UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Prabhant Singh

Analysis of Efficient Neural Architecture Search via Parameter Sharing

Master's Thesis (30 ECTS)

Supervisor:	Tobias Jacobs, PhD
Supervisor:	Meelis Kull, PhD

Tartu 2019

Analysis of Efficient Neural Architecture Search via Parameter Sharing

Abstract:

Deep learning based approaches have improved the state of the art performance of systems in various tasks such as language modeling, computer vision, object recognition, and image segmentation. Every task in deep learning requires custom architectures tailored specifically for that task. This resulted in high demand of domain experts for deep learning who can craft novel architectures. With the cost of domain experts rising and computational expenses falling, automating the neural architecture design is considered as an alternative.

The concept of neural architecture search has been introduced to tackle this problem. Neural architecture search can be considered a subset of automated machine learning(AutoML) domain [HKV18].

In this thesis, we have looked at a state of the art neural architecture search technique "Efficient neural architecture search via parameter sharing"(ENAS) [PGZ+18]. ENAS was introduced by Google Brain and it was a major improvement over its predecessor "Neural architecture search with Reinforcement learning"(NAS) [ZL16]. ENAS use a controller to sample the architectures from a search space which are later selected based on the measure defined by ENAS performance estimation strategy. Due to the impressive performance of ENAS, there has been research to apply ENAS and similar parameter sharing techniques in critical areas like medicine and diagnostics [GS19]. The motivation behind this thesis is to speed up and analyze the learning behavior of ENAS.

In this work we have analyzed the learning process of ENAS, evaluated ENAS performance estimation strategy and applied transfer learning on ENAS controller. We found that architectures do not improve with ENAS controller training via various experiments. We conclude that training of ENAS controller is not necessary and discuss limitations of ENAS performance estimation strategy.

Keywords:
Neural architecture search, deep learning, reinforcement learning, AutoML, machine learning, transfer learning.

CERCS:P176 - Artificial intelligence

Efektiivse neuroniarhitektuuri otsingu analüüs parameetrite jagamise kaudu

Lühikokkuvõte:

Sügavõppepõhised lähenemised on parandanud tehnika taset mitmesugustes ülesannetes nagu keele modelleerimine, raalnägemine, objekti tuvastamine ja pildi segmenteerimine. Iga sügavõppe ülesanne nõuab spetsiaalselt selle ülesande jaoks kohandatud arhitektuuri. Selle tõttu on suur nõudlus sügavõppe domeeniekspertide järele, kes suudavad uudseid arhitektuure luua ja käsitseda. Domeeniekspertide tasu tõusu ja arvutuslike kulutuste languse tõttu peetakse alternatiiviks tehisnärvivõrgu arhitektuuri disainimise automatiseerimist.

Selle probleemi lahendamiseks on kasutusele võetud närviarhitektuuri otsingu kontseptsioon. Tehisnärvivõrgu arhitektuuri otsingut võib pidada automatiseeritud masinõppe (AutoML) domeeni alamhulgaks. [HKV18]

Käesolevas töös on uuritud uusimat närvivõrgu arhitektuuri otsingutehnikat "Efektiivne närviarhitektuuri otsing parameetrite jagamise kaudu"(ENAS)[PGZ+18]. Google Brain tutvustas ENAS-i ja see oli suur areng võrreldes eelkäijaga „Närviarhitektuuri otsing stiimulõppega" (NAS) [ZL16]. ENAS kasutab kontrollerit, et võtta otsinguruumist arhitektuuride valim, millest omakorda valitakse arhitektuurid ENAS-i tulemuslikkuse hindamise strateegias määratletud meetme alusel. ENASi muljetavaldava jõudluse tõttu on uuritud ENAS-i ja sarnaste parameetrite jagamise tehnikate rakendamist olulistes valdkondades nagu meditsiin ja diagnostika [GS19]. Selle töö motivatsioon on kiirendada ja analüüsida ENAS-i õpikäitumist.

Selles töös on analüüsitud ENAS-i õppeprotsessi, hinnatud ENAS-i tulemuslikkuse hindamise strateegiat ja rakendatus ülekandeõpet ENAS-i kontrolleril. Erinevate katsete käigus leiti, et arhitektuurid ei muutu ENAS-i kontrolleri treenimise abil paremaks. Järeldati, et ENAS-i kontrolleri treenimine ei ole vajalik ja arutleti ENAS-i tulemuslikkuse hindamise strateegia piiranguid.

Võtmesõnad:
Neural architecture search, deep learning, reinforcement learning, AutoML.

CERCS:P176 - Artificial intelligence

Contents

1 Introduction

Machine learning has enabled us to achieve remarkable performance improvements in tasks related to computer vision, speech synthesis, language, and machine translation. This progress was achieved through complex architectures of deep neural networks like AlexNet [KSH12] in computer vision and BERT [DCLT18] in natural language processing.

One of the most crucial and important tasks for achieving this state of the art performance is research into novel architectures of deep neural networks. These novel architectures are carefully handcrafted by domain experts with years of experience. Finding novel architectures for new tasks can be highly human-intensive. As an example for computer vision, most of the current state of the art systems for image classification tasks are trained on standard academic datasets like CIFAR-10 [Kri09] or IMAGENET [DDS+09]. Using the same architectures for different datasets might not always be an optimal solution, so we need different architectures for different tasks. As deep learning is widely adopted in a wide range of application domains, the supply of the deep learning experts is not sufficient according to the demand. Hence a need of systems to automate neural network design arises.

To automate classical machine learning various methods were introduced like bayesian optimization, evolutionary strategies, and meta-learning. This automation helped to achieve state of the art accuracy on classification problems. Though for Neural networks automation remained a challenge due its complex and compute-intensive nature.

Neural architecture search was introduced to tackle this problem. Neural architecture search is the process to automate architecture engineering, hence reducing the human efforts in designing architecture by hand. Due to the wide search space and enormous permutations and combinations of architecture choices it is a highly computational and time-intensive task. Efficient Neural architecture search via parameter sharing (ENAS) [PGZ+18] was introduced to counter these limitations of neural architecture search.

Our motivation behind this work is to speed up ENAS. In this work we analyzed the ENAS technique, validated its performance as well as applied transfer learning on ENAS. Our research started with applying transfer learning to ENAS for both image classification and language modelling and then after observing the behaviour of controller in our experiments we decided to change our research direction and analyze the ENAS technique in general. In this work we have not included results from language modeling experiments as the authors of ENAS announced in May'2019 that their implementation of language modeling is incorrect hence we have also removed those results from this work to avoid any misleading conclusions.

In this thesis, we look at the need of neural architecture search, history of neural architecture search, current state of the art of neural architecture search in Section 2. In Section 3 we will discuss elements of neural architecture search. We review the ENAS technique, parameter sharing, and architecture construction approach in Section 4. A series of experiments are performed which provide insights into ENAS as well as performance evaluation of transfer learning with ENAS and ENAS performance evaluation strategy are described in Section 5. In Section 6 the results of experiments are discussed. The thesis is concluded in Section 7.

2 Background

Neural networks are being used in a range of application domains like language modeling, computer vision and time series forecasting. Due to the diverse nature of these tasks one cannot design a single universal neural network which can fit on every task. Sometimes, neural network design has to be altered even if the application domain remains the same but the dataset changes. For example in the area of image classification the state of the art architecture on IMAGENET [DDS+09] is different from the state of the art architecture for Caltech-256 [GHP07]. One way to overcome this problem is to construct a large neural network which can fit on every dataset but that comes with it's own problems like computational overkill to train a huge network. As all the state of the art architectures are constructed for academic datasets like CIFAR-10 or IMAGENET, an architecture designed for these datasets might not perform well for a dataset which might consist of images of different nature. With the cost of deep learning experts rising and computational expenses falling, automating the neural architecture design is a viable option.

The idea of automating neural architecture search was first proposed by Geoffrey et al. [MTH89] by using evolutionary strategies, later works used similar evolutionary strategies [ASP94][YL96][Kit90] to automate the design of neural networks. Though these approaches were not popular due to lack of interest in the field of neural networks itself. As deep learning got adopted widely the interest in the field grew. Today, neural architecture search can be considered as a subset of automated machine learning (AutoML). A typical Neural architecture search pipeline looks like the one described in Figure 1. The user supplies the data to the neural architecture search model, the neural architecture search model designs a model optimized for the particular dataset, and the final model can be trained from scratch by supplying it with data again.

The idea of using machine learning methods to determine neural network architectures has been popularized by the seminal work of Zoph et al. [ZL16], who have demonstrated that Neural Architecture Search (NAS) is able to find networks that outperform the best human designed neural architectures. Their approach used reinforcement learning at its core to find a neural network design. NAS also incorporated modern design elements like skip connections, batch normalization and rectified linear units. The most obvious drawback of the NAS method is its resource consumption, as the learning process involves generating a large number of deep neural networks, whose performance is evaluated by fully training them with the given training dataset and evaluating them with the test set, which takes several hours or even days per candidate architecture. This approach uses 450 GPUs for 3-4 days to find an optimal architecture for CIFAR-10.

There is a large body of follow-up work studying techniques to speed up the search process. Much attention has been gained by Pham et al.[PGZ+18], who have proposed

"Efficient Neural Architecture Search"(ENAS) method based on the idea to design the search space such that each candidate network is a subgraph of a joint network. The weights of the joint network are trained while an architecture generator (controller) is trained to select the best sub-network for the given dataset. Using this approach, the authors were able to improve the time complexity from thousands of GPU days to less than a single GPU day for the CIFAR-10 dataset while reaching a performance similar to the NAS [ZL16] results. We have discussed the ENAS approach in detail in Section 4.

Numerous alternative methods to ENAS have been presented as well. Suganuma et al.[SSN17] are among the authors who have provided evolutionary methods for finding neural architectures. The search method described by Liu et al. [LZS$^+$17] starts with evaluating simpler architectures and gradually makes them more complex, using a surrogate function to predict the performance of candidate architectures. Liu et al. [LSY18]) employ a continuous relaxation of the network selection problem, which enables the search procedure to use gradients to guide the search. A continuous search space is also employed by Luo et al. [LTQ$^+$18]. Here a model is trained to estimate the performance of an architecture based on a representation in a continuous space, providing a gradient based on which the architecture can be improved. A comprehensive survey of neural architecture methods has been published by Elsken et al. [EHH18].

One-shot approaches for finding neural architectures have been described in several recent works. Brock et al. [BLRW18] have proposed to have the weights of candidate neural architectures generated by a hypernetwork that is trained only once. A simpler mechanism inspired by the ENAS approach has been presented by Bender et al. [BKZ$^+$18]. Here the joint network containing all weights is first trained, and then the validation accuracy using these weights is used to predict the performance of candidate architectures. Instead of using a reinforcement learning based controller, the candidate architectures are generated by random search. While the latter two approaches are still applying search techniques which enumerate many possible architectures, our finding in Section 6 supports the hypothesis that a single architecture generated at random from an appropriate search space has competitive performance to architectures resulting from search processes.

Our findings in Section 6 are consistent with the very recent results of Adam et al.[AL], who have analyzed the behavior of the ENAS controller during search and found that its hidden state does not encode any properties of the generated architecture, providing an explanation of the observation made by Li et al. [LT19] that ENAS does not perform better than simple random architecture search. Our work is complementing this line of research by providing an experimental study of the learning progress of ENAS, demonstrating that the good performance of ENAS is already achieved before any controller training takes place.

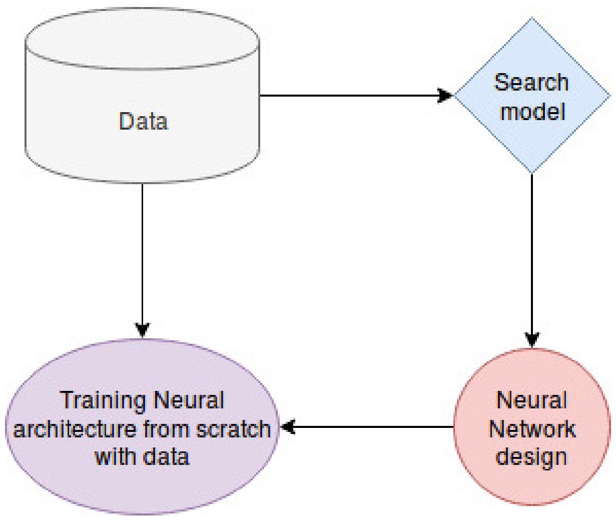


Figure 1. Basic neural architecture search pipeline

3 Neural architecture search

3.1 Elements of neural architecture search

To describe more about neural architecture search we use the terminology stated in Elsken et al. [EHH18] According to that, neural architecture search methods can be separated in 3 different components search space, search strategy and performance estimation strategy.

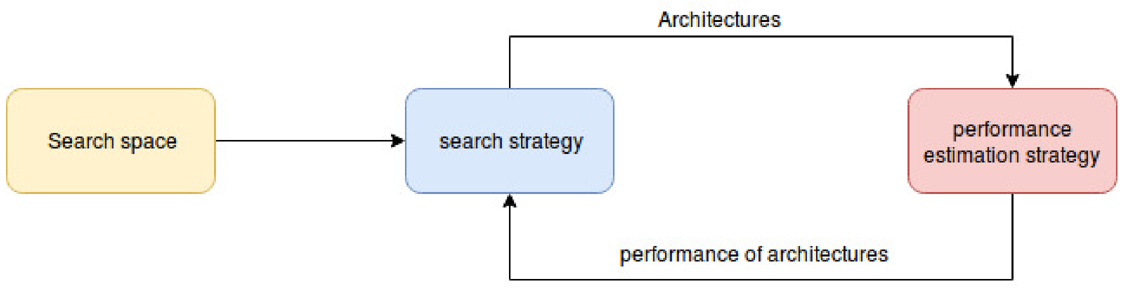


Figure 2. 3 Elements of Neural architecture search

1. **Search space:** The search space for the neural architecture search consists of the possible architectures which can define a neural network. As this search space can consist of vast number of possible combinations, one can incorporate prior knowledge and domain expertise to reduce the size of the search space. For example, one might limit the search space to convolutions and skip connections in image classification tasks as we already know that they are state of the art in this task. Similarly, we can limit search space to recurrent cells for language tasks. Though limiting this search space can result in human bias, which may prevent novel architectures from being discovered that go beyond human knowledge and expertise.

2. **Search Strategy:** The search strategy is defined as the algorithm used to explore the search space. We need to select an optimal search strategy which can find an optimal architecture quickly but also avoid getting stuck in the local minima.

3. **Performance estimation strategy:** Performance estimation strategy refers to the measure of success for our neural architecture search. The search strategy supplies architectures to the performance estimation strategy and the performance estimation strategy evaluates architectures and gives result back to search strategy.

In Figure 2 we can see how these three elements operate together.

3.2 Search space

We discuss different search spaces from recent works and our search space in this section.

A simple search space can be described as chain-structured neural networks. Chain-structured neural network can be described as a sequence of n layers, where layer i receives its input from a previous layer i-1 and serves as an output for layer i+1. Here layer 1 and n are input and output layer of the neural network. This search space can further be parameterized by

1. The number of layers
2. The set of layer operations, for example convolution, max pooling or average pooling.
3. The set of hyperparameters associated with layer operation, for example stride length.

Recent work on neural architecture search also incorporates modern design elements like skip connections. Skip connections provide search space with branched networks hence provides more degrees of freedom.

Motivated by repeated motifs in hand-crafted architectures, a new design philosophy of dubbed cells or blocks came. Here rather than searching the whole architecture, the neural architecture search method will search for cell blocks and final architecture can be constructed by stacking these blocks in a predefined manner. As shown in Figure 3 the cells are constructed to form a block and then this block can be repeated to construct the entire architecture. This search space has two different advantages compared to full architecture search.

1. The search space size is exponentially reduced as the cells are comparably small.
2. Cells are easily transferable to another dataset by adapting the number of cells used within the model.

The cell-based search was later incorporated in Efficient neural architecture search via parameter search (ENAS) [PGZ$^+$18], Our work is closely related to ENAS; hence we will utilize the same search space as ENAS.

3.3 Search strategy

Various kind of search strategies can be utilized to explore the search space for neural architecture search, including random search, bayesian optimization, reinforcement learning agents, evolutionary methods and gradient-based methods. As our work is

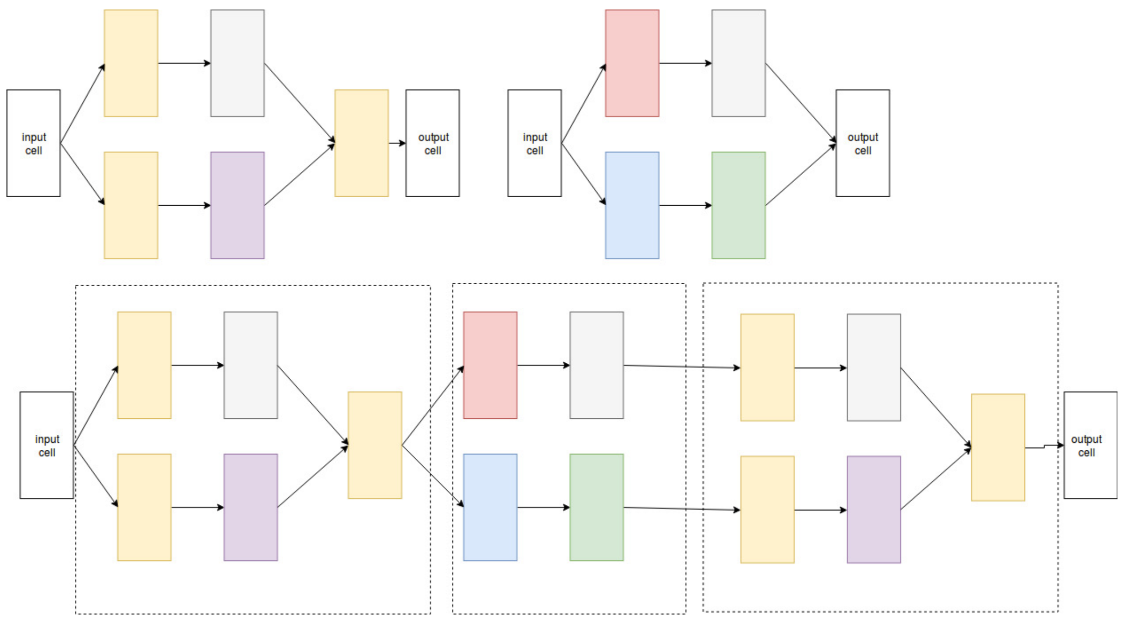


Figure 3. An example of a cell based architecture, here we can see that the cells are iteratively repeated to construct a final neural architecture

closely related to ENAS our search strategy is the same. ENAS frames the neural architecture search problem as a reinforcement learning problem.

In reinforcement learning, we have an agent which takes action, the choice of actions are based on reward maximization. In the case of neural architecture search the generation of neural architectures can be considered to be the agent action and the search space takes the place of the action space. The reward signal is the performance of the neural architecture on unseen or validation data. In ENAS our agent is an LSTM [HS97] which sequentially samples a string that in turn encodes a neural architecture. The agent is trained with REINFORCE policy gradient algorithm [Wil92]. We have described this process in Section 4.

3.4 Performance estimation strategy

The search strategy for neural architecture search is dependent on the signal from our performance estimation strategy. It can be described as a measure to judge the neural architecture A sampled by our search strategy. Let search strategy S samples an architecture A, a simple search strategy would be to train this architecture from scratch and measure its accuracy on unseen data. However, this approach is often infeasible as training each architecture from scratch is computationally extensive. Hence the requirement for more efficient performance estimation strategy arises. Current performance strategies can be classified into 4 major categories as described by Elsken et al. [EHH18].

1. **Lower fidelity estimates:** In lower fidelity estimation strategies the training time is reduced by training the network for fewer epochs on a subset of data or down-scaled data. These estimations can prove to be helpful in reducing training time but can create a bias on selecting a neural network. This technique can be especially problematic when the predictive performance difference between cheap approximations and a full evaluation is large.

2. **Learning curve extrapolation:** Learning curve extrapolation techniques have been quite common in the area of AutoML. In these estimation techniques, neural architecture performance is measured by training only on few epochs and selecting the neural architectures with a better initial learning curve. However as neural architecture performance is not always linear in fashion, therefore, there is a probability of selecting a neural architecture which can prove to be inefficient at the later part of the learning curve and eliminating a neural architecture with better performance, hence getting stuck at a relatively high local minimum.

3. **Weight inheritance/ network morphisms:** In these estimation techniques, rather than training from scratch, the models are warm started by inheriting weights from a larger parent model. These techniques have proven to be quite efficient, however, they can make architectures very large and lead to very complex architectures for simple problems.

4. **One shot models/ weight or parameter sharing:** These estimation techniques treat all the network architectures as a subgraph of one large directed acyclic graph and share the weights between architectures that have edges of this DAG in common. These shared weights exponentially reduce the computational cost of training the neural architectures. ENAS also uses weight sharing based performance estimation strategy.

4 ENAS

Efficient Neural Architecture search received considerable attention due to its ability to find well performing architectures within a considerably short span of time. ENAS was a significant improvement over NAS [ZL16]. NAS used over 450 GPUs for 3-4 days to find the architecture for CIFAR-10 and Pentree bank. In NAS all the sampled architectures were trained from scratch and after evaluation, their weights were thrown away. According to Pham et al. [PGZ^{+}18] ENAS improved efficiency of NAS by *"forcing all child models to share weights"*, hence avoiding to train each model from scratch to convergence. By using similar search strategy but implementing a search space and parameter sharing restriction, ENAS improved significantly over NAS and achieved the comparable performance in around 1 GPU day(to search the architecture) for CIFAR-10 and Pentree bank. In this section we discussed the components of ENAS, its search space, search strategy and performance estimation strategy. ENAS uses two kind of search spaces:

1. Macro search space
2. Micro search space

We have dedicated separate subsections to each search space to describe them in a clearer way.

4.1 Parameter sharing approach

ENAS uses parameter sharing approach. To enable this parameter sharing, all the architectures are treated as a subgraph of a single directed acyclic graph. This DAG can be considered as a superset of all the child models sampled by ENAS. In Figure 4 we can see a simple DAG defining an entire search space and two architectures sampled from that search space. The nodes of the graph represent local computations and the edges of the graph represent flow of information. Hence if the computation between two nodes is already done at one point while sampling an architecture then that computation or weight can also be used during the training of another architecture. Therefore two architectures in Figure 4 share the common parameters with each other. This parameter sharing approach is the main factor behind ENAS efficiency as it overcomes the major drawback of NAS. In NAS all the architectures were trained from scratch to estimate the performance and then their weights were discarded for next controller iteration. ENAS saves those weights and shares them whenever the nodes are sampled again from the DAG. So if an edge is common among architectures they share the same tensor. One question arises here is that if we train the architectures separately then the weights will be different because the architectures are different in topology and sharing weights will make everything suboptimal. Though that turns out to be false in case of ENAS, to justify

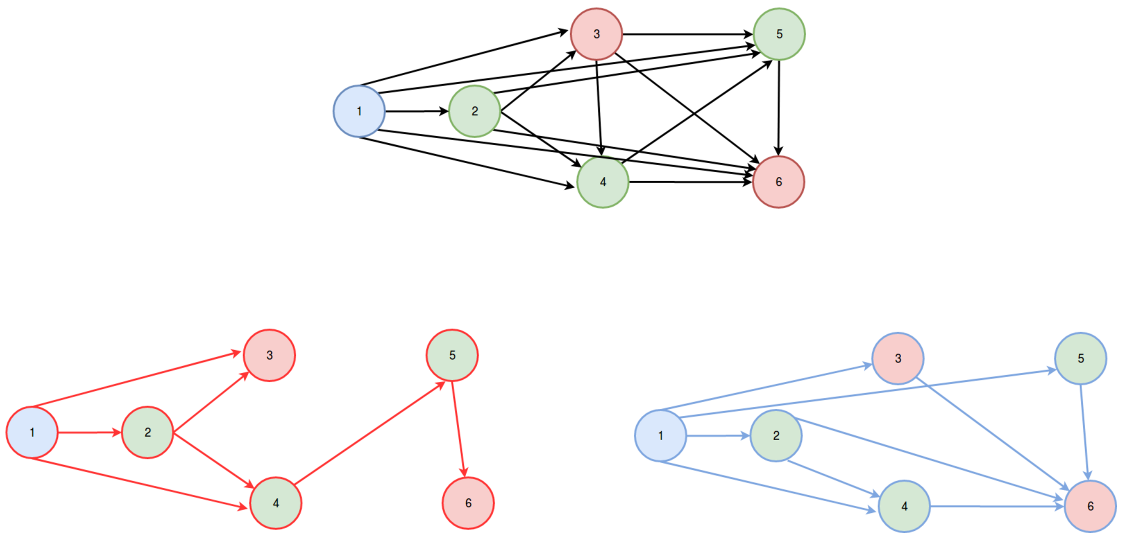


Figure 4. DAG for ENAS, Here let the first graph(top) be the entire search space of all the possible architectures. From this DAG two different architectures were sampled(down left and down right). Different colored edges represent different connections in the architectures. Though these architectures are different but they still share the weights of the common edges between the two graphs.

this efficiency Pham et al. [PGZ$^+$18] claims that the motivation behind this approach is multitask learning, where different tasks are given to the neural network and the sampled neural architectures tends to generalize well due to this approach. Though there are no theoretical proofs which proves that parameter sharing can find the local optimum but this approach is still being used in the state of the art neural architecture search techniques like DARTS [LSY18] and Neural Arhictecture Optimization [LTQ$^+$18].

4.2 ENAS search strategy

ENAS uses a controller for sampling the architectures within the search space. The controller for ENAS is an LSTM with 100 hidden units. This LSTM samples architectures via softmax classifiers in an autoregressive manner, which means it predicts one hyperparameter at a time, conditioned on previous predictions. For ENAS there are two sets of learnable parameters.

- Weights of the controller: θ
- Shared parameters for the child models in the DAG: ω

The training happens in two alternating phases, the first phase trains shared parameters ω of the child models and the second phase trains the parameters θ of the controller.

For Image classification tasks shared parameters ω are trained on 45000 training images, separated into mini-batches of size 128. Where the gradient ∇_ω is computed using backpropagation. The parameters of the controller θ are trained for a fixed number of steps, we set that number to 2000 in accordance to the original experimental settings by Pham et al. [PGZ$^+$18].

4.2.1 Training shared parameters

To train the shared parameters the controller policy is fixed. Then the stochastic gradient descent is applied on shared parameters to minimize the expected loss function, The gradient is computed using Monte Carlo estimate method:

$$\nabla_\omega \mathbb{E}_{m\sim\pi(;\theta)}\left[\mathcal{L}(m;\omega)\right] \approx \frac{1}{M}\sum_{i=1}^{M}\nabla_\omega \mathcal{L}(m_i,\omega) \tag{1}$$

Here :
$\sim \pi(m;\theta)$: Controller's policy for the model with it's parameters
$\mathbb{E}_{m\sim\pi}\left[\mathcal{L}(m;\omega)\right]$: Expected Loss function
$\mathcal{L}(m;\omega)$: Standard cross entropy loss computed on a training data with a a model sampled from $\pi(m;\theta)$

This estimate of the gradient has a higher variance than standard stochastic gradient descent where m is fixed. The surprising fact discovered by Pham et al. [PGZ$^+$18] is that $M = 1$ woks fine. So shared parameters can be updated from any single model m sampled from $\sim \pi(m;\theta)$. The shared parameters ω are trained during an entire pass through the data.

4.2.2 Training controller parameters

To train controller parameters θ, shared weights ω are fixed. The weights of the controller are updated to maximize the expected reward $\mathbb{E}_{m\sim\pi}\left[\mathcal{R}(m,\omega)\right]$ by employing Adam optimizer. The gradient for Adam optimizer is calculated using REINFORCE [Wil92] with a moving average baseline to reduce variance. To understand this training process we will discuss framing neural architecture search as a reinforcement learning problem. The reward $\mathcal{R}(m,\omega)$ is calculated using the holdout validation set. Using validation set pushes ENAS to find the architectures which generalize well. Though as this validation set accuracy acts like a reward signal for the controller, there might be a possibility of controller overfitting on the validation dataset.

4.3 Macro search space

Macro search space in ENAS consists of the entire architecture search space where the controller samples the entire architecture design.
In ENAS macro architecture for a neural network with N layers consists of N parts, indexed by 1, 2, 3, ..., N. i consists of:

- A number in [0, 1, 2, 3, 4, 5] that specifies the operation at layer i-th, corresponding to conv 3x3, separable conv 3x3, conv 5x5, separable conv 5x5, average pooling, max pooling.
- A sequence of i - 1 numbers, each is either 0 or 1, indicating whether a skip connection should be formed from the corresponding past layer to the current layer.

An example of architecture sampled from macro search space for a 24 layer convolution neural network looks like this:
[2]
[5 0]
[1 0 0]
[3 1 1 0]
[3 0 0 1 1]
[1 0 0 0 0 1]
[0 0 0 0 0 0 0]
[3 0 0 0 0 0 0 1]
[4 0 0 0 0 1 0 0 0]
[4 1 0 1 0 1 0 1 0 1]
[1 1 1 0 0 1 0 1 1 1 1]
[5 1 0 1 0 1 0 1 0 0 0 1]
[1 0 0 1 1 0 0 1 0 1 0 0 0]
[2 1 0 1 0 1 0 0 0 1 1 0 0 0]
[3 0 0 1 1 0 0 0 1 0 1 1 0 0 1]
[2 0 0 1 0 1 1 0 1 0 0 1 1 0 1 1]
[4 0 0 1 1 0 1 0 0 0 0 1 1 0 1 0 1]
[5 0 0 0 1 1 0 1 0 1 1 1 1 1 0 0 0 0]
[1 0 1 0 0 0 1 1 0 1 0 0 1 0 0 0 0 0 0]
[1 0 0 0 0 1 1 1 0 0 0 0 1 0 0 0 1 1 1 1]
[4 1 1 0 1 1 0 1 1 0 0 1 1 0 0 0 0 0 0 1 1]
[5 1 0 1 0 0 1 0 1 0 1 0 0 0 1 0 1 0 1 0 1 0]
[5 1 1 1 1 0 1 1 0 0 0 1 1 1 1 1 1 1 0 1 1 1 1]
[1 1 1 1 0 1 1 0 0 0 0 0 1 0 1 1 1 0 0 0 0 1 0 1]

Here the first number in the array represents the type of operations and next number represents whether we want a skip connection with previous layers or not.

4.4 Designing architectures in macro space

To design convolution network the the controller samples two operations from macro search space:

- Previous nodes to connect to
- Computation operations to use

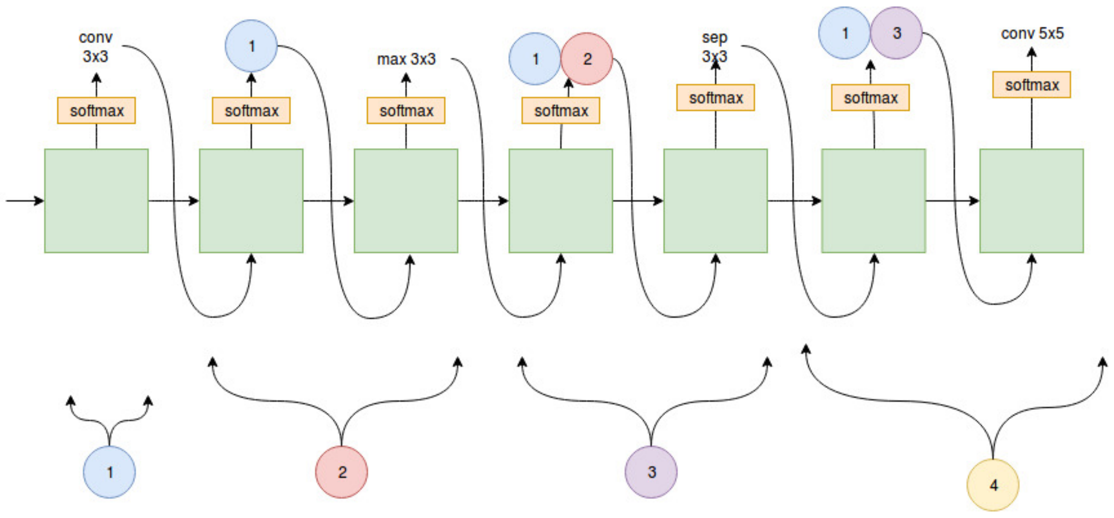


Figure 5. ENAS controller sampling procedure on macro search space, here we can see our controller sampled operations and skip connections layer by layer.

An example of ENAS controller sampling architecture from macro search space of shown in . Let’s say we have 4 nodes to sample:

1. The first node is sampled as *conv 3x3*.
2. Second node is sampled as a *max 3x3* with skip connection to node 1.
3. Third node is sampled with operation *sepconv 3x3* and skip connections with node 1 and 2.
4. Fourth node is sampled with operation *conv 5x5* and skip connection with node 1 and node 3.

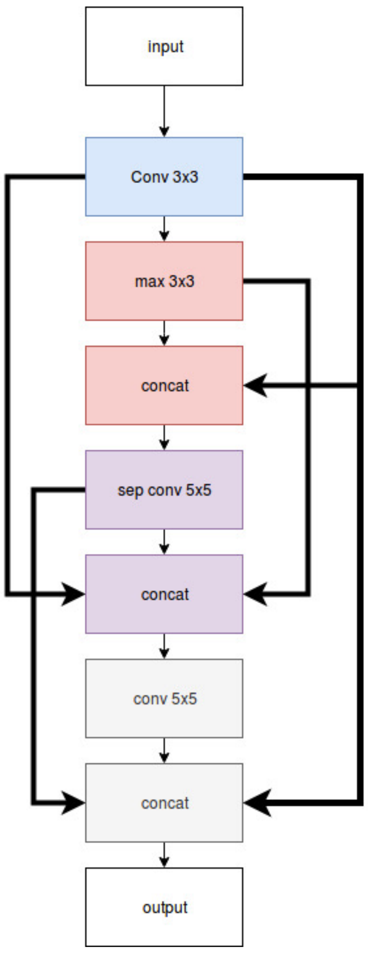


Figure 6. Sampled macro architecture by ENAS controller; Architecture sampled by ENAS controller as described in Figure 5

.

Final architecture for above example can be seen in Figure 6. The corresponding DAG is shown in Figure 7

Complexity: Previous node connection allows possibility of skip connections. So at layer n there are already n-1 layers are sampled which gives possibility to connect to previous layers leading upto 2^{k-1} decisions. For the operations the controller has option of 6 operations as described in section 3.1 . Making these decisions for a total N times so that we can sample N layers. As all decisions are independent of each other this fives us $6^{L} \times 2^{L(L-1)/2}$ networks in the search space.

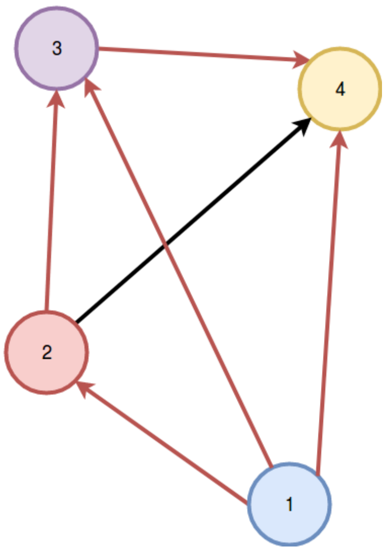


Figure 7. Corresponding DAG for macro architecture described in Figure 6, the red edges corresponds to the activated connections

.

4.5 Micro search space

The idea of micro search was first proposed by Zoph et al. [ZVSL17]. The idea behind micro search is to construct a best possible convolution layer(or "cell") instead of the entire convolution neural network architecture. The overall architecture of the networks are manually predetermined. The convolution network consist of convolution cells repeated many times. This network was termed as NASNet[ZVSL17], though efficient this network still required 2000 GPU-hours to search an architecture for CIFAR-10 dataset.
To describe micro search in simple terms, we can think neural network as a set of building blocks. Blocks are repeated to construct the architecture. The blocks consist of N "cells" and every cell is made up of B nodes. An output from a block is treated as input to the other block. Similar process happens with the cells and nodes. A simple representation of a block is shown in Figure 8. In Figure 9 we can see the entire neural network constructed with 3 blocks. This idea was later incorporated into ENAS by Pham et al. [PGZ$^+$18]. Instead of designing an entire neural network, ENAS controller can sample smaller modules then connect them together to form a large network. ENAS again uses parameter sharing with micro search to increase the efficiency. The controller search strategy and the performance estimation strategy remains the same for micro search as it was in macro search.
For micro search ENAS computational graph consists of B set of nodes. Here B is the number of computations inside the cell. The first two nodes sample are treated as inputs

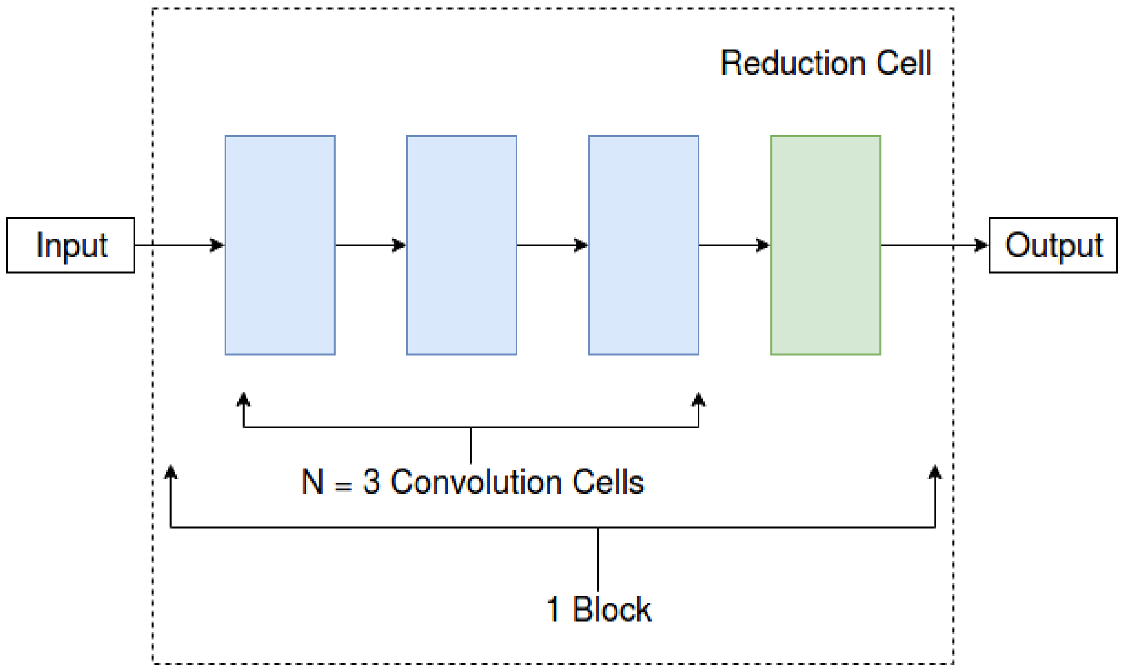


Figure 8. Structure of a block generated with micro search. The block with $N = 3$ consist of 3 convolution cell and 1 reduction cell.

to the cell. These inputs can be the output from the previous cells or first 2 sampled nodes. For remaining $B - 2$ nodes the ENAS controller makes two decisions:

- To use the previous two nodes as the inputs to the current node
- Operations applied to the two sample nodes

The micro search space consist of cells which act as a building blocks to the final architecture. In ENAS micro architecture can be described as a set of two cells which form a block and then that block is iteratively repeated. In ENAS the controller sampled 2 arrays every iteration, the first array described the convolution cell and the second array

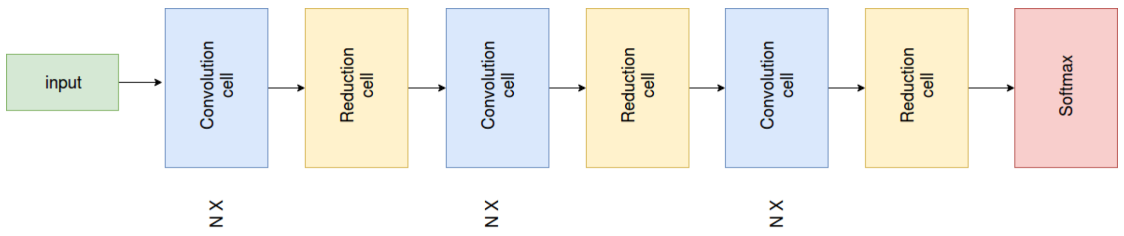


Figure 9. A simple neural network architecture constructed with 3 blocks, each block consist of N convolution cells and 1 reduction cell.

described the reduction cell.
The cell architecture sampled by controller looks like this:
[0 3 1 0 2 1 0 0 2 1 3 1 0 0 1 3 0 1 5 1]
[1 0 1 0 1 1 1 0 1 0 2 0 3 2 2 1 3 4 4 0]
Here the array is in the formation of first index, first operation, second index, second operation. The indexes can be any previous index. The operations can be between [0, 1 ,2 ,3 ,4] corresponding to separable conv 3x3, separable conv 5x5, average pooling, max pooling and identity respectively.

4.6 Designing convolution cells

The cell design procedure is as follow, let's say $B = 4$ for our experiments here. Every cell will contain 4 operations, out of which 2 operations will be chosen from the previously sampled nodes. The controller does the following steps to sample a single cell block:

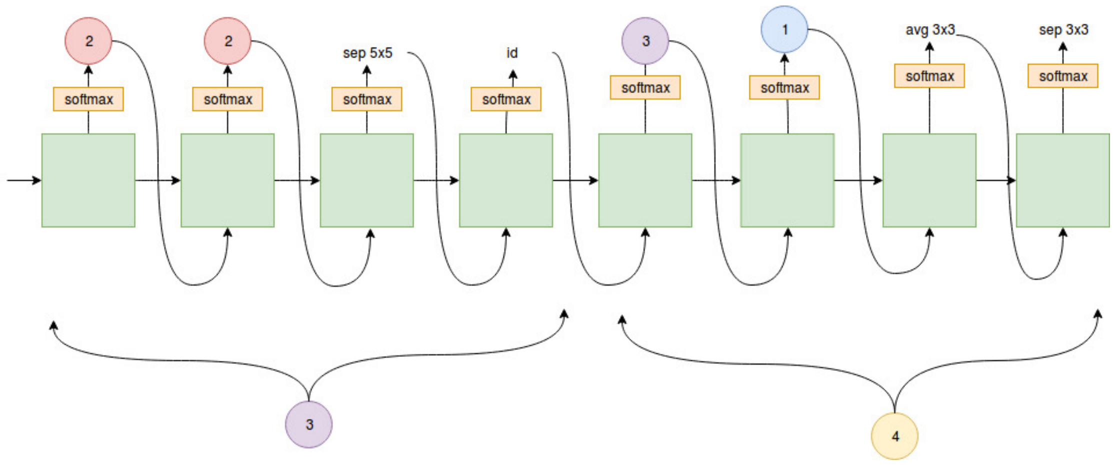


Figure 10. Micro architecture search by ENAS controller

1. Node 1 and 2 are input nodes so no computations are required from controller side here. let o_1 and o_2 be the output from these nodes. Node 1 and node 2 can either be single operations or independent cells.

2. At node 3: the controller samples two previous nodes and two operations. In *Top Left*, it samples *node* 2, *node* 2, *separable_conv_5x5*, and *identity*.

3. The previous step leads to $o_3 = \text{sep\_conv\_5x5}(o_2) + \text{id}(o_2)$.

4. At node 4: the controller generates a skip connection with *node* 3, *node* 1, and sample operations *avg_pool_3x3*, and *sep_conv_3x3*. This leads to $o_4 = \text{avg\_pool\_3x3}(o_3) + \text{sep\_conv\_3x3}(o_1)$.

5. Since all nodes other than o_4 were used as inputs to another node, the loose end, o_4, is now the cell's output. To deal with multiple loose ends, ENAS concatenates the loose ends along the depth dimension.Finally all the concatenated loose ends together forms the final cell output.

Reduction cell: A reduction cell can also be constructed from the search space by sampling the computational graph from the search space and applying stride 2 to all operations. This helps in reducing the spatial dimensions of the input bu a factor of 2. ENAS samples the reduction cell conditioned on the convolution cell. This sampling makes the controller run for $2(B-2)$ blocks.

Figure 10 represents the procedure we described in a visual way, Figure 11 is the final cell architecture sampled by the controller and Figure 12 is the corresponding DAG.

Complexity: As in micro search space the controller can sample two nodes from $i-1$ previous nodes and all the decisions are independent. For designing cells the complexity for micro search space drops down to $(5 \times (B-2)!)^4$ which is way less than macro search space.

4.7 ENAS performance estimation strategy

ENAS uses weight sharing based performance estimation strategy. As we mentioned earlier that ENAS samples architectures from a DAG with shared with node edges sharing weights. After every epoch controller samples, 10 architectures and ENAS evaluates those architectures based on a hold out validation set and sends that as a reward signal back to the controller. In case of our image classification task the performance estimate is the accuracy on the validation dataset.

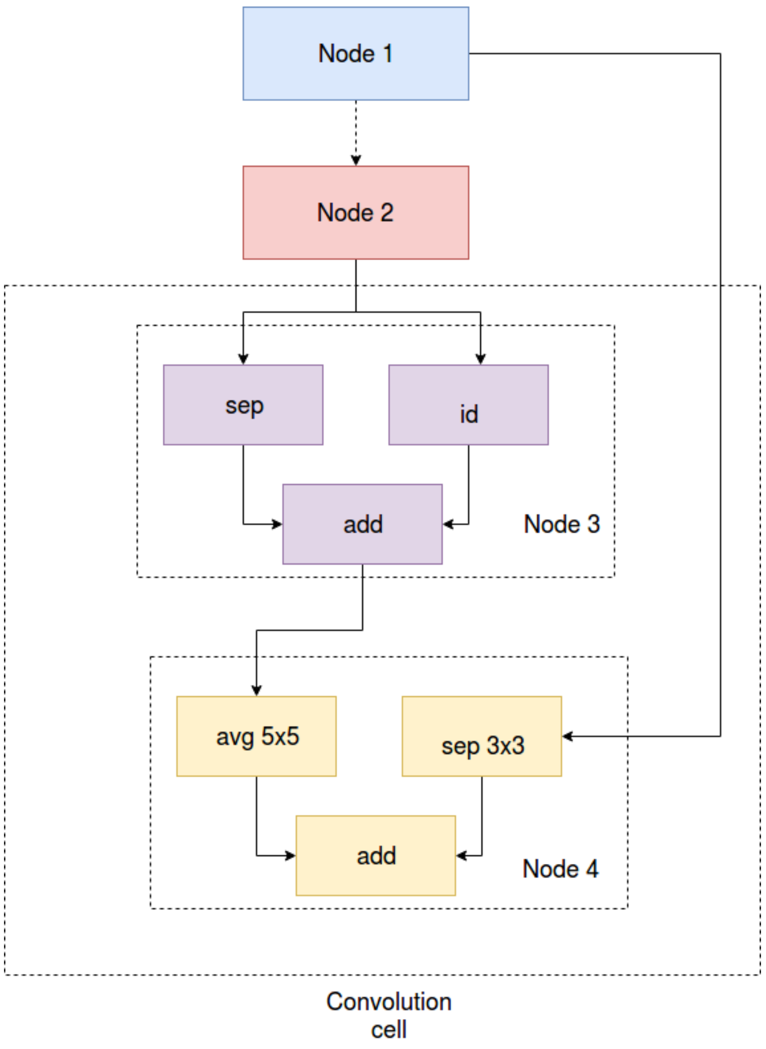


Figure 11. Final cell architecture sampled by the controller for micro search

5 Experimental setup

For our experiments we used the same settings as the authors described here (https://github.com/melodyguan/
. We used CIFAR-10 and CIFAR-100 datasets for our experiments.

1. CIFAR-10: CIFAR-10 dataset consist of 50,000 training image and 10,000 test images. it's a very standard dataset used for image classification benchmarks.
2. CIFAR-100: CIFAR-100 dataset is just like CIFAR-10 except it has 100 classes containing 600 images each. There are 500 training images and 100 testing images for each class.

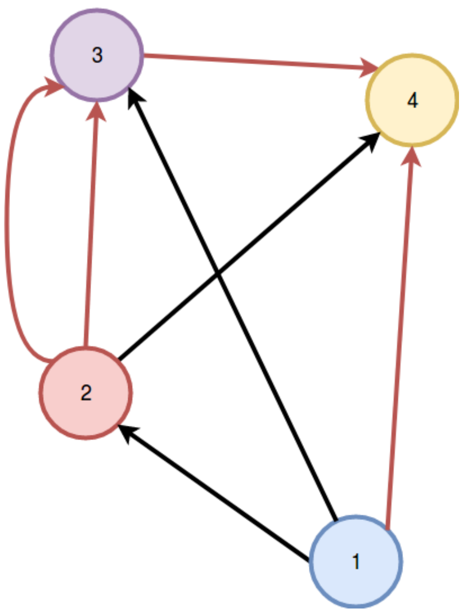


Figure 12. Corresponding DAG for Micro architecture search, the red edges represents the activated connections.

Using CIFAR-100 allows us to independently evaluate ENAS as it has not been tested on CIFAR-100. CIFAR-100 dataset is much more challenging dataset due to lesser number of training images per class.

For ENAS uses following data pre-processing and augmentation techniques:

1. Subtracting channel mean and dividing the channel standard deviation

2. Centrally padding the training images to 40 X 40

3. Randomly cropping them to 32 X 32 then randomly flipping them horizontally.

The hyper parameters for our macro search are as follows:

Hyperparameter	**Value**
batch size	128
number of child layers	24
number of output filters	96
child L2 Regularization rate	0.00025
number of child branches	6
cell layers	5
child keep probability	0.90
child max learning rate	0.05
child minimum learning rate	0.0005
controller entropy weight	0.0001
controller training steps	50
controller learning rate	0.001

The hyper parameters for our micro search are as follows:

Hyperparameter	**Value**
batch size	160
number of child layers	15
number of output filters	36
child L2 Regularization rate	0.00025
number of child branches	5
cell layers	5
child keep probability	0.80
child max learning rate	0.05
child minimum learning rate	0.0005
controller entropy weight	0.0001
controller training steps	50
controller learning rate	0.001

5.1 Transfer learning experiments

Transfer learning refers to the idea that one model trained for one task can be reused on other similar tasks with some fine-tuning. Our main idea at the beginning of this research was to apply transfer learning on ENAS controller to get more efficient performance from ENAS. Usually, transfer learning is very efficient for warm starting the model by initializing it with weights from a pre-trained model on a task. In this setting, we trained an ENAS controller on the CIFAR-10 dataset and then transferred that controller weights to the more challenging CIFAR-100 dataset.

We first trained the controller for 310 epochs as recommended by authors on CIFAR-10 on macro search space. We saved the file in pickle format and then initialized the controller for CIFAR-100 task with saved weights.

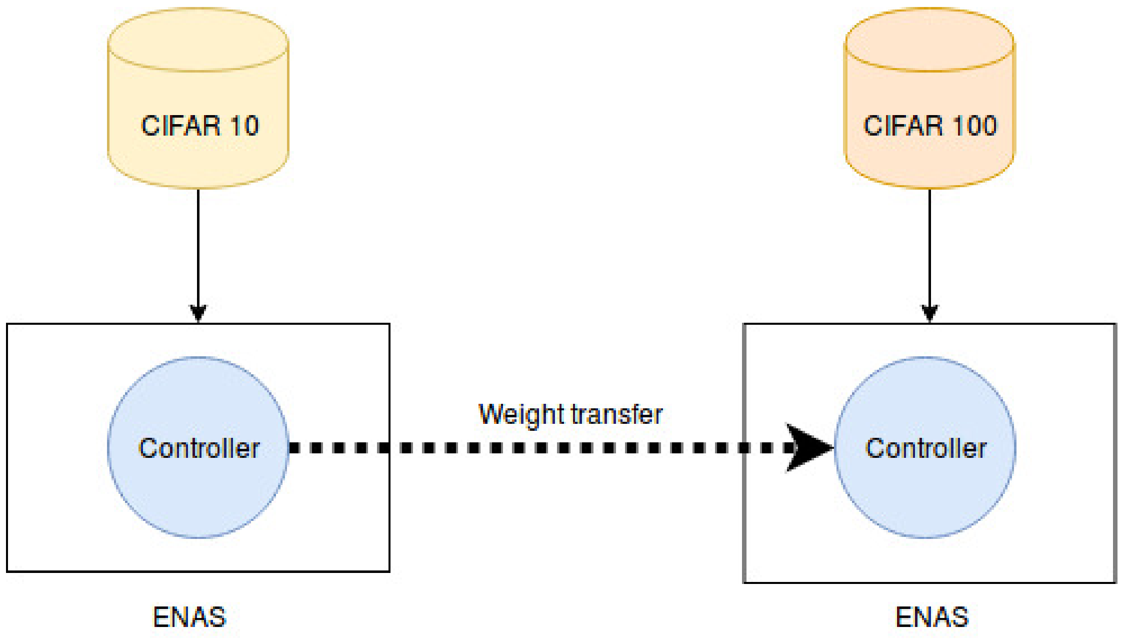
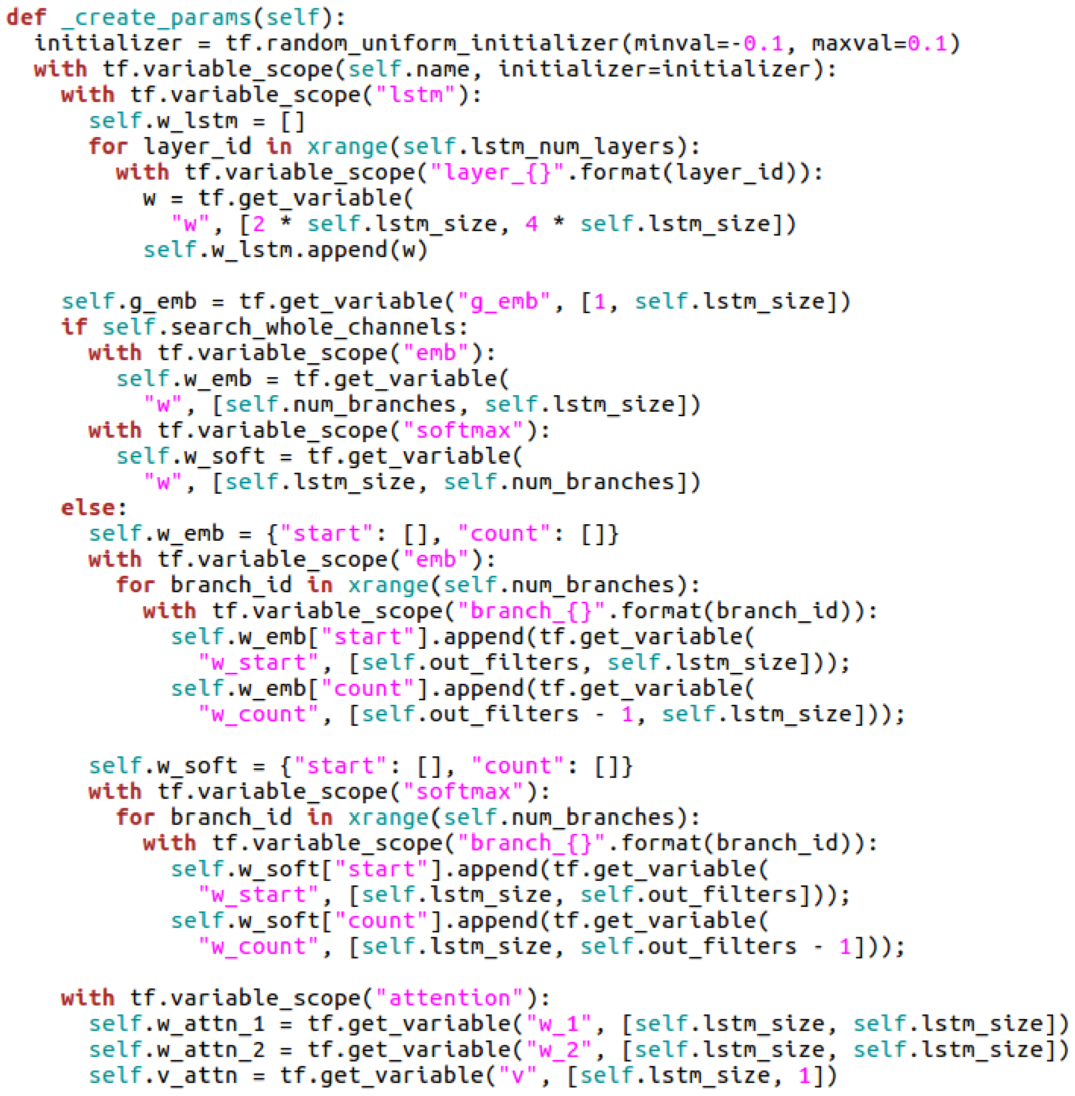


Figure 13. Transfer learning of ENAS controller

For our experiments we trained the architectures from scratch sampled by controller at epoch 310, 155 and 100.



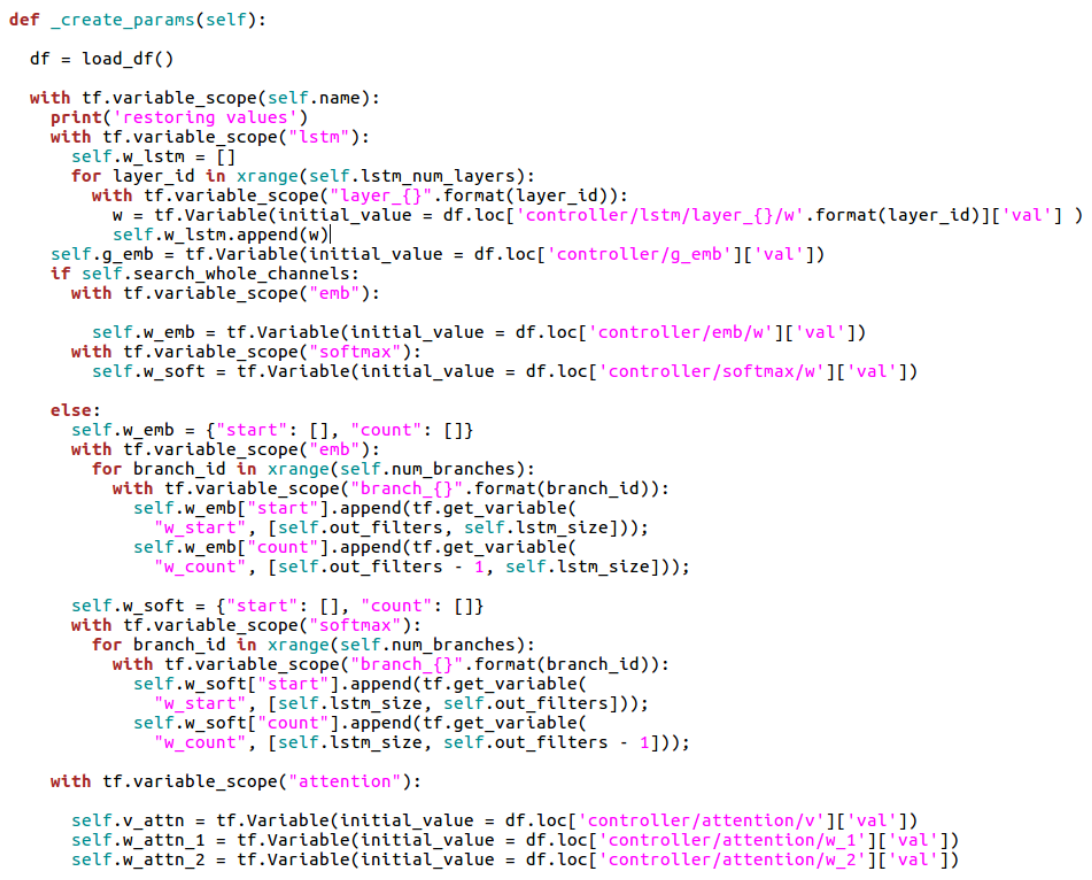
```python
def _create_params(self):
  initializer = tf.random_uniform_initializer(minval=-0.1, maxval=0.1)
  with tf.variable_scope(self.name, initializer=initializer):
    with tf.variable_scope("lstm"):
      self.w_lstm = []
      for layer_id in xrange(self.lstm_num_layers):
        with tf.variable_scope("layer_{}".format(layer_id)):
          w = tf.get_variable(
            "w", [2 * self.lstm_size, 4 * self.lstm_size])
          self.w_lstm.append(w)

    self.g_emb = tf.get_variable("g_emb", [1, self.lstm_size])
    if self.search_whole_channels:
      with tf.variable_scope("emb"):
        self.w_emb = tf.get_variable(
          "w", [self.num_branches, self.lstm_size])
      with tf.variable_scope("softmax"):
        self.w_soft = tf.get_variable(
          "w", [self.lstm_size, self.num_branches])
    else:
      self.w_emb = {"start": [], "count": []}
      with tf.variable_scope("emb"):
        for branch_id in xrange(self.num_branches):
          with tf.variable_scope("branch_{}".format(branch_id)):
            self.w_emb["start"].append(tf.get_variable(
              "w_start", [self.out_filters, self.lstm_size]));
            self.w_emb["count"].append(tf.get_variable(
              "w_count", [self.out_filters - 1, self.lstm_size]));

      self.w_soft = {"start": [], "count": []}
      with tf.variable_scope("softmax"):
        for branch_id in xrange(self.num_branches):
          with tf.variable_scope("branch_{}".format(branch_id)):
            self.w_soft["start"].append(tf.get_variable(
              "w_start", [self.lstm_size, self.out_filters]));
            self.w_soft["count"].append(tf.get_variable(
              "w_count", [self.lstm_size, self.out_filters - 1]));

    with tf.variable_scope("attention"):
      self.w_attn_1 = tf.get_variable("w_1", [self.lstm_size, self.lstm_size])
      self.w_attn_2 = tf.get_variable("w_2", [self.lstm_size, self.lstm_size])
      self.v_attn = tf.get_variable("v", [self.lstm_size, 1])
```

Figure 14. Structure of ENAS controller



```
def _create_params(self):

  df = load_df()

  with tf.variable_scope(self.name):
    print('restoring values')
    with tf.variable_scope("lstm"):
      self.w_lstm = []
      for layer_id in xrange(self.lstm_num_layers):
        with tf.variable_scope("layer_{}".format(layer_id)):
          w = tf.Variable(initial_value = df.loc['controller/lstm/layer_{}/w'.format(layer_id)]['val'] )
          self.w_lstm.append(w)
    self.g_emb = tf.Variable(initial_value = df.loc['controller/g_emb']['val'])
    if self.search_whole_channels:
      with tf.variable_scope("emb"):

        self.w_emb = tf.Variable(initial_value = df.loc['controller/emb/w']['val'])
      with tf.variable_scope("softmax"):
        self.w_soft = tf.Variable(initial_value = df.loc['controller/softmax/w']['val'])

    else:
      self.w_emb = {"start": [], "count": []}
      with tf.variable_scope("emb"):
        for branch_id in xrange(self.num_branches):
          with tf.variable_scope("branch_{}".format(branch_id)):
            self.w_emb["start"].append(tf.get_variable(
              "w_start", [self.out_filters, self.lstm_size]));
            self.w_emb["count"].append(tf.get_variable(
              "w_count", [self.out_filters - 1, self.lstm_size]));

      self.w_soft = {"start": [], "count": []}
      with tf.variable_scope("softmax"):
        for branch_id in xrange(self.num_branches):
          with tf.variable_scope("branch_{}".format(branch_id)):
            self.w_soft["start"].append(tf.get_variable(
              "w_start", [self.lstm_size, self.out_filters]));
            self.w_soft["count"].append(tf.get_variable(
              "w_count", [self.lstm_size, self.out_filters - 1]));

    with tf.variable_scope("attention"):

      self.v_attn = tf.Variable(initial_value = df.loc['controller/attention/v']['val'])
      self.w_attn_1 = tf.Variable(initial_value = df.loc['controller/attention/w_1']['val'])
      self.w_attn_2 = tf.Variable(initial_value = df.loc['controller/attention/w_2']['val'])
```

Figure 15. Transferred ENAS controller, here we can see that all the layer initialisation for LSTM were replaced by dataframe values stored from the last controller training on CIFAR-10

5.2 Learning curve experiment

For learning curve evaluation we trained ENAS controller for 310 epoch as recommended by authors for finding the best architecture. Then we took sampled architectures at epoch 1 and epoch 310. We trained these architectures from scratch on CIFAR-10 and CIFAR-100 datasets. We had 3 runs for macro search space and 2 runs for micro search space. This experiment is performed to evaluate architectures sampled at different epoch by the controller and checking for the performance variance of these sampled architectures at initial and final epochs.

5.3 Performance estimation strategy experiment

For evaluation of the performance estimation strategy for ENAS we trained architectures sampled on epoch 155 from scratch. ENAS controller samples 10 architectures every epoch. We took 5 architectures sampled at epoch 155 with various validation accuracy and trained them from scratch. Our reason for choosing epoch 155 is to avoid the phenomenon that all ENAS sampled architectures might give the best possible accuracy when sampled by the controller at the last epoch. At epoch 310 the difference between the validation accuracies of different architectures is very small, at epoch 155 the difference between the validation accuracies is large(20 percent), hence an epoch in the middle of the training is chosen. This experiment is performed to find out any correlation between our performance metric performance estimation strategy and actual performance of the architectures.

6 Results

6.1 Transfer learning

Now we compare the results from our transfer learning experiments with ENAS trained from scratch

Dataset	Search Space	Sampled epoch	Transfer applied	Accuracy
CIFAR-100	macro	310	NO	80.55
CIFAR-100	macro	100	NO	80.78
CIFAR-100	macro	155	NO	80.33
CIFAR-100	macro	310	YES	80.35
CIFAR-100	macro	100	YES	80.19
CIFAR-100	macro	155	YES	80.39

Table 1. Performance of architectures trained by CIFAR-100. The controller was trained for macro search space for 100, 155 an 310 respectively in case of both transfer learning and ENAS.Before testing, the architectures were retrained for 310 epochs.

Here we can notice some unexpected results as the architectures sampled at epoch 100 and 155 gave the same performance as compared to the final architectures while training from scratch as well as in transfer learning. These results became our motivation to conduct learning curve evaluation for ENAS controller.

We also noticed that for CIFAR-10, ENAS samples final architectures only consisting of convolution operations but for CIFAR-100 it samples the architectures involving pooling operations and convolution operations. Though after the controller transfer ENAS only sampled convolution operations for CIFAR-100, which concludes that the ENAS transfer was successful. We also notice in Figure 16 and Figure 17 that learning curve of every architecture sampled by ENAS for CIFAR-100 follows the same learning curve. This phenomenon should be inspected in future work.

6.2 Learning curve evaluation

For CIFAR-10 and CIFAR-100 we trained architectures with highest validation accuracy sampled at initial epoch and final epoch by the controller. We observed that there is no significant difference between the accuracy of trained architectures in both cases. We ran it several times and observed no significant difference except for a few outliers which might occur due to random weight initialization.

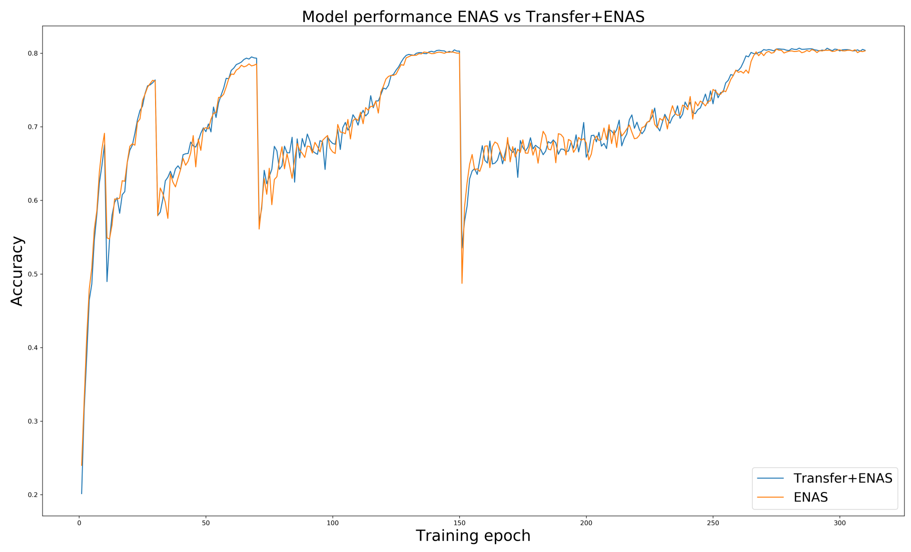


Figure 16. Learning curve of sampled architectures for both ENAS and transfer learning with ENAS for epoch 155

The analysis of the learning curve leads to the conclusion that ENAS controller does not learn from its past actions as the architecture sampled on the first epoch performed as good as the architecture sampled on the final epoch. Also, the fact the search space is highly constrained and almost every model sampled has the capacity to fit both CIFAR-10 and CIFAR-100.

We were not able to train architecture sampled from micro search space for CIFAR-10 due to various GPU errors.

We notice that there is no significant difference in initial and final architectures in both macro and micro search. This leads up to question the use of controller and search strategy in general in case of ENAS. As it might be possible that the architecture construction procedure is so sophisticated that any random controller can lead to a good architecture.

One might question that a 24 layer network is too big for these datasets, hence any network with skip connections in the constrained search space will perform equally well. To avoid that, we also conducted experiments with 12 layer networks. We noticed some decrease in accuracy with 12 layer architectures but the behaviour of ENAS for 12 layer was similar to ENAS with 24 layers.

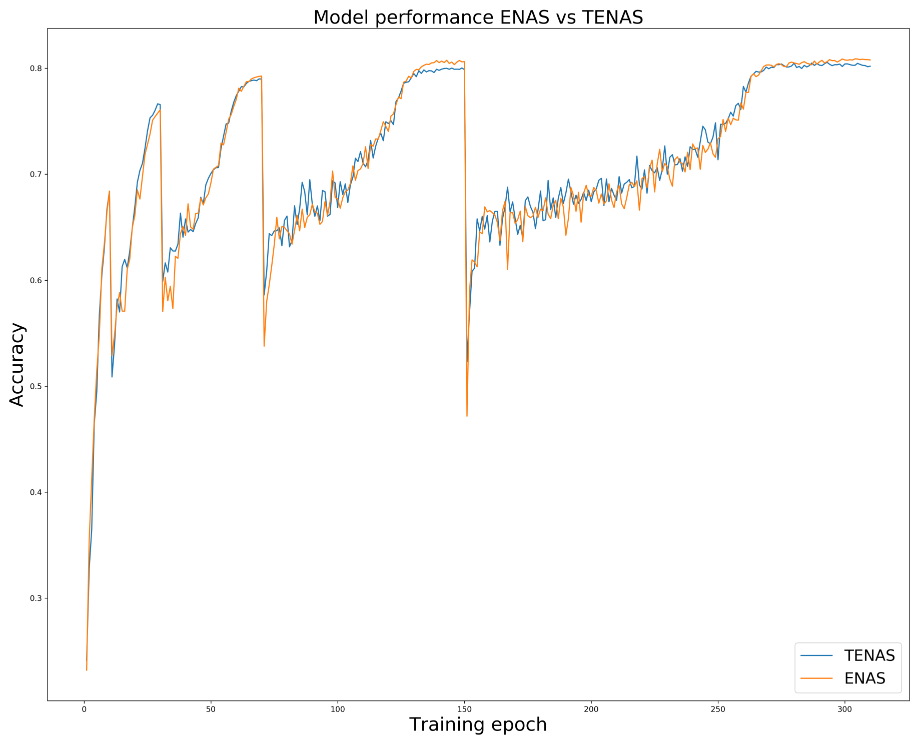


Figure 17. Learning curve of sampled architectures for both ENAS and transfer learning with ENAS for epoch 100

6.3 Performance estimation strategy evaluation

All architectures were trained from scratch for 310 epochs, which were sampled at epoch 155 with different validation accuracy and gave the same final accuracy on CIFAR-100. This raises some serious question about the validity of performance estimation strategy. In Table 3 we can see the results from this experiment, sampled epoch is the number of epoch controller was trained for and gave the following validation accuracy of the architectures.

In Figure 18 we take a snapshot of the macro search space controller for CIFAR-100 at training epoch 155 out of 310 and compare the validation accuracies (used for further controller training) of some generated architectures and test accuracies of the same architectures after having re-trained them. We infer from Figure 18 that there is no positive correlation between these two metrics, and thus we cannot expect that the validation accuracy with shared weights represents a useful training feedback for

Dataset	Search Space	Sampled epoch	Accuracy
CIFAR-10	macro	1	96.69 95.80 95.71
CIFAR-10	macro	310	95.38 95.81 95.76
CIFAR-100	macro	1	80.75 77.12 80.55
CIFAR-100	macro	310	80.39 80.07 80.47
CIFAR-100	micro	1	79.59 77.67
CIFAR-100	micro	310	80.50 80.02

Table 2. Performance of architectures trained by ENAS for CIFAR-10 and CIFAR-100. For the macro search space, both the was trained for 310 epochs. For the micro search space, the it was trained for 150 epochs. Before testing, the architectures were re- trained for 310 epochs. Table cells with multiple numbers correspond to multiple runs of the same experiment.

controller improvement.

Dataset	Sampled epoch	Validation accuracy	Final accuracy
CIFAR-100	155	41.41	80.33
CIFAR-100	155	32.81	81.11
CIFAR-100	155	17.97	80.81
CIFAR-100	155	21.09	80.50
CIFAR-100	155	28.12	81.12

Table 3. Results from the analysis of performance estimation strategy

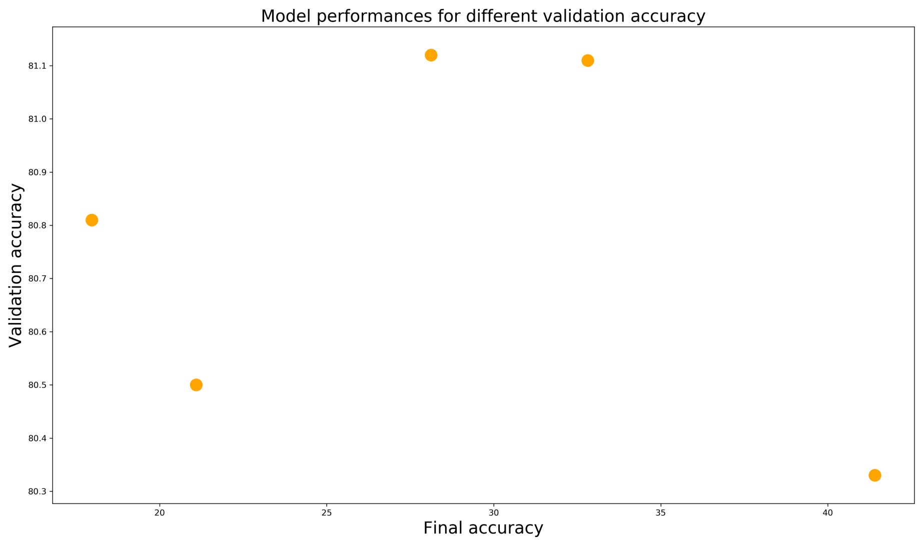


Figure 18. Scatter plot showing validation accuracy using the shared weights (x-axis) and the test accuracy of the architecture after re-training architecture for 310 episodes. Each point represents a neural architecture generated by the ENAS controller after having been trained for 155 episodes using the macro search space.

7 Conclusion

In this thesis we have analyzed the learning curve for ENAS controller, ENAS performance estimation strategy, and applied transfer learning to the ENAS controller. Our experimental results suggests to question the search strategy and performance strategy of ENAS and similar techniques.

We conclude this thesis in three sections, where we will evaluate ENAS search space, search strategy and performance estimation strategy.

1. **Analysis of Search space:** Our experimental results which are consistent with the findings by [LT19] indicate that the ENAS search space is the main contributor to its an impressive performance. Even an untrained controller can result in a good accuracy for given dataset. This performance can be attributed to a carefully crafted search space consisting of elements like convolution operations, pooling operation and skip connections. In the future work we will measure the performance of ENAS with more diverse elements like dilated convolution, highway networks etc

2. **Analysis of search strategy:** The ENAS search strategy turns out to be ineffective in finding better architectures. Our research questions the learning process of the controller itself, as after training for 310 epochs the controller does not sample architectures with better performance than the architectures sampled initially. Instead of finding an optimized architecture ENAS controller seems to overfit on the validation dataset and its shared parameters of DAG. From our research, we cannot conclude that a trained controller samples better architectures for CIFAR-10 and CIFAR-100 tasks. As described in Adam et al. [AL], ENAS controller can lead to biased sampling. As the ENAS training loop alternates between training shared parameters and then training the controller parameters, previously sampled architectures which are sampled again performs better because of improvement in shared parameters. This phenomenon leads to less diverse architectures as the ENAS controller will prefer the architectures which have more shared weights and less new weights or connections. This leads to a trained controller which samples less diverse architectures than a random controller. This can be noticed in the training for CIFAR-10 as the controller only samples convolution operations and stops sampling pooling operations.

3. **Analysis of performance estimation strategy:** The performance estimation strategy in ENAS proves to be ineffective for evaluation of architectures. As we observed in our experiments, almost all architectures with different validation accuracy gave the same final accuracy.This raises a question on the validity of the weight sharing based performance estimation strategies, as architectures which share more weights will be rewarded higher as compared to other diverse architectures. This can lead to the conclusion that weight sharing in general can lead to bias in architecture search, which will lead to sampling of similar operation again and again to increase its reward. A systematic study on the predictive power of the validation error using shared weights will shed more light on the effectiveness of this speedup-technique in general.

ENAS was a major improvement over NAS [ZL16], though our experiments and analysis provided us with some interesting insights about ENAS and what works and what does not. This analysis was necessary as ENAS and other techniques are now being used in critical areas like medicine and diagnostics [GS19].A systematic study on the predictive power of the validation error using shared weights will shed more light on the effectiveness of this speedup-technique in general. Our results motivate future studies of one-shot neural architecture design as a pragmatic and efficient alternative to architecture search.

Finally, the methodology applied in this work can be applied to evaluate the learning progress of other neural architecture search methods like DARTS [LSY18] , Neural architecture optimization [LTQ+18] and Progressive neural architecture search [LZS+17] to validate their performance and claims.

8 Acknowledgements

I am grateful to my supervisors Tobias Jacobs and Meelis Kull for their support and valuable suggestions. I would also like to thank Sebastian Nicolas and Mischa Schmidt for their constructive suggestions during the course of this thesis. I am thankful to NEC Labs Europe GmbH for financial and logistic support required for this thesis.

References

[AL] George Adam and Jonathan Lorraine. Understanding neural architecture search techniques. *CoRR*.

[ASP94] Peter J. Angeline, Gregory M. Saunders, and Jordan B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Trans. Neural Networks*, 5(1):54–65, 1994.

[BKZ$^+$18] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc V. Le. Understanding and simplifying one-shot architecture search. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 549–558, 2018.

[BLRW18] Andrew Brock, Theodore Lim, James M. Ritchie, and Nick Weston. SMASH: one-shot model architecture search through hypernetworks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[DDS$^+$09] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[EHH18] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural Architecture Search: A Survey. *arXiv e-prints*, page arXiv:1808.05377, Aug 2018.

[GHP07] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical Report 7694, California Institute of Technology, 2007.

[GS19] Nils Gessert and Alexander Schlaefer. Efficient neural architecture search on low-dimensional data for oct image segmentation. *arXiv preprint arXiv:1905.02590*, 2019.

[HKV18] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. *Automated Machine Learning: Methods, Systems, Challenges*. Springer, 2018. In press, available at http://automl.org/book.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[Kit90] Hiroaki Kitano. Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, 4(4), 1990.

[Kri09] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[LSY18] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. *CoRR*, abs/1806.09055, 2018.

[LT19] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. *CoRR*, abs/1902.07638, 2019.

[LTQ$^+$18] Renqian Luo, Fei Tian, Tao Qin, En-Hong Chen, and Tie-Yan Liu. Neural architecture optimization. In *Advances in neural information processing systems*, 2018.

[LZS$^+$17] Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan L. Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. *CoRR*, abs/1712.00559, 2017.

[MTH89] Geoffrey F. Miller, Peter M. Todd, and Shailesh U. Hegde. Designing neural networks using genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms, George Mason University, Fairfax, Virginia, USA, June 1989*, pages 379–384, 1989.

[PGZ$^+$18] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *ICML*, 2018.

[SSN17] Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. A genetic programming approach to designing convolutional neural network architectures. *CoRR*, abs/1704.00764, 2017.

[Wil92] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[YL96] Xin Yao and Yong Liu. Evolving artificial neural networks through evolutionary programming. In *Evolutionary Programming*, pages 257–266, 1996.

[ZL16] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[ZVSL17] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012, 2017.

Appendix

I. Glossary

1. LSTM: Long short-term memory
2. NAS: In thesis it’s referred to "Neural architecture search with reinforcement learning" [ZL16]
3. ENAS: Efficient neural architecture search via parameter sharing [PGZ+18]
4. AutoML: Automated machine learning
5. DAG: Directed acyclic graph

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Prabhant Singh**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,
   Analysis of Efficient Neural Architecture Search via Parameter Sharing
   supervised by **Tobias Jacobs and Meelis Kull.**

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Prabhant Singh
16.05.2019