A view on automated neural graph topology generation and a viable direction of innovation

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

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

Nicolae Tapus
University Politehnica of Bucharest
Bucharest, Romania
ntapus@cs.pub.ro

***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, currently in research and experimentation phase.**

***Index Terms*—neural network architecture search, deep neural graph topology design**

I. 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 [15], [22], [24], [7]. 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 [18] from Google Brain, but there exist other modern evolutionary research [23].

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 [4], [26], [27], [17], [25].

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 [5], MCTS [16] or Bayesian Optimization [19], [13]).

Finally one of the most important approaches forgoes the need to optimize the hyper-parameter search space analysis and uses gradient descent optimization [6], [14]. 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 the *Highway networks* research [21] and continue with the important *Squeeze-and-Excitation Networks* [11].

In section II, 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 sections of this paper clearly state our position by contrasting the reviewed methods with our proposed current and future work.

II. State-of-the-art directions in automated graph topology architecture

A. Reinforcement Learning methods

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, [26] 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) Learning Transferable Architectures for Scalable Image Recognition (NASNet): The method proposed by [27] 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 [2]. Basically, both methods drop (stochastically or not) each path in the convolutional cell.

B. Evolutionary methods

1) Regularized Evolution for Image Classifier Architecture Search (AmoebaNet): Another work biased towards image classification is that proposed by [18]. 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* [27] ("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.

C. Architecture search through optimization

1) SMASH: One-Shot Model Architecture Search through HyperNetworks: In *SMASH* [6], the focus is to accelerate the architecture selection process by learning an auxiliary HyperNet [8] 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) DARTS: Differentiable Architecture Search: This work [14] addresses the drawbacks of the best existing architecture search algorithms (either RL-based - [26] and [27], or evolutionary - [18]), 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 [27] and [18], *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.

D. Gating mechanisms

1) Highway Networks: One of the most influencing papers for our work, [21], 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 [10] 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 equation 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 [21].

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 [11] that approaches deep convolutional graphs topology from a similar perspective as [21] 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.

III. Compare & Contrast

The work of [26] (presented in II-A1) 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. [27] designed the "NASNet search space" (presented in II-A2) 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 [12], [1], adding or not additional skip/residual connections [9], dropout [20] 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 the grid search space. 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, [18] proposed *AmoebaNet-A* (presented in II-B1) 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 [6] (presented in II-C1) presents a *Dynamic HyperNet* that is able to generate 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 [14] (presented in II-C2) 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 [21] (presented in II-D1) 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 II-D2) 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 in-

dividual 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.

IV. Overview of MultiGatedUnit proposal

The proposed architecture of multi gated sub-graph unit - or *MGU* in short - encapsulates any kind of neural computational layer (linear, convolutional, etc.) and it is able to self-learn the optimal sub-graph configuration. Intuitively, the sub-graph containing the encapsulated layer, as well as various feature processors, is automatically configured through self-gating mechanisms. Using just one end-to-end optimization process based on a task-oriented objective function, the deep neural graph using *MGUs* will learn a unique individual module structure for all modules within the graph.

Half-automated architectural design approaches have two major drawbacks. The first issue is that the search space grows exponentially with the number of feature processors used within a module as well as with the number of options for each processor. 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 the 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. The proposed *MultiGatedUnit* efficiently addresses these major issues.

A. Feature processors and data flow

The main building block of *MGU* is the gate-activation block that is composed of a linear transformation and a squash function - *sigmoid* - that will constrain the gate-activation block output values between 0 and 1. Intuitively, the gate-activation output acts like a *pass* or *not pass* signal for a feature processor of the *MGU*'s input . In other words, the linear transformation weights encapsulates whether a feature processor is a good one or not to be *employed* in the computational module.

The proposed work is focused towards automatic generation of graph topologies using the deep neural network feature processors that are currently most used both in convolutional and feed-forward neural networks literature such as *Batch Normalization* [12], *Layer Normalization* [1] and residual connections [9]. Each gate-activation block inputs two feature processors and computes the *pass* / *no pass* signal given the *MGU*'s input. Thus, it has the purpose of choosing between one a certain feature processing method or pathways within the graph.

In the following equations we will present formalization of the *MultiGatedUnit* information pipeline. We will denote $L_x = f_{lyr}(x)$, where f_{lyr} is the *MGU* encapsulated transformation and x is the input of the *MGU*. Also, the encapsulated activation is denoted by $A(z)$ and a linear transofrmation of the input is denoted by $t(x)$.

The first gate is responsible for choosing whether to use pre-activation or post-activation *Batch Normalization*.

$$l_{bnpp}(x) = W_{bnpp}x + b_{bnpp} \tag{4}$$

$$g_{bnpp}(x) = \sigma(l_{bnpp}(x)) \tag{5}$$

$$f_{bn1}(x) = \mathrm{BN}(\mathrm{A}(L_x)) \tag{6}$$

$$f_{bn2}(x) = \mathrm{A}(\mathrm{BN}(L_x)) \tag{7}$$

$$f_{bnpp}(x) = g_{bnpp}(x) * f_{bn1}(x) + (1 - g_{bnpp}(x)) * f_{bn2}(x) \tag{8}$$

The second gate chooses the best pathway between *Batch Normalization* or *Layer Normalization*.

$$l_{bnln}(x) = W_{bnln}x + b_{bnln} \tag{9}$$

$$g_{bnln}(x) = \sigma(l_{bnln}(x)) \tag{10}$$

$$f_{ln}(x) = \mathrm{A}(\mathrm{LN}(L_x)) \tag{11}$$

$$f_{bnln}(x) = g_{bnln}(x) * f_{bnpp}(x) + (1 - g_{bnln}(x)) * f_{ln}(x) \tag{12}$$

The third gate decides whether it is better to use any kind of feature normalization or not.

$$l_{non}(x) = W_{non}x + b_{non} \tag{13}$$

$$g_{non}(x) = \sigma(l_{non}(x)) \tag{14}$$

$$f_{non}(x) = g_{non}(x) * f_{bnln}(x) + (1 - g_{non}(x)) * \mathrm{A}(L_x) \tag{15}$$

The fourth gate is able to choose whether to use or not residual connections.

$$l_{ron}(x) = W_{ron}x + b_{ron} \tag{16}$$

$$g_{ron}(x) = \sigma(l_{ron}(x)) \tag{17}$$

$$f_{res}(x) = f_{non}(x) + t(x) \tag{18}$$

$$f_{ron}(x) = g_{ron}(x) * f_{non}(x) + (1 - g_{ron}(x)) * f_{res}(x) \tag{19}$$

Finally, the last gate can decide to skip all the feature processors and use the transformed input. The output of the last gate represents also the output of the *MultiGatedUnit*

$$l_{skip}(x) = W_{skip}x + b_{skip} \tag{20}$$

$$g_{skip}(x) = \sigma(l_{skip}(x)) \tag{21}$$

$$o(x) = g_{skip}(x) * f_{ron}(x) + (1 - g_{skip}(x)) * t(x) \tag{22}$$

V. Conclusion and further 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 [3] dataset, as well as employ the *MGU* in predictive analytics experiments on private business data. 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.

The quest for low-carbon-print neural model training and efficient architecture/topology search is an active area of research. At this point extensive experimentation is further required to show strong evidence that the added value of the proposed automated layer is consistent and generalizable. As a result, we decided for our experimentation to augment the classic hypothesis testing approaches based on often-used metrics such as accuracy, recall, precision with two other metrics that have the main purpose of measuring and comparing the carbon footprint of the *MGU* based topology search and optimization process with the one based on grid-search: time-to-solution and GPU load. The *time-to-solution* metric purpose is to measure the time it takes to obtain the best-model both with a grid-search approach as well as based on the optimization process of the *MGU* based deep neural graph. In order to have a realistic comparison of the *MGU* approach with that of a hyper-parameter space search we opted to analyze the potential search-space and limit it to the lowest number of dimensions possible. As a result, our hyper-parameters search algorithm performs a minimal number of iterations within the grid-search process. Finally we compare this total time with the time it takes to train, evaluate and *self-analyze* the *MGU* deep neural graph. Actual energy consumption is directly correlated with the time-to-solution as well as with the average loading of the GPU device needed for the optimization of the proposed models.

Exploration of *MGU* self explain-ability and self-pruning potential, experimentation in multiple domains as well as comparison with grid-search or NAS approaches, *open-source* publication of MGU *python* libraries for *Tensorflow* and/or *PyTorch* are all currently underway and, in our own view, will soon generate important results.

References

[1] Lei Ba, J., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. In arXiv. *arXiv:1607.06450*

[2] Larsson, G., Maire, M., & Shakhnarovich G. (2016). FractalNet: Ultra-Deep Neural Networks without Residuals. In arXiv. *arXiv:1605.07648*

[3] Xu, T., Xin, C., Long, L. R., Antani, S., Xue, Z., Kim, E., & Huang, X. (2015). A new image data set and benchmark for cervical dysplasia classification evaluation. International Workshop on Machine Learning in Medical Imaging pages 25-36. Springer.

[4] Baker, B., Gupta, O., Naik, N., & Raskar, R. (2017). Designing neural network architectures using reinforcement learning. 5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings.

[5] Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. Journal of Machine Learning Research.

[6] Brock, A., Lim, T., Ritchie, J. M., & Weston, N. (2017). SMASH: One-shot model architecture search through hypernetworks. In arXiv. *arXiv:1708.05344*

[7] Floreano, D., Dürr, P., & Mattiussi, C. (2008). Neuroevolution: From architectures to learning. In Evolutionary Intelligence. https://doi.org/10.1007/s12065-007-0002-4

[8] Ha, D., Dai, A. M., & Le, Q. V. (2017). Hyper networks. 5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings.

[9] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. https://doi.org/10.1109/CVPR.2016.90

[10] Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation. https://doi.org/10.1162/neco.1997.9.8.1735

[11] Hu, J., Shen, L., Albanie, S., Sun, G., & Wu, E. (2020). Squeeze-and-Excitation Networks. IEEE Transactions on Pattern Analysis and Machine Intelligence. https://doi.org/10.1109/TPAMI.2019.2913372

[12] Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. 32nd International Conference on Machine Learning, ICML 2015.

[13] Kandasamy, K., Neiswanger, W., Schneider, J., Póczos, B., & Xing, E. P. (2018). Neural architecture search with Bayesian optimisation and optimal transport. Advances in Neural Information Processing Systems.

[14] Liu, H., Simonyan, K., & Yang, Y. (2018). Darts: Differentiable architecture search. In arXiv. *arXiv:1806.09055*

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

[16] Negrinho, R., & Gordon, G. (2017). Deeparchitect: Automatically designing and training deep architectures. In arXiv. *arXiv:1704.08792*

[17] Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., & Dean, J. (2018). Efficient Neural Architecture Search via parameter Sharing. 35th International Conference on Machine Learning, ICML 2018.

[18] Real, E., Aggarwal, A., Huang, Y., & Le, Q. V. (2019). Regularized evolution for image classifier architecture search. 33rd AAAI Conference on Artificial Intelligence, AAAI 2019, 31st Innovative Applications of Artificial Intelligence Conference, IAAI 2019 and the 9th AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019. https://doi.org/10.1609/aaai.v33i01.33014780

[19] Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical Bayesian optimization of machine learning algorithms. Advances in Neural Information Processing Systems.

[20] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research.

[21] Srivastava, R. K., Greff, K., & Schmidhuber, J. (2015). Training very deep networks. Advances in Neural Information Processing Systems.

[22] Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. Evolutionary Computation. https://doi.org/10.1162/106365602320169811

[23] Suganuma, M., Shirakawa, S., & Nagao, T. (2018). A genetic programming approach to designing convolutional neural network architectures. IJCAI International Joint Conference on Artificial Intelligence. https://doi.org/10.24963/ijcai.2018/755

[24] Wierstra, D., Gomez, F. J., & Schmidhuber, J. (2005). Modeling systems with internal state using evolino. GECCO 2005 - Genetic and Evolutionary Computation Conference. https://doi.org/10.1145/1068009.1068315

[25] Zhong, Z., Yan, J., Wu, W., Shao, J., & Liu, C. L. (2018). Practical Block-Wise Neural Network Architecture Generation. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. https://doi.org/10.1109/CVPR.2018.00257

[26] Zoph, B., & Le, Q. V. (2017). Neural architecture search with reinforcement learning. 5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings.

[27] Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2018). Learning Transferable Architectures for Scalable Image Recognition. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. https://doi.org/10.1109/CVPR.2018.00907