American University Of Armenia
College of Science and Engineering

Capstone Project

Finding Network: Neural Architecture Search

Grigor Nalbandyan

Supervisor: Arsen Yeghiazaryan

Abstract

Neural networks are widely used in computer vision and natural language processing. Despite their advancements during the last 10 years, manually designing a new neural network is still a challenge and mainly limited to big research labs with field expert scientists and huge amounts of computational power. Furthermore, the process of creating new architectures is slow and can be very hard given the constraints such as number of parameters or inference time. Such problems have given rise to the need for *neural architecture search or NAS,* which aims to automate the design of neural network architectures.

The aims of the capstone project are

- Extensive research of the current state of NAS, main approaches, results compared to human-made networks, advantages, and drawbacks
- Understand one of the NAS algorithms from an engineering perspective due to its difference from the conventional neural network training.
- Gain research experience by questioning and studying each part of the algorithm and experiment aiming to improve its results.

Source code of the paper can be found at https://github.com/GRIGORR/capstone

1. Introduction

The advancements of neural networks make the search for new and better architectures harder, computationally expensive, and requiring significant human effort. To tackle those problems, neural architecture search (NAS) has become a hot topic of research during the last 4 years. After comprehensive research on NAS literature, various aspects, and main approaches of NAS, comparing the necessary computational resources for each method, we chose gradient-based method NAS as a future research area. Gradient-based search using Differentiable Architecture Sample or GDAS [1] is the fastest gradient-based NAS algorithm to the best of our knowledge producing high results for the moment of its publication. It can be trained on a single GPU for several hours. As the aim of the project was the research of one of the NAS algorithms, GDAS was ideal as it enabled us to experiment having limited computational power.

GDAS searches normal and reduction cells and then stacks them to get final architecture. To find the normal and reduction cells, it treats each cell as a directed acyclic graph (DAG) and samples a cell each time. It allows the search to be very fast as at each step only part of the parameters is learned.

2. Pillars of NAS: Space, Scope and Main Strategies

To search a model by NAS one should state three aspects of the search procedure: search space, the scope of the search, and search strategy.

Search space is the collection of the operations that NAS should combine and find the desired architecture. It can consist of already defined modules (e.g., 3x3 convolution with batch normalization), parameters of such modules (filter height, filter width, stride, etc.) or anything else. Each NAS approach requires its own way of defining the search space. In general, search space is constructed using the historical experience of the field (e.g. use conventional depthwise or dilated convolutions), which can result in small search space and less computation, but may also induce human bias and can block NAS to find new ways of solving the problem.

The scope of the search can be macro or micro. During macro search strategy, there is no constraint on what kind of architecture search algorithm should produce. Micro search strategy fixes the skeleton of the architecture and searches particular cells. Figure 1 is an example of a micro search where the structure of the architecture is fixed, normal and reduction cells should be searched.

We have separated three main families of search strategies for NAS: reinforcement learning (RL), evolutionary search, and gradient-based methods.

RL was the first big step towards NAS in [2]. They used LSTM as an agent to sample a multilayer convolutional neural network. The agent sampled convolutional layer’s parameters - filter height, filter width, stride height, stride width, and the number of filters for each layer and then all layers were concatenated to get the architecture and trained. The accuracy of the sampled model was used as a reward to train the agent (LSTM). Their approach resulted in comparable results with human-made models of that time both in CV and NLP and proved that NAS indeed has the potential to open new possibilities in architecture development. On the other side, they had to train each sampled model independently, which resulted in the usage of huge amounts of computational power - 800 GPUs for 28 days. [3](NASNet) used RL with micro search to search for normal and reduction cells and stacked multiple cells together to get the final architecture (macro architecture in FIgure 1). This approach of normal and reduction cell search became very popular in the NAS community. Their search space consisted of already defined convolutional modules, which resulted in much smaller search space than the one used in previous work. NASNet was trained for 4 days on 500 GPUs which is less computational power than previous work but still huge. [4] used RL with micro search and parameter sharing to reduce the training to 1 GPU and half a day and got even higher results. [5] used RL to search for networks to use in constrained environments. Their aim was not to just find an accurate model, but also a model with low latency to be used in mobile phones. Some of the works mentioned above also used NAS to search recurrent cells as a replacement for LSTM and again got either comparable or higher results on language modeling tasks.

Evolutionary search is based on the idea of evolution and genetic algorithms. Main approach of evolutionary search is

1) initialize a pool of random models
2) at each step mutate some of the models from the pool and train them
3) put all new models with higher accuracy into the pool
4) remove the oldest or worst model from the pool
5) repeat until steps 2-4 for a specified amount of time.

This way, only the best models stay in the pool after the search procedure ends. [6] used an evolutionary search to get comparable results with RL based search models and showed that RL is not the only way of NAS. [7] used evolutionary search to find a new transformer language model. Still, as at each step, multiple networks are trained independently, and a substantial amount of computational power was used - 450 GPUs for 7 days in case of [6].

Gradient-based methods transform NAS into conventional deep learning model training by gradient descent. [8] was the first to propose continuous relaxation and use gradient descent based training for NAS. They used micro search, searched for normal and reduction cells. The advantage of gradient-based methods is that they can significantly decrease the required amount of computational resources and can be trained on a single GPU. The idea and methodology of gradient based methods is presented in the next section.

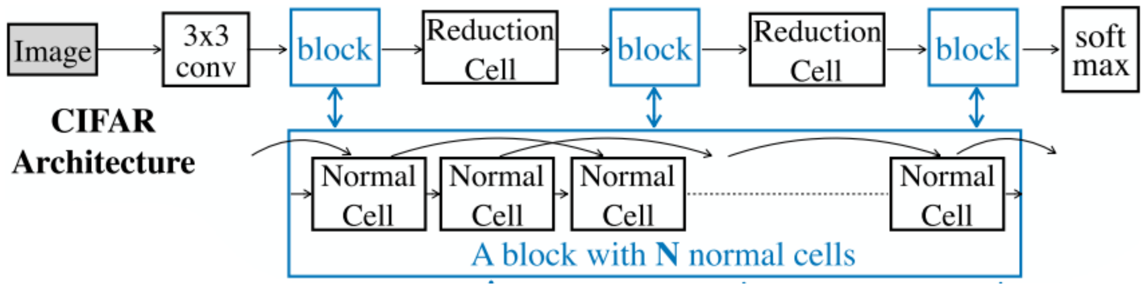

Figure 1. Macro architecture of GDAS for CIFAR-10. Reduction and Normal cells are searched, then stacked to form the final architecture. Each cell receives as input the outputs of two previous cells. (Source, GDAS paper)

3. GDAS - gradient based NAS

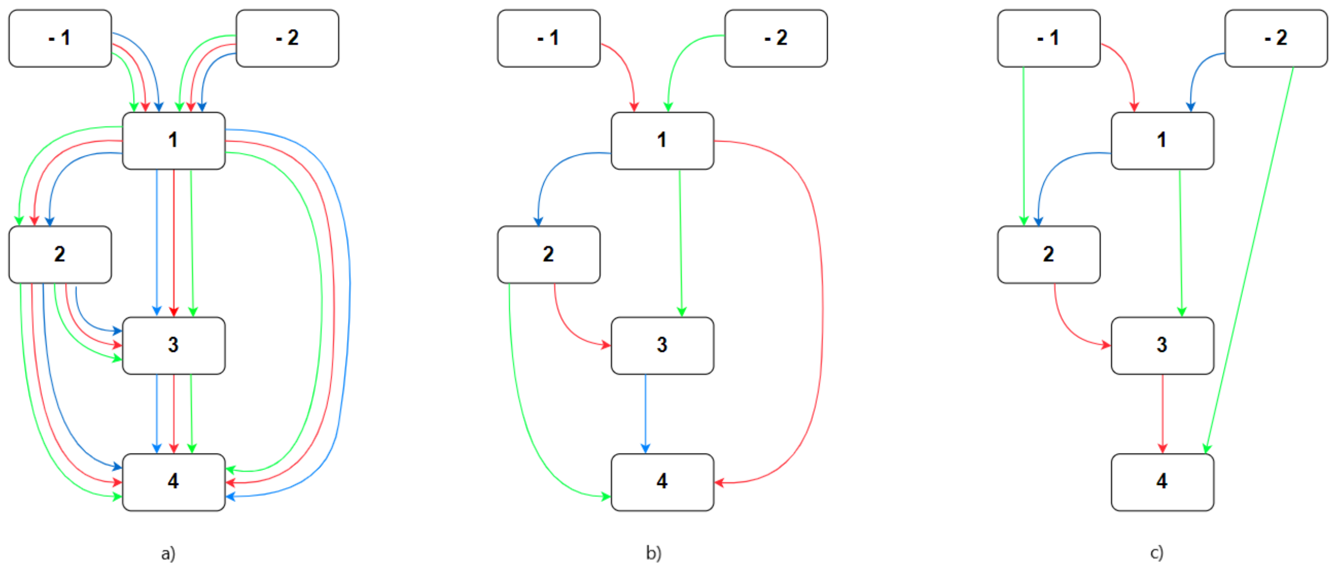

Figure 2. GDAS cell structure

a) Initially, all nodes are connected to their previous nodes by functions from search space **F**. Each arrow is a function form search space. Nodes -1 and -2 are outputs of the previous 2 cells. Connections from nodes -1, -2 to nodes 2,3,4 are not shown to make the illustration simple but there are similar 3 arrow connections between -1,-2 and nodes 2,3,4.

b) To sample a cell, one function(one arrow) is left to connect each two nodes.

c) After search, to derive a cell, each node should be connected to T previous nodes (T=2 in GDAS paper). Connections with top-T logits or probabilities are kept.

GDAS does micro search to search for normal and reduction cells. After the search, it stacks the found cells to form the final architecture. Figure 1 shows the macro architecture of GDAS. It starts by 3x3 convolution, followed by 3 blocks of stacked N normal cells and reduction cells between blocks. Both normal and reduction cells consist of **B** computational nodes. Input of the cell is the output of 2 previous cells. Search space **F** has 8 functions: (1) identity, (2) zeroize, (3) 3x3 depth-wise separate conv, (4) 3x3 dilated depth-wise separate conv, (5) 5x5 depth-wise separate conv, (6) 5x5 dilated depth-wise separate conv, (7) 3x3 average pooling, (8) 3x3 max pooling. In a normal cell, each function has stride 1 whereas in the reduction cell each function has stride 2. Lets denote cardinality of **F** as K.

Every node in a cell is connected to each of its previous nodes by K connections where each connection is a function from search space **F** (Figure 2). Output of the cell is depthwise concatenation of **B** nodes. Lets denote a discrete probability distribution over connections between node *i* and node *j* as $\mathbf{G}_{i,j}$, which is defined by a K dimensional learnable vector $\mathbf{A}_{i,j}$. Each element of $\mathbf{A}_{i,j}$ encodes the unnormalized log probability or logits of connections between node *i* and node *j* by one of the functions from **F**. We can get the probability of connection of two nodes by F_k - one of the functions from **F** by

$$\Pr(f_{i,j} = \mathbb{F}_k) = \frac{\exp(\boldsymbol{A}_{i,j}^k)}{\sum_{k'=1}^{K} \exp(\boldsymbol{A}_{i,j}^{k'})} \tag{1}$$

Each element of $\mathbf{G}_{i,j}$ can be obtained by (1) and we will have the probability distribution over the connection of each pair of nodes *i* and *j*.

Having the probability distribution over connections between each two nodes, we can connect two nodes by sampling a function from the distribution. Hence a cell can be sampled and then, an architecture. After sampling the architecture, each node will be calculated as

$$\boldsymbol{I}_i = \sum_{j=1}^{i-1} f_{i,j}\ (\boldsymbol{I}_j; \boldsymbol{W}_{f_{i,j}}) \tag{2}$$

where $f_{i,j}$ is the sampled function from search space **F** using the probability distribution $\mathbf{G}_{i,j}$ and $\mathbf{W}_\mathbf{f}$ is the corresponding weight of each sampled $f_{i,j}$. Each node is the sum of the previous nodes connected to it by the respective sampled functions.

Let's denote the set of all $\mathbf{A}_{i,j}$ as A = {$\mathbf{A}_{i,j}$} and the set of all $\mathbf{W}_\mathbf{f}$ as W. The goal of the search procedure is to learn A - probability distribution for connections between nodes and W - the corresponding weights of each function. After an architecture is sampled, we can easily use backpropagation algorithm to update W weights. To sample an architecture and enable back propagation through sampled architecture to update A paper uses Gumbel-Max trick [9] to turn (2) to

$$\begin{aligned} \boldsymbol{I}_i &= \sum_{j=1}^{i-1} \sum_{k=1}^{K} \boldsymbol{h}_{i,j}^k\ \mathbb{F}_k(\boldsymbol{I}_j; \boldsymbol{W}_{i,j}^k), \\ \text{s.t.}\ \ \boldsymbol{h}_{i,j} &= \text{one\_hot}(\arg\max_k(\boldsymbol{A}_{i,j}^k + \boldsymbol{o}_k)), \end{aligned} \tag{3}$$

Where $\mathbf{o}_\mathbf{k}$ are i.i.d samples from Gumbel(0,1) distribution and
$\mathbf{o}_\mathbf{k}$ = - log(- log(*u*)), *u* sampled from Unif[0,1]

To make (3) differentiable, softmax function is used to relax argmax

$$\hat{\boldsymbol{h}}_{i,j}^{k} = \frac{\exp((\log(\Pr(f_{i,j} = \mathbb{F}_k)) + \boldsymbol{o}_k)/\tau)}{\sum_{k'=1}^{K} \exp((\log(\Pr(f_{i,j} = \mathbb{F}_{k'})) + \boldsymbol{o}_{k'})/\tau)} \tag{4}$$

Where $\boldsymbol{\tau}$ is the softmax temperature. If $\tau \rightarrow 0$, ĥ → **h**, and if $\tau \rightarrow \infty$, elements of ĥ will be nearly equal and the distribution will be very smooth.
To perform the search, two disjoint datasets $\mathbb{D}_T$ - train dataset and $\mathbb{D}_V$ - are used. $\mathbb{D}_T$ is used to update W and $\mathbb{D}_V$ to update A. Loss function is negative log likelihood.

$$\ell(x, y) \;\; = -\log \Pr(y|x; \alpha, \omega_\alpha)$$

where x - input, α - sampled architecture, ω - architecture weights from W
Pseudo Algorithm for GDAS therefore becomes
While not converged

1) Sample a batch of data $d_t = \{(x_i, y_i)\}$ from $\mathbb{D}_T$ and sample architecture using (3)
2) Calculate loss and update W by gradient descent
3) Sample a batch of data $d_v = \{(x_i, y_i)\}$ from $\mathbb{D}_V$ and sample architecture (3)
4) Calculate loss and update A by gradient descent using ĥ

Note that while forward pass is done by (3), backward pass is done by (4).

After training is done and A is learned, we should get the final architecture and train the network from scratch. To infer the final architecture, each node should be connected to T previous nodes. To decide which connections to keep for node *i*, connections of node *i* with top-T biggest logits are kept. GDAS paper chose T = 2.

Paper also elaborates on the necessity to search the reduction cell. Authors state that most reduction cells both human designed and automatically discovered are very simple and perform well. Furthermore, searching for both reduction and normal cells have bigger search space and is a harder optimization problem compared to searching just a normal cell. Paper experiments also show better performance if the reduction cell is fixed. Following the paper, we performed experiments with both fixing and searching for reduction cell.

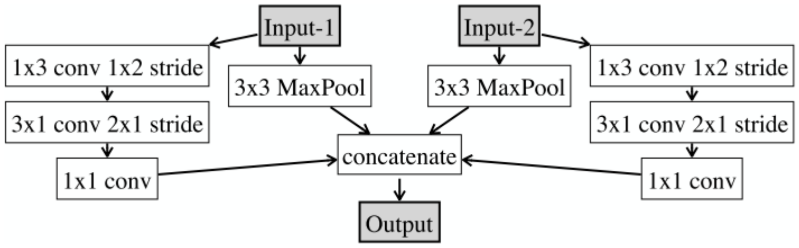

Figure 3. Manually designed reduction cell used in experiments when reduction cell is fixed. (Source, GDAS paper)

4. Dataset

CIFAR-10 [10] was used for main experiments and ablation studies as it is one of the standard datasets that were used to measure NAS performance. It has 50K training images and 10K test images of size 32x32. We can quickly experiment with it due to its size. Many papers also used CIFAR-10 to search for an architecture and transfer it to bigger datasets (e.g. ImageNet).
CIFAR-100 [10] was also used for experiments. It has 100 classes and 600 images for each class

5. Experiments and Results

To search CNN on CIFAR-10, the official train set of CIFAR-10 is split into two parts - $\mathbb{D}_T$ and $\mathbb{D}_V$ each containing 25000 images and are used as train and validation datasets as described in the search algorithm above. Official GDAS paper uses the following hyperparameters and training settings: number of nodes in a cell **B**=4, number of normal cells in one block is **N**=2, SGD with learning rate 0.025 annealed to 1e-3 by cosine schedule to optimize network weights W, Adam with learning rate 3e-4 for A, τ with an initial value of 10 and linearly reduced to 0.1. Searching for both normal and reduction cells takes about 4 hours and less than 3 hours to search for a normal cell while the reduction cell is fixed.

After searching, the winner architecture should be retrained from scratch to get the final accuracy. Final CNN is created by C=36, N=6, B=4, SGD optimizer with learning rate 0.025 annealed to 0 by cosine schedule. Auxiliary tower by 0.4 weight is used before the last reduction cell. Data augmentation involves random crops, horizontal flipping, and CutOut[11].

Table 1 and Table 2 show the results of other methods, GDAS paper, and our results on CIFAR-10 and CIFAR-100.

Type	Method	GPU	Params (million)	CIFAR-10 Error
Human	ResNet + CutOut	-	1.7	4.61
	DenseNet-BC	-	25.6	3.46
Macro	NAS [2]	800	7.1	4.47
	NAS + more filters [2]	800	37.4	3.65
	ENAS [4]	1	38	3.87
Micro	NASNET-A + CutOut [3]	450	3.3	2.65
	ENAS + CutOut [4]	1	4.6	2.89
	DARTS + CutOut [8]	1	3.4	2.76
	AmoebaNet-A + CutOut [6]	450	3.1	3.12
	GDAS +CutOut (paper)	1	3.4	2.93
	GDAS + CutOut (our result)	1	3.1	3.4
	GDAS (FRC) +CutOut (paper)	1	2.5	2.82
	GDAS (FRC) +CutOut (our result)	1	1.9	4.34

Table 1. Classification errors on CIFAR-10 of different models: Human made, Macro and Micro search. FRC - fix reduction cell. (Source of other methods' results - GDAS paper)

Type	Method	GPU	Params (million)	CIFAR-100 Error
Human	ResNet + CutOut	-	1.7	22.10
	DenseNet-BC	-	25.6	17.18
Micro	DARTS + CutOut [8]	1	3.4	17.54
	AmoebaNet-A + CutOut [6]	450	3.1	18.93
	GDAS +CutOut (paper)	1	3.4	18.38
	GDAS + CutOut (our result)	1	3.7	18.78
	GDAS (FRC) +CutOut (paper)	1	2.5	18.13
	GDAS (FRC) +CutOut (our result)	1	3.1	17.98

Table 2. Classification errors on CIFAR-100 of different models: FRC - fix reduction cell. (Source of other methods' results - GDAS paper)

As stated in the paper, we get faster convergence by fixing the reduction cell during the search, but in contrast to the paper, we get worse results after fully training the winning model. To understand the reason for such a significant difference, we started to study the final distribution of connections between cells and the architecture of the winning cells. As both normal and reduction cells have 4 nodes, to form a cell, two functions from the search space should be chosen for each node or 8 functions overall. If we search for both normal and reduction cells and construct a normal cell after the search, 6 of 8 connections in the normal cell are zeroize(make the input zero), which makes the cell extremely simple. If we omit zeroize function in the construction of the cell, then the normal cell contains different operations from search space and performs well. When the reduction cell is fixed, and only the normal cell is searched, again, there are many zeroize functions in the winner normal cell, and if we omit them, the resulting cell has mainly identity and pooling operations, which can be the reason for the lower result. In contrast, our metrics are consistent with the paper's results in the case of CIFAR-100 (Table 2), which lowers the risk of possible mistakes on our side.

We also experimented with taking not top 2 connections, but top2,3 or top3,4 (Table 3)

Method	**Params (million)**	**CIFAR-10 Error**
GDAS + CutOut (paper)	3.4	2.93
GDAS (FRC) + CutOut (paper)	2.5	2.82
GDAS (FRC) + CutOut (our result top1,2)	1.9	4.34
GDAS (FRC) + CutOut (our result top2,3)	1.9	5.02
GDAS (FRC) + CutOut (our result top3,4)	2.5	3.4
Random + CutOut (our result)	2.96	3.34
Random (FRC) + CutOut (our result)	2.5	4.05

Table 3. Comparison of results when search is done with fixed reduction cell and final normal cell is derived using different top connections.

Results in Table 3 show that taking top3,4 connections has lower error. Normal cell with top3,4 connections has convolution operations in contrast to top1,2 normal cell. This explains both the number of parameters and a lower error rate. Authors also state that GDAS can have variance in final results, and they have run the algorithm with 4 different seeds and taken the best one. We also have done the search with a fixed reduction cell with 4 different seeds aiming to get better normal cell architecture, but all experiments gave nearly the same simple normal cell containing only identity and pooling operations.

Random search is one of the important baselines to compare against new results in NAS. As authors have not established the random baseline, we have trained architectures with random normal and reduction cells and also with fixed reduction and random normal cells (Table 3). Architecture with random normal and reduction cells have achieved the highest score among our experiments, and highly comparable with paper's result with normal and reduction cells searched. This can be explained by the extremely good search space meaning that one can easily construct a well-performing cell from the paper's search space. It should be noticed that we in no way declare the GDAS algorithm unusable, but we have found several issues in reproducibility and also nearly equivalent results compared with the random baseline. Those problems may have roots in the search space of the algorithm, and the true power of GDAS can be assessed with a bigger search space. If the search space is big, and diverse enough, the random baseline will be lower and all gradient-based NAS algorithms can be compared on that search space. The search space used in GDAS paper comes from DARTS[8], and other gradient-based NAS algorithms [1, 12] do not change the search space too to be able to make apple-to-apple comparisons. But having the high results of random cells, a highly diverse benchmark search space for NAS algorithms is necessary to assess the performance of the models truly.

The necessity of the Gumbel-Max trick was explored. Instead of the paper's way of sampling a cell during training, we tried to sample without the trick. If $\mathbf{A}_{\mathbf{i,j}}$ is the logits vector of connection between node *i* and node *j*, we defined probability distribution by

$$\hat{h}_{i,j} = \frac{\exp\left(\mathrm{A}_{\mathrm{i,j}} / \tau\right)}{\sum_{k'=1}^{K} \exp\left(\mathrm{A}_{i,j} / \tau\right)} \tag{5}$$

After getting the corresponding probabilities, we sampled torch.multinomial() to get a sample from the probability distribution (torch.multinomial(probabilities) first samples a number from Uniform [0,1], divides [0,1] into bins of probabilities' respective length, and returns the number of the bin where the sampled number has fallen). We observed nearly no difference between the Gumbel-Max trick and direct sampling. Note that direct sampling can be done due to the fact that backpropagation is done through (5) and not the sampled index. In general Gumbel-Max trick is useful for backpropagation through samples from discrete probability distribution, but in this particular case we empirically saw no evidence between using it or direct sampling.

Apart from understanding the model's performance, experiments were conducted to achieve even faster convergence or better final cell. Hyperparameter tuning did not produce any noticeable difference. We tried to use mixed-precision training [13] to decrease the training time, but training time stayed the same, which can be due to the model's uncommon structure, and Nvidia apex library may not be optimized for such networks.

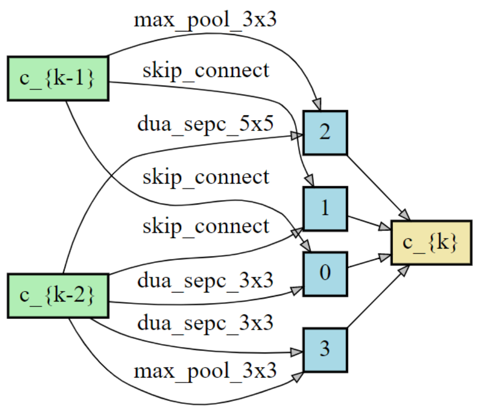

Figure 5. Found normal cell

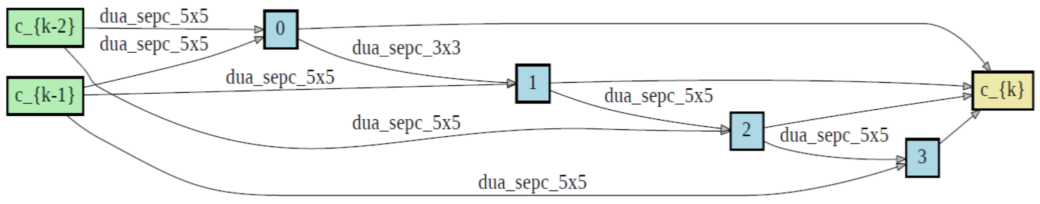

Figure 6. Found reduction cell

6. Discussion and Future Work

First of all, the capstone project resulted in a huge amount of research experience in the current state of neural architecture search. Given that NAS is one of the hottest topics in deep learning, the capstone project will be a good foundation for future experience in the NAS field. Although we could not fully reproduce the paper’s results and found some questionable points in the algorithm, it significantly raised my abilities of practical research experience in deep learning. I gained theoretical knowledge of not only NAS but also computer vision in general as the search space of different models involved new types of convolutional modules I didn't know about, tricks and tips of training CV models.

Due to time and hardware constraints, experiments were done only on CIFAR-10 and CIFAR-100. We plan to apply it to ImageNet - bigger and real-world dataset and compare the results with other ImageNet based models. Although there is some research on applying NAS to other tasks such as object detection and image segmentation, other CV areas are less explored for NAS and are both interesting and challenging. While NAS shows high results, we still have little insights on how it finds particular architectures, what are the underlying patterns that the model may discover during search. Answers to such questions can improve our understanding of neural networks and lead to much efficient and accurate models.

References

[1] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In CVPR, pages 1761–1770, 2019 (GDAS paper)
[2] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In International Conference on Learning Representations (ICLR), 2017.
[3] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 8697–8710, 2018
[4] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In International Conference on Machine Learning (ICML), pages 4095–4104, 2018
[5] Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. MnasNet: Platform-aware neural architecture search for mobile. CVPR, 2019.
[6] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In AAAIConference on Artificial Intelligence (AAAI), pages 4780–4789, 2019.
[7] David R. So, Chen Liang, and Quoc V. Le. 2019. The evolved transformer. In Proceedings of the 36th International Conference on Machine Learning (ICML)
[8] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. International Conference on Learning Representations (ICLR), 2019.
[9] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. International Conference on Learning Representations (ICLR), 2017.
[10] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.
[11] Terrance DeVries and Graham W Taylor. Improved regular- ization of convolutional neural networks with cutout. arXiv preprint arXiv:1708.04552, 2017.
[12] DATA: Differentiable ArchiTecture Approximation Jianlong Chang, Xinbang Zhang. Yiwen Guo, Gaofeng Meng, Shiming Xiang, Chunhong Pan. 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.
[13] https://github.com/NVIDIA/apex