ERA: Evolution of Residual Architectures

Samuel Lundberg

Master's thesis
2020:E35

LUND UNIVERSITY

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

ERA: Evolution of Residual Architectures

Samuel Lundberg
samuel.lundberg96@gmail.com

Supervisors at LTH: Kalle Åström, Luigi Nardi, Pontus Giselsson

Supervisor at Sentian: Johan Roos

Examinor: Mikael Nilsson

Abstract

This thesis investigates how well a neural architecture search can find competitive image classifiers on the CIFAR-10 data set with limited computational resources. Most work done on architecture search either uses vast computational resources or narrow and strongly informed space of possible solutions.

The solution space proposed consists of residual convolutional networks with separable convolutions and is explored by a primitive evolutionary procedure. When regularizing the evolution it proceeds to find an architecture outperforming the ResNet-110 baseline. Further evolution is shown to find networks with significantly lower average performance than random search. In comparison with Bayesian optimization and local search, evolution show competitive final performance. Bayesian optimization however gives unmatched sample efficiency.

One pressing matter in the process of reducing computations is how extensive networks must be trained before evaluation. Small scale experiments shows a tolerable rank-correlation between early performance and final performance of residual networks but indicates a bias for shallow networks.

Keywords: *Neural Architecture Search, Automated Machine Learning, Black-box Optimization, Residual Networks, Convolutional Networks, Image Classification, Neuro-Evolutionary Computing*

Contents

1 Introduction

1.1 Background

The use of machine learning, and artificial neural networks in particular, have been growing rapidly in the last five years. While deep neural networks have proven to be very capable at learning various tasks [18, 10] they often require extensive engineering to perform optimally. Primarily the user must design a neural topology suitable to the task. Then multiple hyper-parameters must be tuned in order to maximize the learning of the network. In all, this takes extensive work and often requires a specialist in the area.

The aim of automated machine learning is that the user simply introduce a task and a data set. Then the machine should find a complete machine learning pipeline that solves the problem. The fact that this process is computationally expensive could then be accepted as it would replace another time consuming task. For example an engineer could press a button on the machine and then work on other tasks for a few days until the pipeline was returned, instead of spending these days constructing the pipeline manually.

However designing a tool that finds solutions to every possible problem is a big task. This thesis will focus finding image classifiers for low resolution images. Deep convolutional networks [18] with residual connections [12] are proven to be top performing image classifiers [12, 1, 9, 31, 32]. This prior knowledge is used to limit what types of networks that will be considered.

To find suitable candidates among a large number of possible networks an evolutionary algorithm is implemented. Similar algorithms have proved to perform well in the architecture search setting given a large computational budget [28, 29]. Here it is also investigated whether they also perform well on a smaller computational budget. The software base used in this project already includes implementations for Bayesian optimization, local search and random search, used as baselines for the evolutionary procedure.

1.2 Problem Description

The following questions will be investigated in this work.

- Can an evolutionary architecture search with limited computational budget find architectures that outperforms known baselines such as ResNet-110?
- Can an evolutionary architecture search with limited computational budget outperform a random search and how does it compare to methods like Bayesian optimization and local search?
- How well does final performance of network correlate to early performance? If it does not, is bias given to specific types of architectures at early estimates?

2 Theory

2.1 Machine Learning

This thesis will mainly study deep learning, a specific kind of machine learning. Machine learning in turn is one of the most exploited sub-fields of Artificial Intelligence (AI) in resent years. It is necessary to get a grasp of the basic concepts of machine learning before diving into deep learning [10].

2.1.1 Basic Concepts of Machine Learning

In machine learning data is supplied to an algorithm from which the algorithm is supposed to *learn.* How do we define what it means to learn? A widely accepted definition of learning is the one presented by Mitchell et al. (1997) [22]:

*A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*

Now to explain the task, T, one distinction is necessary. The learning process alone is not the task. The learning process has the goal of being able to perform the task better then before. If, for example, the aim is for a robot to gain the ability of playing chess, the task would be playing chess, not learning to play chess [10].

The resent hype about machine learning can partly be credited to the rapidly rising number of tasks that can be solved. Common machine learning tasks are: classification where the algorithm should categorize inputs to one of k given classes; regression where the algorithm should predict a numerical value from an input value; anomaly detection where the algorithm finds atypical or unusual values in a series of data [10].

In order for the learning algorithm to know how to improve its performance on the task T. It is necessary to quantify a performance measure P that can evaluate the algorithm's ability on a the task. Also the performance measure P can help to visualize the progress of the learning [10].

The workflow is often to first make the machine learn and then perform the task on data it never has seen before. When measuring performance it is interesting how well it performs on the unseen data, not on the data it has been trained on. If the machine performs well on unseen data, it is said to generalize. As performance on seen and unseen data can differ a lot it is common to split the data set into two parts. One that the machine actually learns from and one that is purely dedicated to measure the performance after the training procedure. This to make the measure more representative of how well the machine would perform on the task in a real world setting [10].

An other aspect of performance measure is that people and learning algorithms does not want the same kind of information. When classifying images for example people are interested in whether it is right or not. A binary 1/0 measure, that after classifying many images, results in a percentage of correctly classified images. To a learning algorithm this is not interesting. Scoring 84% does not include information about how to get a higher score. The learning algorithm rather wants a continuous measure that it can differentiate. This to find a direction in the parameter space which would improve performance. While presenting the result in terms of accuracy the learning algorithms are often trained with log-probability as loss-function [10].

On to the experience E. When working with a experience in form of a fixed data set learning algorithms are divided into two general categories. Supervised learning, where the data is labeled, and unsupervised learning where no labels are available. Having labeled data means that if the data set contains images of animals it also contains information about what animal that is in each image. This obviously makes it easier for the machine to learn how to classify the images [10, 22].

2.1.2 Supervised Learning

In this work the focus will be on supervised learning. The data sets are of the type $[\mathbf{X}, \mathbf{y}]$ where one sample is $[X_i, y_i] \in [\mathbf{X}, \mathbf{y}]$. Here X_i can be an image and y_i would then be the label of which class X_i belongs to. In a different scenario where the task was to approximate a polynomial, X_i could be coordinates and y_i would be the function value. Notating the data set as $[\mathbf{X}, \mathbf{y}]$ is common for supervised learning independent of the task [10].

The learning process in supervised learning generally consists of making the learning algorithm find the relation between data points X_i and their labels y_i. This builds on the assumption that the samples $[X_i, y_i] \in [\mathbf{X}, \mathbf{y}]$ is independent and identically distributed random samples from an underlying distribution. And then the learning actually consists of identifying the underlying distribution of $\mathbf{X}$ [10].

2.1.3 Artificial Neural Networks and Deep Learning

Artificial neural networks are a type of machine learning algorithms that have become very popular in resent years [18, 10]. They are called artificial neural networks, or just neural networks, as they are inspired by how the neural system in the human brain works. The simplest form of a neural network is the perceptron model. This is, mathematically, a non-linear transform of a linear combination:

$$y = \varphi\big(\sum_{k=0}^{N} w_k x_k\big) = \varphi(\mathbf{wx}). \tag{1}$$

Here $\mathbf{x}$ and y are the input and the output of the model. The non-linear transform φ is called the activation function and $\mathbf{w}$ is the model parameters. The learning of a network consists of finding the optimal set of weights $\mathbf{w}$. As $\mathbf{wx}$ is a linear combination it is essential that φ is non-linear, otherwise the output would always be a linear combination of the input.

In deep neural networks these functions are stacked on each other. For example a two layer network with multiple outputs would look like:

$$y_i = \varphi^o\Big(\sum_{j=0}^{J} w_{ij}\, \varphi^h\big(\sum_{k=0}^{K} \tilde{w}_{jk} x_k\big)\Big).$$

The activation functions are denoted o and h referring to the output layer and the hidden layer. In a deep neural network all layers but the last one are called the hidden layers. Although neural networks consists of simple operations the networks get highly complex when the number of nodes and layers increases. Because of this, neural networks are considered to be black-box models [25].

2.1.4 Activation Functions

The activation function can be designed in many ways. Some traditional choices are;

the threshold function:

$$\varphi(x) = \begin{cases} 0 & if \;\; x < 0 \\ 1 & if \;\; x \geq 0 \end{cases}$$

the logistic function:

$$\varphi(x) = \frac{1}{1+e^{-x}}$$

and the linear rectifier function

$$\varphi(x) = \begin{cases} 0 & if \;\; x < 0 \\ x & if \;\; x \geq 0 \end{cases}.$$

The decision is up to the user but some are better suited to different problems and networks. For example the linear rectifier, ReLU, is a very common choise when using convolutional neural networks [25, 10].

2.1.5 Training a Neural Network

Training a neural neural network can be seen as an optimization problem. The problem is to minimize the networks error on a task with respect to the weights in a network. To do so gradient based methods are often used, this is called gradient descent learning [10, 25]. Algorithm 1 shows the basic steps of gradient descent learning where the half squared error is used as error function

$$E(w) = \delta^2/2, \;\; \delta = (y_{predicted} - y_{true})$$

$$\Delta_w E(w) = (y_{predicted} - y_{true})x = \delta x.$$

In the equation above x is the inner derivative when differentiation with respect to w, see equation 1.

Algorithm 1: Gradient descent learning

```
Randomly initiate the weights $w_k$.
Define the learning rate $\eta$.
Repeat until convergence
while True do
    Compute output: $y_{predicted} = y(x)$
    Compute the difference: $\delta = y_{predicted} - y_{true}$
    Update weights by stepping in the opposite gradient direction:
    $w = w - \eta\delta x$
end
```

This method always steps in the direction of most immediate improvement. When the error function is convex, as it is for many machine learning algorithms, this often works well. However for deep neural networks the error function can be non-convex. This means that finding a point w where $\Delta_w E(w) = 0$ is not equivalent to w being the optimal solution. In this case, standard gradient descent learning is not sufficient. Today there are a number of different approaches to overcome the difficulty of non-convex error functions [25]. Most of them are still designed around the gradient descent idea, for more details the reader is referred to other literature.

2.1.6 Convolutional Neural Networks

Convolutional neural networks [18] are a special kind of neural networks that are very popular in image analysis [25]. Instead of connecting each node in layer i with each node in layer $i+1$ using linear combinations, convolutional networks connect layers by performing convolutions between the input and a kernel [25].

A filter, or a kernel, is a matrix and the filtered output, y, of the layer is the convolution of the input, I, and the kernel, K:

$$y(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m,\; j-n)\, K(m,n).$$

See this example below of a convolution in one dimension:

$$[1,\ 2,\ 3,\ 4,\ 5] * [1,\ 0,\ 1] = [(1+3),\ (2+4),\ (3+5)] = [4,\ 6,\ 8].$$

The trainable weights in these networks are the kernels, in the example above there were three weights. While the network learns the kernels values it self their behavior can be explained mathematically to a certain extent. For the example above, if the kernel would be $[1/3,\ 1/3,\ 1/3]$ it would take the average of the nearby inputs and make the sequence smoother. If the kernel would be $[-1,\ 0,\ 1]$ it would differentiate around the middle point [25].

In a regular neural network each input was connected to each output. In these convolutional networks an output node only is connected to as many input nodes as there are entries in the filter. In the example above the difference is not to big, but imagine if the input was a image with 256×256 pixels and the filter was a 3×3 matrix. Then one output would be connected to only 9 out of the 65,536 pixels from the input. This is called sparse connectivity. Further a kernel slides over the entire input, all outputs are therefore computed using the same parameters. This is called parameter sharing [25].

Up til now a convolutional layer has been described as a single filter. In convolutional networks a convolutional layer usually has multiple filters an hence outputs multiple filtered maps. Further, a convolutional layer often includes three components: the filters, a pooling or batch normalization [14] layer and an activation function [25, 10]. There is much more to convolutional networks than described here, the interested reader is referred to [10].

2.1.7 Residual Neural Networks

For image classification tasks, using a residual structure [12] for the convolutional networks have proved to improve performance, especially for deeper networks. A residual connection, or just a skip connection, means that a layer transmits its output not only to its following layer but also to at least one other layer further forward in the network see figure 1. Then the function expressing the residual segment can be considered as $Y = F(X) + X$ rather than $Y = F(X)$ [12].

Three structures can be used to describe the residual networks, or just ResNets; sections, blocks and layers. A ResNet is consists of an initial feature extraction, a sequential chain of sections, a downsampling and an output layer [12]. The blocks contain convolutional layers and are the keys to the residual structures, as they are the only part including residual connections. To do so, the block passes its input both through the convolutional layers and straight to the output activation function, see the long bent arrows in figure 1. This figure shows two types of residual blocks, basic blocks and bottleneck blocks, which both where introduced in [12]. The section structure has the sole task of reducing the size of the feature maps, which is done in the first block of each section, except the first one. An interesting extreme case of the residual connections is the DenseNet [9]. In a DenseNet each convolutional layer transmits its output to all subsequent layers.

2.2 Neural Architecture Search

The main topic studied in this work is neural architecture search, a branch of automated machine learning. Automated machine learning is the process of automating the construction of good learning algorithms for a given problem. This so that the user can make good models without being an expert of the problem. Further the automated process might find learning algorithms beyond the human knowledge. Neural architecture search specifically regards how to design the model topology of a neural network, ex. how many and what type of layers to use.

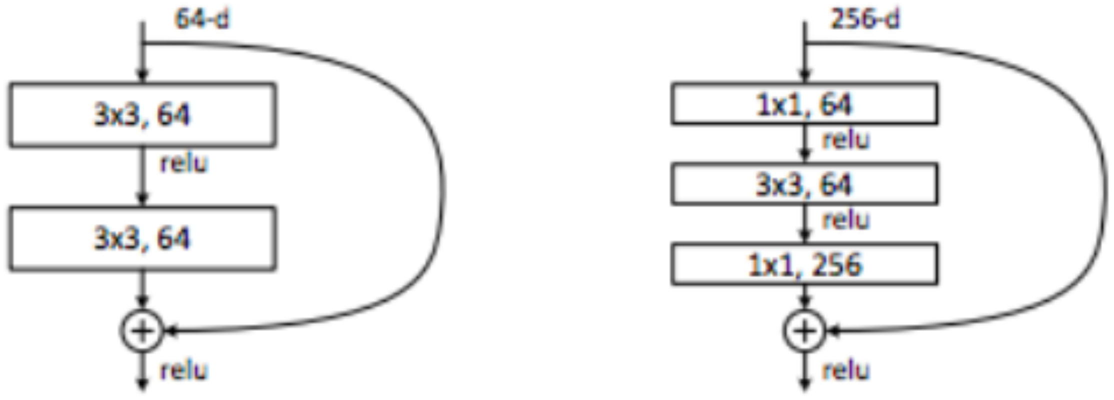


Figure 1: Left: A basic residual block which has two convolutional layers. Right: A residual bottleneck block which has three convolutional layers. Image from [12].

As this is a relatively new field there is not a many educative books describing neural architecture search and there are no general procedure for how to execute an architecture search, instead inspiration must be taken from recent research. This section will give a rather basic explanations of the sub tasks of an architecture search: defining a search space, searching the space and evaluating performance. More in depth explanations of how architecture searches can be conducted are described in section 2.3, the literature review. The workflow of the sub tasks are shown in figure 2.

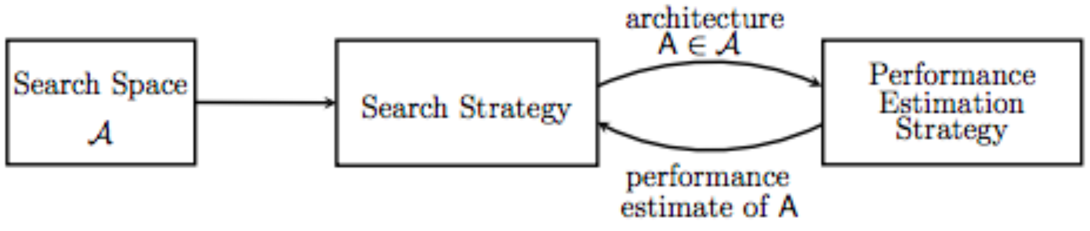


Figure 2: Here the simplified architecture search procedure is shown. Divided into the three sub tasks, image from [7].

2.2.1 Search Space

The first thing to do in an architecture search is to define a search space. This is the space of possible solutions. Defining the space can be done in many ways. The size of the search space can be reduced by prior knowledge from the user, while this simplifies the search it also increases the human bias. Reducing the search space might prevent the search from finding novel architectures beyond the human knowledge [7].

As the search space in practice must be limited, there will always be a bias. The question is rater how much bias the user choose to introduce.

2.2.2 Search Strategy

The Search strategy is how to navigate through the search space. Measuring the performance of a network requires training, which is time consuming. Hence we cannot use gradient based searches. Searching through the space can be considered a black-box optimization problem. Different approaches to search the space have been used in research and it is not clear what is the best way to do it. The methods have in common is that they must deal with the exploration-exploitation trade-off. On one hand you want to find good solutions fast, on the other hand you avoid local minimums [7].

2.2.3 Performance Evaluation

The objective of the architecture search is usually the same as in general machine learning: to find architectures that perform well on unseen data. However normally this is done by fully training a network and then evaluating it. This is time consuming and limits how many architectures that can be considered. Hence much work in the field now focus on how to measure performance more efficiently ex. weight inheritance and low-fidelity estimates [7].

2.3 Literature Review

2.3.1 Search Space

The basic way of building networks is to simply stack layers on top of each other. This approach is taken by Davison (2017) in DEvol, an open source software for architecture search and hyper-parameter optimization [3]. The objective in DEvol is always to maximize the performance om the MNIST data set [19], an image classification problem. Here the prior is strong, using the knowledge that convolutional networks are suitable for these tasks [18]. The search space only includes sequential networks that first have convolutional layers and end with fully connected layers. The user choose the maximum number of each type of layer as well as the maximum allowed width of the layers. In this way the user spans the search space under the given constraints. This approach manages to find networks with high classification accuracy, but MNIST is a relatively simple task.

State-of-the-art performance on more complex tasks is not reached with sequential networks though. To find networks with higher performance recent studies have incorporated more complex network structures [5, 6, 34, 29]. These non-sequential networks can for example have skip connections or divide networks into branches. Examples of manually designed non-sequential structures are residual networks [12] and dense networks [9].

A new paradigm, the cell-based search space, was suggested by Zoph et al. (2018). In this setting a cell is a non-sequential structure of layers. Then the search is divided into two separate processes. The inner part is to find good cells and the outer part is to stack the cells on top of each other. In this approach there are two different cells, the normal cell that preserves dimensionality of the input and the reduction cell that reduces the spacial dimension by a factor two. This leads to three searches, one for each cell and one for how to stack them [34].

The concept of simply stacking cells or blocks, of non-sequential layer structures, has a proven track record [12]. Using the cell-based search space gave new state-of-the-art performance [34] on the ImageNet data set [4]. Following this, other work have also had success using this cell-based search space [29, 6].

2.3.2 Search Strategy

Common approaches for navigating the search space includes Bayesian optimization, evolutionary algorithms, random search and reinforcement learning [7].

In a survey over the field of architecture search by Elsken et al. (2019b) a few papers stood out as they had moved the field forward with their work in some sense. Early work on evolutionary approaches was done by Real et al. (2017). The suggested algorithm is a very simple tournament selection procedure. The initial population consist of one layer models that grows into well performing architectures solely by mutating the most promising architectures in the population. They obtained competitive results on both CIFAR-10 and CIFAR-100. However this method is very computationally expensive and solutions might get stuck in local minimums [7, 28].

Further Real et al. (2019) combined the cell-based search space from Zoph et al. (2018) with the large scale evolution designed in Real et al. (2017). To make the evolution less greedy, the procedure is regularized by removing the oldest individual from the population instead of removing individuals based on performance. Argue that the regularization relates to how biological evolution works. However while showing that regularizing the evolution improves the performance over time the focus lies in comparing these evolutionary strategies to the reinforcement learning methods used by Zoph et al. (2018). The conclusion is that both evolution and reinforcement learning can outperform random search, albeit by a small margin. Evolution and reinforcement learning gave similar final results but evolution has better anytime performance and found less complex solutions. Using the regularized evolution they found a new architecture, AmoebaNet, that set a new state of the art performance on CIFAR-10 and ImageNet [29].

A simple hill-climbing procedure is used in network morphism by Elsken et al. (2018), a network morphs into a new one by taking a small step in a promising direction. The key to network morphism is efficiency in evaluation, which limits the options in the search phase. These limitation prohibits networks from reducing complexity. Constantly growing goes against the idea of simplicity and also makes the search very greedy and prone to get stuck in local minimums [5]. To improve upon network morphism Elsken et al. (2019a) introduced approximate network morphism. Here the key is the introduction of complexity reducing operations. This allows executing an evolutionary multi-objective architecture search that approximates the entire Pareto front [6].

Note that the evolutionary approaches used are rather simple, new networks are obtained by adding a mutation. Neither the large scale evolution in Real et al. (2017) and Real et al. (2019) or the morphism in Elsken et al. (2018) and Elsken et al. (2019a) manages to implement a successful cross-over mechanism although both mentioned it as an interesting extension to their work. As a mutation would represent a small step in a random direction these evolutionary algorithms are similar to a local search. In Elsken et al. (2018) the search is actually referred to as a hill climb, they do not consider the search but rather the morphism concept as the evolutionary aspect. The evolutionary resemblance strengthens in the following study Elsken et al. (2019a) as Lamarckin inheritance is introduced. In Real et al. (2017) the search is always referred to as an evolution as the population of individuals is always in focus. However initializing a random population and updating it by mutating promising individuals bears strong resemblance to a multi-start local search. To conclude, it is common to use evolutionary concepts in search strategies but not a full evolutionary procedure [28, 29, 5, 6].

An other way of conducting an architecture search is to consider it as a reinforcement learning problem. In the reinforcement learning setting the sampling of a neural architecture can be considered as an agent taking an action. In Zoph et al. (2018) a proximal policy optimization is implemented to search for architectures. The agents rewarding system is based on the neural architectures estimated performance on unseen data

[34].

2.3.3 Performance Estimation

One pressing issue in neural architecture search is the computational cost. Organizations like Google have the resources to train thousands of big networks but that is not the case for smaller companies and institutions. This is why effort is made to make these searches more efficient. An example of this is network morphism by Elsken et al. (2018). The concept is to start with a small network. Then a new network is obtained by adding complexity to the current one. Weight inheritance is used in such a manner that the new network is a function preserving update of the old one, $f_{new}(x) = f_{old}(x)$. Hence the new networks do not have to be trained from scratch, which significantly reduces computations [5].

However reducing complexity of networks cannot be done in a function preserving way. But Elsken et al. (2019a) introduces approximate network morphism. There complexity reducing operations are executed in a "almost" function preserving manner, $f_{new}(x) \approx f_{old}(x)$. In the context of evolution, (approximate) network morphism can be considered a Lamarckian inheritance as knowledge is passed on to new generations [6].

It is difficult to explore a vast search space with network morphism. A much more naive approach is taken by Real et al. (2017), Real et al. (2019) and Zoph et al. (2018) where the networks are trained from scratch, but only for 20-30 epochs. As a full training is over 150 epochs this reduces the computations with $\approx 80\%$. The issue is that the networks that performs best when evaluating after ≈ 20 epochs are assumed to be the best performers after the full training, but this assumption does not always hold.

When making an architecture search based on low-fidelity estimates it is desirable to evaluate the networks as fast as possible, and still finding architectures with strong final performance. To do so, inspiration can be taken from Li et al. (2018) where they speed-up random search for hyper-parameter optimization by using multi-fidelity estimates [20]. This means that they sample C random configurations and then train them a short time, T, and does successive halving iteratively until they have one configuration left. This idea assumes that rank-correlation holds reasonably well. Rank-correlation in this setting means that if neural network A performs better then neural network B after ex. 20% of the training it will, most likely, also perform better after the full training. This is the same assumption as in the last paragraph.

In Efficient Neural Architecture Search (ENAS) by Pham et al. (2018) the one-shot architecture search concept is explored. Here the entire space is designed as a directed asyclical graph which is trained on approximate gradients the beginning of the procedure using reinforcement learning. All architectures are subgraphs of the big graph, from which they inherit their weights, so no further training is needed. The search is then guided by a RNN controller that samples subgraphs. ENAS claims to be 1000 times more computationally efficient then standard architecture search but this concept arguably introduces more human bias than some of the previously mentioned work by setting up the search graph. Also ENAS proves to get competitive architectures on both tasks within computer vision, CIFAR10, and language technology, Penn Treebanks [21, 27]. The inherited weights does not give the architectures optimal performance. As models can be retrained after the search, as with the low-fidelity estimates, this is accepted. However this approach also rests on the assumption that rank-correlation holds.

2.3.4 Future Direction

The long term goal of automatized machine learning would be to remove the human bias and still be able machine learning pipelines with competitive performance. This however is far away from where the research is today. Attempts to do so is seen in very recent work by Real et al. (2020) called AutoML-Zero.

In this study the search space only consists of predefined mathematical operations. By an evolutionary search, stacking these mathematical operations, a two-layer neural network trained by backpropagation is discovered. Further these networks are then evolved directly on concrete tasks ex. binary classification variants of CIFAR-10. Doing so they can eventually outperform baseline method linear regression [30]. While this appears promising, it does not compete with state-of-the-art methods on specific tasks and the required computations are vast.

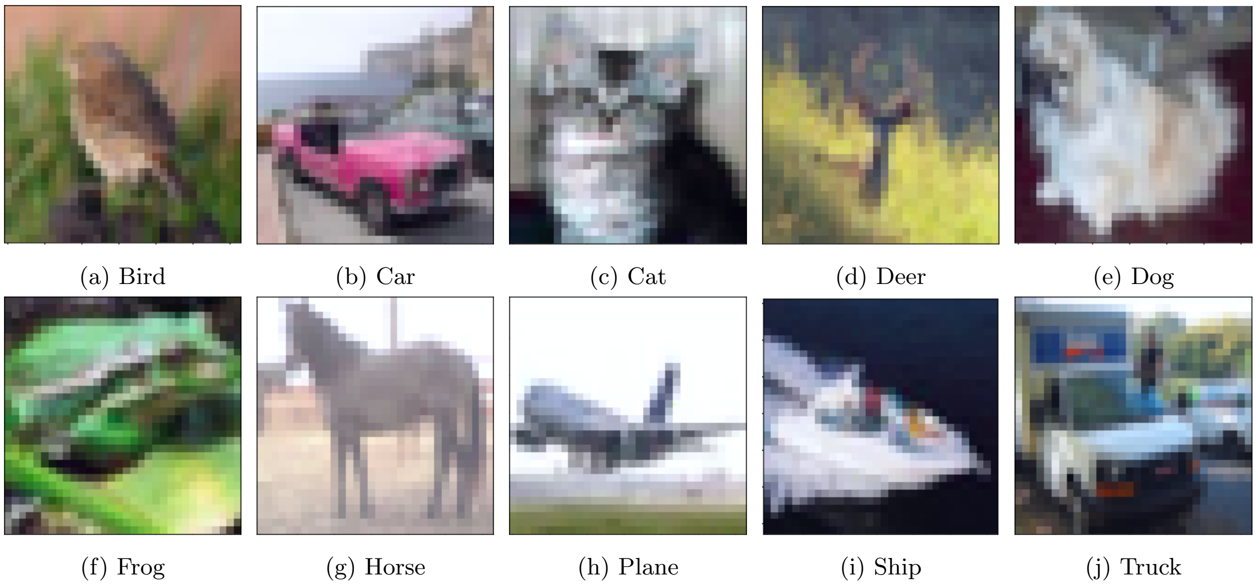
3 Methodology

Now let us go through the end-to-end construction of a full pipeline for a neural architecture search. First the problem and data used, as well as what hardware and software will be described. Then follows the technical procedure with details of how; the ResNet structure was used as a prior for the search space, the evolutionary algorithm was implemented and the assigned low-fidelity estimates of general performance.

3.1 Data

Image classification is one of the most mainstream tasks for neural networks to perform so there are many available data set. For the experiments the public data set CIFAR-10 [16] has been used. It consists of 6000 32×32 pixel images with motifs from ten different classes. Additionally each image has a corresponding label of which class it belongs to. The motifs are different animals and vehicles.

This data set is well used since its images are rater small, this makes computation on relatively fast compared to data sets of bigger images. Also it still has a considerable difficulty level. The best published networks [29, 34] have reached test accuracy of over 97% but finding those required extensive searches taking multiple GPU-weeks. For hand crafted networks, an accuracy of above 95% [11] is considered very good. See example of the CIFAR-10 images in figure 3.


(a) Bird (b) Car (c) Cat (d) Deer (e) Dog

(f) Frog (g) Horse (h) Plane (i) Ship (j) Truck

Figure 3: Random images from each class of the CIFAR-10 data set.

3.2 Software

The code in this project is written in python 3.6. For construction, training and evaluation of neural networks the PyTorch [26] library was used. The pytorch extension torchvision [2] was used for accessing of image data sets and preprocessing of data.

The code basis for this project comes from the multi-objective hyper-parameter optimization tool Hyper-Mapper [24, 23].

To design the residual networks in the search space torchvision's implementation [2] of which was used and also inspiration was taken from Train CIFAR10 with PyTorch by Kuang (2017) [17].

Early in the work inspiration was taken from DEvol - Deep Neural Network Evolution by Davison (2017) [3]. This is one of the few available software that uses evolutionary methods for architecture searches. While the study showed some promising results a more novel search space was desired for this project.

3.3 Hardware

Experiments was run on Sentian's servers in Malmö. For main experiments the stronger server, Frank, was used. Frank has 128 Giga byte RAM and is equipped with two 6 core Xeon E5/2620 v2 @ 2.10GHz CPUs and two Nvidia GTX 1080ti 12Gb GPUs. For smaller experiments the older server Rosenblatt was used. Rosenblatt has 48 Giga byte RAM and is equiped with two 4 core Xeon X5550 @ 2.67GHz CPUs, two Nvidia GTX 1050ti 4Gb GPUs and further two two Nvidia GTX 1650 4Gb GPUs.

3.4 Design of the Search Space

This project focuses solely on image classification for relatively small ($\approx 500 - 1600$ pixels) images. The search space was constructed around the ResNet architectures designed for CIFAR10 in He et al. (2016), which are proven to perform well [12].

The key implementation wise was to construct a mapping from parameter configurations, representing a network, to an actually neural network object in pytorch. To do so a direct encoding was designed what builds complete residual networks guided by eight parameters with predefined possible values.

In some previous work the search procedure consists of both finding blocks and then stacking them [34, 29]. This study is limited to stacking pre-defined blocks. For this, the basic block and the bottleneck block defined in [12] are used. They can be seen in figure 1. A network will have only basic blocks or only bottleneck blocks, not a combination of them. This is the first parameter in the constructed search space.

Further the networks have four sections, and there are one integer parameter representing each one of them. The first section can have zero blocks and the others have at least one block, so in practice the networks will have either three or four sections. The maximum number of blocks in a section is 18 for all sections. The flexibility of having either three or four sections in the networks is due to that both have been proven successful. The standard ResNets have four sections [12] and using ResNets with four section on CIFAR-10 can give good results [17]. However on CIFAR-10 the original study successfully use ResNets with only three sections [12].

This means that our search space contains networks with $b \in [3, 72]$ blocks. The basic block have two weighted layers while the bottleneck block has three layers. Outside of the section the networks will have two weighted layers, the initial feature extraction and the output layer. So in total the possible networks in the search space will have $l \in [8, 218]$ layers.

The width is controlled by two parameters. 1) The number of filters, this decides the number of filters the convolutions in the first section will output. This parameter is ordinal and can take the values 2^i, $i \in [0, 5]$. 2) The filter update, this decide by what factor the number of filters will increase between the sections. The possible values are $\{1, 2\}$. Normally ResNets increase the number of filters with the factor two.

To enable cardinality, see figure 6 in section A.1, in the networks the last parameter is called group size. The group size sets how many filters each group will have, hence the networks cardinality will be the number of filters divided by the group size. Group size can take the values 2^i, $i \in \{0, 1, 2, 3, 5\}$. To clarify, if the number of feature maps increases between the sections the size of the groups will grow but the cardinality will be constant.

As all parameter values can be freely combined we will have 13,296,960 possible configurations. However problems occur when the group size is bigger then the number of filters. Then the group size is set to be the number of filters and networks become duplicates. So the search space only contains 8,864,640 unique networks.

3.5 Searching the Space

To search this space an evolutionary algorithm, inspired by Real et al. (2017) [28], was implemented. The procedure is described in algorithm 2. The mutation function alters a given configuration by randomly assigning a new value to one of its parameters. The new value is drawn from a uniform distribution of the possible parameter values.

Algorithm 2: Evolution

```
Set values for population size $P$, number of generations $G$.
Through the procedure $P + G$ configurations will be evaluated.
Initiate the population pop by drawing $P$ random samples from the search space.
for g in range(G) do
    draw two random individuals, $A$ and $B$ from pop;
    if $A \succ B$ then
        $C = A$;
        remove B form pop;
    else
        $C = B$;
        remove A form pop;
    end
    mutate(C);
    add C to pop;
end
```

Two slightly modified versions of the algorithm was also enabled. The first modification was to regularize the evolution as in Real et al. (2019) [29]. By regularizing, the oldest individual in the population is removed after each generation. It is not based on performance as in algorithm 2. The second modification was to both regularize the evolution and also add a crossover mechanism. When using the crossover mechanism, three individuals is drawn from the population instead of two. Further the child, C, is obtained by randomly combining the two best picked individuals, still the mutation is applied.

The evolutionary approaches will be compared to the search algorithms available in HyperMapper. These are Bayesian optimization, local search and random search. The first two algorithms are explained in section A.2.

3.6 Performance Estimation

For the search to be meaningful, performance of the configurations must be measured sufficiently. As training a network to convergence is time consuming, the networks was evaluated after just 20 epochs of training. The choice of 20 epochs was inspired by similar works where networks were trained for between 20 and 30 epochs [28, 29, 34].

Pre-processing of data and environment for training the networks are the same as when training ResNets on CIFAR-10 in [12]. Training details can be read in section A.3.

4 Results

4.1 Architecture Searches

As described neural architecture searches was conducted with existing algorithms random search, local search and Bayesian optimization as well as with three variants of the evolutionary algorithm. At the end of the day, the most interesting aspect of an architecture search is the general performance of its best found architecture after a complete training. To measure this the best found architecture from each search will be compared to each other as well as the baseline architecture the search space was built around, the ResNet-110. In this process each network was trained identically.

However the algorithms can only be held responsible for the performance of the networks in the search environment, where they are only trained for 20 epochs. In an effort to understand the algorithms the performance of each evaluated network will be analyzed. To further get a grip of how well the low-fidelity estimates in the search correlates to high-fidelity estimates, a rank-correlation study of residual networks rounds of this section.

4.1.1 Final Results

In table 1 the performance of the best network from each architecture search is shown along with the depth and number of trainable parameters in the networks. The search space includes networks with over 200 layers but each architecture search finds networks with less then 100 layers. While the found architectures have less layers then the ResNet-110 baseline they are not necessarily smaller or less complex. On the contrary, the three architectures that outperforms ResNet-110, Bayesian optimization, evolution and regularized evolution, have significantly more trainable parameters then the baseline has. For this reason a wider version of ResNet-110 was included as comparison, and this performed slightly better than its shallow counterpart.

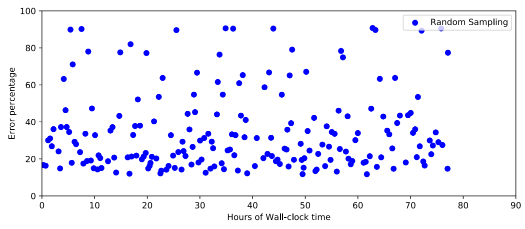
Table 1: CIFAR-10 classification results after full training for the best architectures found by each search. Compared to the baseline architectures.

Model/Origin	# layers	# params	Error (%)
ResNet (baseline)	110	1.7M	7.5
ResNet wide	110	6.9M	7.1
Random Search	47	0.92M	8.5
Local Search	74	1.4M	8.4
Bayesian Optimization	52	12.3M	7.3
Evolution	41	6.8M	7.0
Regularized Evolution	80	15.0M	**6.7**
Crossover Evolution	36	6.6M	8.5

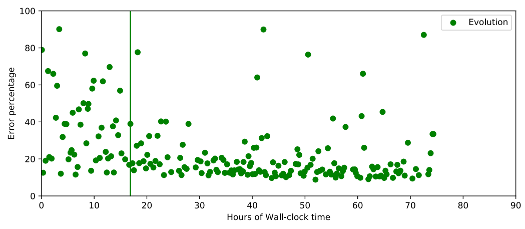
4.1.2 Search Analysis

Previous results have shown that well performing architectures can be found. However only evaluating the best found solution tells little about the search over all. In figure 4 the performance of each evaluated sample for each algorithm is seen. This shows that, after the random sampling phase, the guided searches finds fewer solutions with very high error compared to the random search.

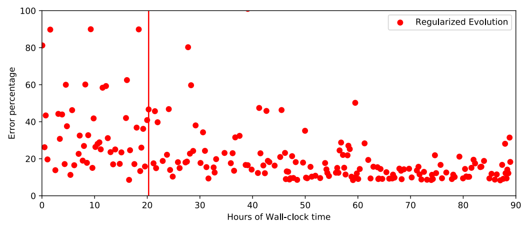
More visualizations of the architecture searches can be seen in figure 5. Subfigure 5a shows that Bayesian optimization finds the best architecture as it slightly outperforms evolution and regularized evolution. Out of the evolutionary methods regularized evolution finds the best solution. But it finds significantly better



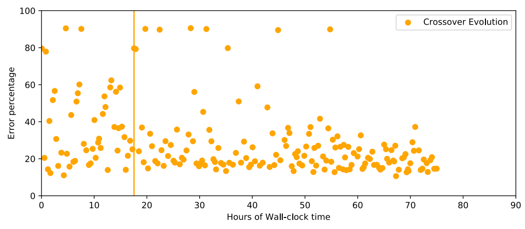
(a) Random Sampling scatter



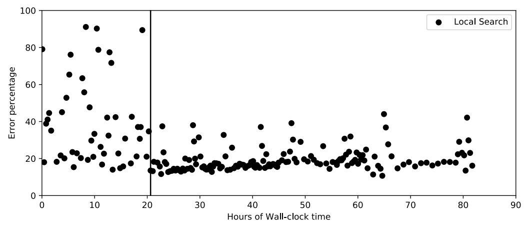
(b) Evolution



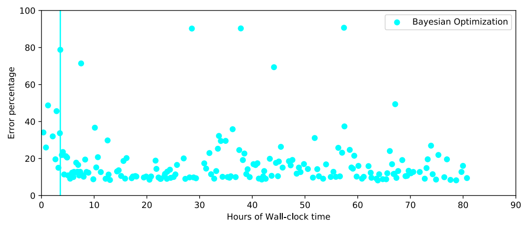
(c) Regularized Evolution



(d) Crossover Evolution



(e) Local Search



(f) Bayesian Optimization

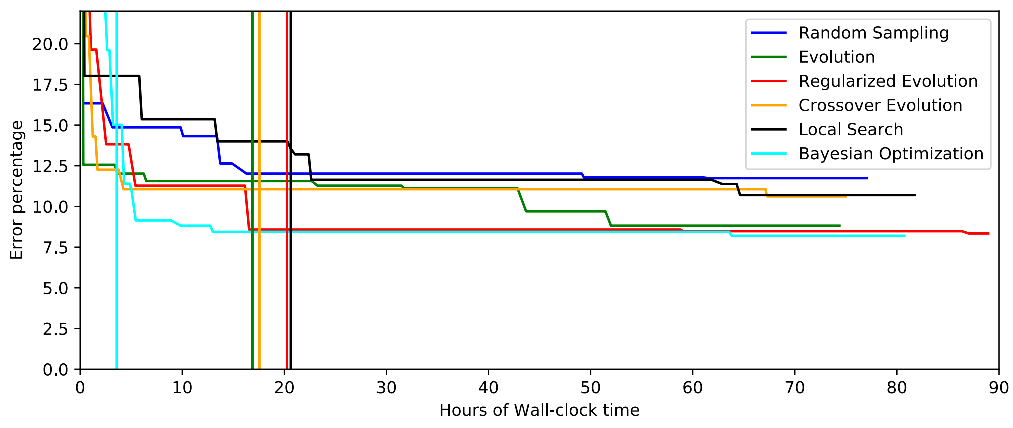
Figure 4: Error percentage for all evaluated network for each search as a function of real elapsed timed. The networks are evaluated after 20 epochs of training.

solutions during the random sampling phase than the other searches do. All guided searches beats random search.

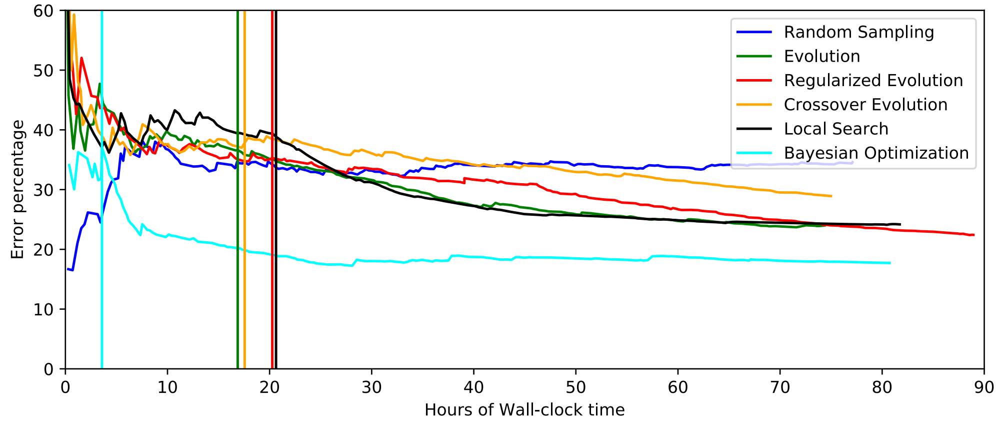
Perhaps the sub figures 5b and 5c are more telling. That the guided searches on average finds better solutions are shown in 5b and 5c tells that the difference is bigger between the guided searches and random search in the later parts of the procedure. All the evolutionary approaches and local search behaves quite similarly while Bayesian optimization has completely different behavior. Early in the process Bayesian optimization significantly outperforms the other methods in average sample performance. However figure 5c indicates that the other algorithms do catch up. It must also be mentioned that while all other algorithms evaluated 200 samples the Bayesian optimization search only evaluated 181 samples. This is due to the fact that Bayesian optimization requires more computations to pick a new sample, so a time limit of 80 hours was put upon that procedure.

The local search algorithm used as benchmark behaves very similar to the basic evolution in particular, see figures 5b and 5c. However comparing figure 4b to figure 4e shows that local search has less volatile performance. This can be the reason to why its best found architectures is slightly worse. Also it finds worse architectures in the random sampling phase, which also could hinder its performance.

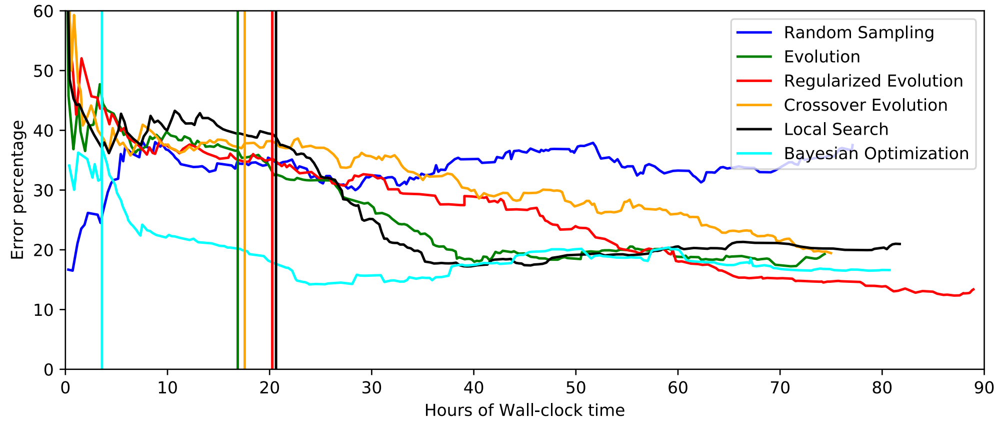
The ordering of the best found architectures is visualized in figure 5a. However the actual performance is easier to quantify by studying table 2. This table looks similar to table 1 from the last section, but now the



(a) Error percentage of the best evaluated network.



(b) Average error percentage of all evaluated networks.



(c) Average error percentage of 50 most recently evaluated networks.

Figure 5: Measures for comparing the different search methods. All networks here were trained for 20 epochs.

scores comes from the low-fidelity estimates when only training the networks for 20 epochs in the architecture search environment.

Worth noting is that Bayesian optimization found the architecture with the smallest error in the search, see table 2. But after full training it only ranks as third best, see table 1. Also it is interesting that while the wide ResNet-110 had a strong final performance it shows a much higher error than the other architectures after 20 epochs, see table 2. These behaviours motivates the following section about rank-correlation.

Table 2: CIFAR-10 classification validation results after 20 epochs of training in the architecture search environment.

Model/Origin	# layers	# params	Error (%)
ResNet (baseline)	110	1.7M	10.4
ResNet wide	110	6.9M	19.9
Random Search	47	0.92M	11.7
Local Search	74	1.4M	10.7
Bayesian Optimization	52	12.3M	**8.2**
Evolution	41	6.8M	8.8
Regularized Evolution	80	15.0M	8.3
Crossover Evolution	36	6.6M	10.6

4.2 Rank-Correlation Experiments

This section analyses anytime performance of ResNets with two different experiments. One where only the depth is varied and one where only the width is varied. The purpose of this is to get a feeling for how biased the architecture search might be. And in general what to expect if comparing performance prematurely. For these experiments networks are validated at 5 different fidelity defined by amounts of data processed in training. The levels used are 5, 10, 20, 50 and 100 epochs on the CIFAR-10 training data.

4.2.1 Width Comparison

In table 3 the performance at the different fidelity is shown for four versions of ResNet-56, meaning that they all have 56 layers of trainable parameters. The regular version with 16 convolutional kernels in the first section as well as variants with 8, 32 and 64 filters respectively. So to be clear, the only difference between these networks are their width. There are clear indications that estimating very early biases networks with less width as the smaller networks dominates at 5 epochs while the wider networks perform best at 100 epochs.

Table 3: Comparing multi-fidelity performance of ResNets of depth 56 with varying width. Performance measured in classification accuracy on CIFAR-10 test data.

Epochs \ Filters	8	16	32	64
5	**72.7**	70.7	56.0	45.3
10	77.6	**79.5**	78.0	63.8
20	81.9	85.0	**86.3**	80.3
50	85.2	87.1	**88.8**	87.5
100	89.1	90.7	**92.3**	92.0

The Rank-Correlation is quantified in table 4. The entry at index (i, j) shows how much the rank of the networks at fidelity i correlates with the rank of the networks at fidelity j, hence the matrix is symmetrical with ones on the diagonal. As the architecture searches in this report evaluated networks at 20 epochs, a high rank-correlation between 20 and 100 epochs is desired. Here it is actually negative, which would suggest that measuring performance at 20 epochs is rather wasteful. However the network with 64 filters in the same group is not included in the search space as it is to wide. And if that network was removed from the computations, the rank-correlation would be 1 between 20 and 100, see table 5.

Table 4: Rank-Correlation for the results in table 3.

Epochs	5	10	20	50	100
5	1	0.4	0.2	-0.8	-1
10	0.4	1	0.8	0	-0.4
20	0.2	0.8	1	0.4	-0.2
50	-0.8	0	0.4	1	0.8
100	-1	-0.4	-0.2	0.8	1

Table 5: Rank-Correlation for the networks in table 3 that fits in the search space.

Epochs	20	100
20	1	1
100	1	1

As this is a very small sample size it would be hazardous to draw to strong conclusions from rank-correlation results. But it indicates that the architecture searches on the search space used in this work should not be limited by bias for thinner networks.

4.2.2 Depth Comparison

Fixating the width to 16 filters, as in He et al. (2016) [12], and instead varying the depth gave similar results, see table 6. Smaller networks, in case less deep, learns faster and performs the best at 5 and 10 epochs. But after 100 epochs the deep networks are the strongest performers. The rightmost column showing ResNet-602 performs as an outlier but this network is very deep and way outside the search space. This is merely included to give perspective.

Table 6: Comparing multi-fidelity performance of ResNets with 16 initial filters and varying depth. Performance measured in classification accuracy on CIFAR-10 test data.

Epochs \ Layers	14	32	56	110	602
5	**75.9**	75.0	70.7	49.3	10.0
10	80.4	**80.9**	79.5	71.4	10.0
20	82.4	**85.1**	85.0	83.5	10.0
50	84.0	86.1	**87.1**	86.3	76.4
100	87.9	89.9	90.7	**91.1**	89.8

Table 7 shows that the rank at fidelity 100 is positively but not totally correlated with fidelity 20 and 50. ResNet-110 performs the best after 100 epochs, and is proven to outperform its shallower counter parts in [12]. However ResNet-110 is not ranked first at any other fidelity. Measuring rank-correlation excluding the

ResNet-602 in table 8 shows that performance after 20 and 100 epochs barely is positively correlated. This could be an indication that the architecture search could struggle in finding deep networks that would have strong final performance.

Table 7: Rank-Correlation for the results in table 6.

Epochs	5	10	20	50	100
5	1	0.9	0.4	0.1	-0.4
10	0.9	1	0.7	0.2	-0.2
20	0.4	0.7	1	0.7	0.5
50	0.1	0.2	0.7	1	0.8
100	-0.4	-0.2	0.5	0.8	1

Table 8: Rank-Correlation for the networks in table 6 that fits in the search space.

Epochs	20	100
20	1	0.2
100	0.2	1

4.2.3 Combined Comparison

Combining the networks from the studies of bias for depth and width, but excluding the networks that was not parts of the search space, gives the rank-correlation shown in table 9. This measure still only includes 6 networks out of 13 million in the search space. But it shows a rank-correlation of 0.71 between fidelity 20 and 100, which can be considered a rather strong correlation. Further, training for less then 20 epochs in the search appears to be rather pointless. The networks ranked by performance after 10 epochs is actually negatively correlated with how they are ranked after 100 epochs.

Table 9: Rank-Correlation for the results in table 3 and 6 excluding networks outside of the search space.

Epochs	5	10	20	50	100
5	1	0.77	-0.31	-0.77	-0.89
10	0.77	1	0.31	-0.26	-0.43
20	-0.31	0.31	1	0.77	0.71
50	-0.77	-0.26	0.77	1	0.94
100	-0.89	-0.43	0.71	0.94	1

5 Discussion

5.1 Search Results

After the search space was constructed around ResNet-110 and included variants of it with increased complexity in most dimensions it was expected that better performing networks would exist. However it was unclear if the architecture searches would be able to find such solutions. Implementations similar to the evolution used in this work had proven successful previously [28, 29]. But then applied to a more vast search spaces and using computational resources on a completely different scale.

The results in table 1 however showed that the regularized evolution and the basic evolution both found architectures that beat ResNet-110, as did Bayesian optimization. But all searches did not and it is not guaranteed that the results would reproduce.

The regularized evolution found an architecture with lower error then random sampling and the other evolutionary methods in the search environment, as it did in [29]. Also figure 5c strengthens the hypothesis of regularized evolution being less greedy then the regular evolution but having stronger final performance. But after only running the algorithms once, and not allowing them to evaluate many samples, it would be hazardous to assume the results would reproduce that behavior consistently. In this discussion it is also worth pointing at figure 5a. This shows that regularized evolution found an architecture that significantly outperforms (8.6 % error) the architectures the other algorithms draws in their random sampling phases. Actually this architecture performed stronger then anything the other algorithms, bar Bayesian optimization, ever found. And the best architecture found by the regularized evolution shares four out of eight parameter values with the strong performing random sample. So it is possible that the random sampling phase is as important as the search procedure it self. If all algorithms where ran again it could be possible for random search to find the best architecture.

The same argument can be made to defend the local search which was expected to perform better. The local search did not really find any good architectures in its random sampling phase. And then its procedure consists of searching close to best found random samples. This is bound to fail if the random samples are not good enough. The mutation function driving the evolutionary procedures assigns, uniformly distributed, random values to a parameter. This allows for it to take bigger steps then the local search can do. Because of this, maybe the random sampling phase for the local search should have been longer.

While local search and evolution behaves similarly when comparing average performance in 5b and 5c they behave significantly different in sample to sample perspective seen in figures 4b and 4e. It is much less deviation in the performance of the solutions from the local search. If the average is similar for both methods, evolution could find the strongest solution by being more volatile.

Bayesian optimization proves to be much more sample efficient then the other methods. As it only takes nine (number of dimensions + 1), random samples and then starts to consistently find good architectures straight away. Looking at figure 5, Bayesian optimization is clearly the preferable choice of algorithm if the time limit was 48 hours or less, given the used hyperparameters. But that is, at least partly, due to the other algorithms taking many more random samples. In the later parts of the searches the other methods eventually catch up as seen in figure 5c. I might be the case that the preferable choice of algorithm depends on for how long the search is allowed to run. Further discussions on algorithmic sample efficiency is out of the scope for this thesis, anytime performance comparison of random sampling and Bayesian optimization can be read in [8].

It is not by looking at the final performance but by looking at the figures 5b and 5c that the guided algorithms shows that they outperform random search. When the basic evolutionary procedure ends, after around 75 hours, the average performance of all evaluated networks are almost identical for both the evolutionary methods without crossover and for local search. Outperforming random search with a clear margin.

5.2 Rank-Correlation

As shown in both table 1 and 2 the deepest network any of the searches found had 80 layers. This is just over a third of the number of layers the search space allows. Further this network has the bottleneck block structure, see details of the best found architecture from each search in table 10 and 11, which has three layers, compared to the ResNet-110 that has basic blocks of just two layers. So counting blocks ResNet-110 has 54 blocks, more than twice as much at the 26 blocks in the 80-layer network. An other interesting aspect is that the 80-layer network performs the best after 20 epochs while being the deepest, as table 6 showed a bias for shallower networks. On the other hand, table 3 showed that wide networks, within the search space, outperformed their shallower counterparts already at 20 epochs. And the 80-layer network has 15 million weight, more than all of the other architectures, indicating that it probably is wide. And further it can be assumed to have a very high potential to learn complex patterns.

Looking at table 2 the performance for the networks from the six architecture searches is largely correlated with their number of weights. This would indicate that large networks are preferable. But then, why do they not find even bigger networks? There are networks in the search space with over 80 million weights.

In section A.4 table 10 and 11 shows the parameter values of the best architecture each search finds. Again as the there is no guarantied significance in the results it is dangerous to draw to strong conclusions from the found architectures. However there are clear indications that wide networks are preferable. All but one of the architectures have 64 filters which is the highest allowed number. Increasing the number of filters between the sections also seem to be successful as most do so to. The number of blocks in the sections varies a lot. There the only emerging pattern is that few sections have ten or more blocks. Only three out of the six networks have such a section. If randomly drawing a network from the search space the probability of it to have 10 or more blocks in at least one section is over 90%. This strengthen the hypothesis that evaluating after short training prohibits the search from finding deep networks.

5.3 Training Networks to Optimal Performance

In table 1 there was an attempt to train the networks to optimal performance. There, the final performance of the networks found by the architecture searches was compared to each other, and the ResNet-110 baseline. All networks was trained in the same manner and long enough to converge, so the comparison is considered fair. However the achieved performances was not quite as good as expected. While it is not known how well the architectures from the searches could perform the ResNet-110 performs a classification error of less then 7% consistently in [12]. In the almost identical training environment used for training in this thesis it got a error of 7.5%. The difference does not sound huge but on CIFAR-10 1% is a lot. Now the network found by regularized evolution performs comparable to the best architectures in [12]. If its error could be decreased by 1% it would look like a clear improvement upon the architectures proposed in [12].

5.4 Answering of Research Questioning

- Can an evolutionary architecture search with limited budget find architectures that outperforms known baselines as the ResNet-110?

  Yes, it certainly can. But it is not guaranteed to do so.

- Can an evolutionary architecture search with limited budget outperform a random search and how does it compare to methods like Bayesian optimization and Local Search?

  The results indicates that the evolutionary approach is preferable to random search, but it is not guarantied to perform better.

  The results also suggest that evolution competes well with local search and Bayesian optimization, but Bayesian optimization was more sample efficient in the conducted experiments. Although all methods might improve with a hyper-parameter tuning.

- How well does final performance of network correlate to early performance? If it does not, is bias given to specific types of architectures at early estimates?

  Performance after 20 epochs appear to correlate quite well with final performance. However results suggest that the short training gives a disadvantage to deeper networks that might prohibit us from finding even better architectures.

References

[1] Y. Chen, J. Li, H. Xiao, X. Jin, S. Yan, and J. Feng. Dual path networks. In *Advances in neural information processing systems*, pages 4467–4475, 2017.

[2] S. Chintala. Torchvision source code. `https://github.com/pytorch/vision`, 2016.

[3] J. Davison. Devol - deep neural network evolution. `https://github.com/joeddav/devol`, 2017.

[4] 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.

[5] T. Elsken, J.-H. Metzen, and F. Hutter. Simple And Efficient Architecture Search for Convolutional Neural Networks. *ICLR*, 2018. doi: https://arxiv.org/abs/1711.04528.

[6] T. Elsken, J. H. Metzen, and F. Hutter. Efficient Multi-objective Neural Architecture Search via Lamarckian Evolution. *ICLR*, 2019. doi: https://arxiv.org/abs/1804.09081.

[7] T. Elsken, J. H. Metzen, and F. Hutter. Neural Architecture Search: A Survey. *Journal of Machine Learning Research 20*, 2019. doi: https://arxiv.org/abs/1808.05377.

[8] S. Falkner, A. Klein, and F. Hutter. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. *ICML*, 2018. doi: https://arxiv.org/abs/1807.01774.

[9] K. Q. W. Gao Huang, Zhuang Liu. Densely Connected Convolutional Networks. *CVPR 2017*, 2018. doi: https://arxiv.org/abs/1608.06993.

[10] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[11] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. *CVPR*, 2016. doi: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/He_Deep_Residual_Learning_\\CVPR_2016_paper.pdf.

[13] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, 2014.

[14] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[15] J. Knowles. Parego: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1):50–66, 2006.

[16] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[17] L. Kuang. Train cifar10 with pytorch. `https://github.com/kuangliu/pytorch-cifar`, 2017.

[18] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time-series. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*. MIT Press, 1995.

[19] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[20] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *JMLR*, 2018. doi: https://arxiv.org/abs/1603.06560.

[21] M. Marcus, G. Kim, M. A. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger. The penn treebank: annotating predicate argument structure. In *Proceedings of the workshop on Human Language Technology*, pages 114–119. Association for Computational Linguistics, 1994.

[22] T. Mitchell and M. Hill. *Machine Learning*. McGraw-Hill, 1997. `http://profsite.um.ac.ir/~monsefi/machine-\\learning/pdf/Machine-Learning-Tom-Mitchell.pdf`.

[23] L. Nardi and A. Souza. Hypermapper. `https://github.com/luinardi/hypermapper`, 2019.

[24] L. Nardi, D. Koeplinger, and K. Olukotun. Practical design space exploration. In *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 347–358. IEEE, 2019.

[25] M. Ohlsson and P. Edén. Lecture Notes on Introduction to Artificial Neural Networks and Deep Learning. *Lund University*, 2019.

[26] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-\\performance-deep-learning-library.pdf`.

[27] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient Neural Architecture Search via Parameter Sharing. *CoRR*, 2018. doi: https://arxiv.org/abs/1802.03268.

[28] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, and A. Kurakin. Large-Scale Evolution of Image Classifiers. *ICML*, 2017. doi: https://arxiv.org/abs/1703.01041.

[29] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized Evolution for Image Classifier Architecture Search. *AAAI*, 2019. doi: https://arxiv.org/abs/1802.01548v7.

[30] E. Real, C. Liang, D. R. So, and Q. V. Le. AutoML-Zero: Evolving Machine Learning Algorithms From Scratch. *arXiv preprint*, 2020. doi: https://arxiv.org/abs/2003.03384.

[31] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[32] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[33] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated Residual Transformations for Deep Neural Networks. *CVPR*, 2017. doi: https://arxiv.org/abs/1611.05431.

[34] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Leraning transferable architectures for scalabe image recognition. *CVPR*, 2018. doi: https://arxiv.org/abs/1707.07012v4.

A Appendix

A.1 Cardinality

Cardinality is considered to be the third dimension of a CNN [33], depth and width being the first two. This concept is sometimes referred to as separable convolutions or grouped convolutions. The idea is to make the convolutional kernels sparsely connected. In regular convolutional neural networks the cardinality is one. But if, for example, the cardinality would be two, half the filters would act on half the input feature maps. This is equivalent to having two parallels convolutional layers each having half the number of filters and each getting half of the input feature maps. For visualization see figure 6. Increasing cardinality is show to improve performance under the restrictions of maintaining complexity and also, when increasing complexity, increasing cardinality is shown to be more efficient then increasing depth or width [33]. On the original ResNets, increasing cardinality is only used together with bottleneck blocks [33] [2]. But for this thesis it will be allowed for the basic blocks as well.

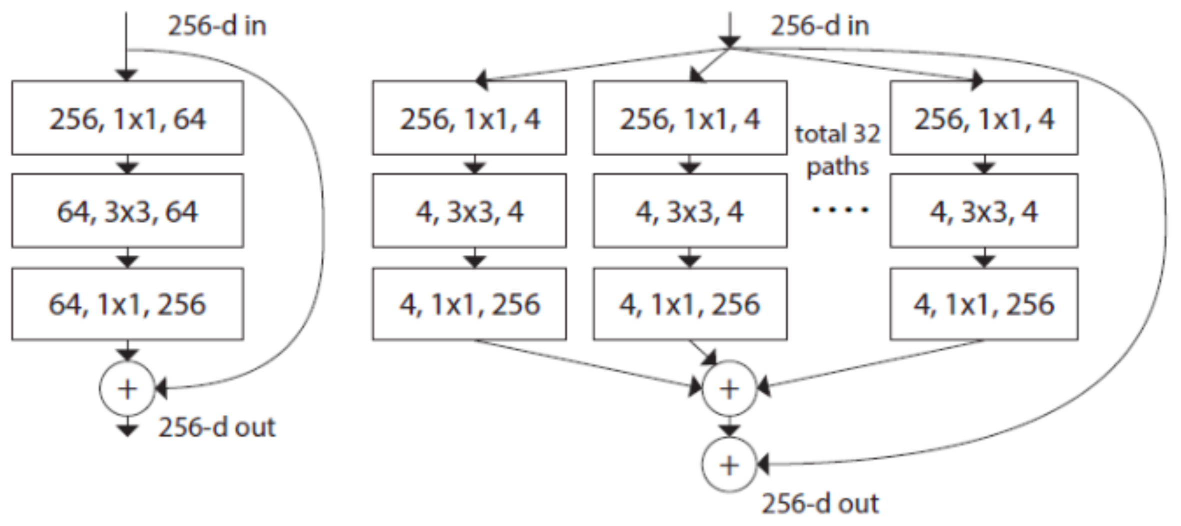


Figure 6: Left: A regular residual bottleneck block. Right: A residual bottleneck block with cardinality 32. Image from [33].

A.2 Search Algorithms

A.2.1 Bayesian Optimization

Bayesian optimization, the default optimizer in HyperMapper, is recommended when the goal is to minimize the black-box function with the least number of function evaluations [23]. The Bayesian optimization algorithm in HyperMapper can use different scalarization methods and acquisition functions. In architecture search the Acquisition function is expected improvement (EI) [13]. The scalarization method used is tchebyshev [15].

A.2.2 Local Search

Local search is used as a benchmark algorithm in this thesis as it was available in the HyperMapper software. More specifically it is a best-improvement multi-start local search algorithm. The search procedure consists of randomly sampling 10,000 configurations and then make best-improvements local searches on the 10 best random samples. Local search is recommended for problems where the black-box function is easy to evaluate [23]. As 10,000 random samples is well out of the budget for the Neural architecture search the number of random samples was set to 50 instead.

A.3 Details for Training of Networks

Most details are done in the same fashion as when training residual networks on CIFAR-10 in [12]. This include random cropping, horizontal flips and train-validation splits of the images, the batch size is 128, the methods for weight initialization and batch normalization and the networks are trained against cross entropy loss. Also for optimization stochastic gradient decent was used with a initial learning rate of 0.1, momentum of 0.9 and a weight decay term 0.0001. When making a final performance estimation on the test data, networks are trained for 164 epochs on all the training data. The learning rate is divided by 10 after half the training, and then again after three fourths of the training. What differed from the pre-processing in [12] was that the data was normalized per channel rather then per pixel, this because it is common in open source implementations and easier to conduct. When conducting the 20 epoch training in the architecture searches the learning rate scheme obviously cannot be followed, then the initial learning rate is kept but reduced to 0.01 for the last three of the 20 epochs in order for the candidates to get somewhat accurate. As the test error after the full training should be unbiased the networks in the search cannot be evaluated on the test data. To avoid this the training data is randomly divided into two parts. Each network will be trained on 45,000 of the samples in the training set and evaluated on the other 5,000.

A.4 NAS Architectures Discovered

Table 10 shows the best performing architecture found by the evolutionary search methods. The best found architectures found by the baseline algorithms can be seen in table 11.

Table 10: The parameters for the best found architectures from the evolutionary methods.

Parameter	Evolution	Regularized Evolution	Crossover Evolution
Block type	Bottleneck	Bottleneck	Basic
Sections	[3,6,3,1]	[7,9,9,1]	[1,10,1,5]
Filters	64	64	32
Filter update	2	2	2
Cardinality	64	2	1
Depth	41	80	36
Weights	6.8M	15.0M	6.6M

Table 11: The parameters for the best found architectures from random sampling, local search and bayesian optimization.

Parameter	Random Sampling	Local Search	Bayesian Optimization
Block type	Bottleneck	Basic	Basic
Sections	[0,3,5,7]	[15, 2, 14, 5]	[5,2,17,1]
Filters	64	64	64
Filter update	1	1	2
Cardinality	2	2	2
Depth	47	74	52
Weights	0.92M	1.4M	12.3M

Master's Theses in Mathematical Sciences 2020:E35
ISSN 1404-6342

LUTFMA-3415-2020

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/