. 2017. Deeparchitect: Automatically designing and training deep architectures.

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. 2018. Efficient Neural Architecture Search via parameter Sharing. In *35th International Conference on Machine Learning, ICML 2018*.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. 2019. Regularized evolution for image classifier architecture search.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Highway networks. ArXiv:1505.00387.

Kenneth O. Stanley and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evolutionary Computation*.

Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. 2018. A genetic programming approach to designing convolutional neural network architectures. In *IJCAI International Joint Conference on Artificial Intelligence*.

Daan Wierstra, Faustino J. Gomez, and Jürgen Schmidhuber. 2005. Modeling systems with internal state using evolino. In *GECCO 2005 - Genetic and Evolutionary Computation Conference*.

Tao Xu, Cheng Xin, L Rodney Long, Sameer Antani, Zhiyun Xue, Edward Kim, and Xiaolei Huang. 2015. A new image data set and benchmark for cervical dysplasia classification evaluation. In *International Workshop on Machine Learning in Medical Imaging*, pages 26–35. Springer.

Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng Lin Liu. 2018. Practical Block-Wise Neural Network Architecture Generation. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

Barret Zoph and Quoc V. Le. 2017. Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. 2018. Learning Transferable Architectures for Scalable Image Recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.