Universiteit Leiden
Opleiding Informatica

Skippy: Automated configuration of CNNs
with skip connections

Name:	Christiaan Lamers
Date:	15/08/2019
1st supervisor:	Bas van Stein
2nd supervisor:	Thomas Bäck

MASTER THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

It takes time and requires expertise to design Convolutional Neural Networks (CNNs) that can be trained to a high accuracy. Because the search space is large and training times are long, it is infeasible to automatically test all possible CNN architectures and hyper parameters. A Mixed Integer (MI) optimization heuristic, named S-Metric Selection Mixed Integer Parallel Efficient Global Optimization (SMS-MIP-EGO), was used to combat this problem. It applies architectural design parameter optimization and hyper parameter optimization in order to construct CNNs and optimizes on both validation accuracy and training time. SMS-MIP-EGO generates MI vectors using Mixed Integer Evolutionary Strategies (MIES), which are evaluated on a fast surrogate model. This surrogate model is trained on parameters of actually evaluated solutions.

A method called *Skippy* was developed to construct deep CNNs with skip connections from a given MI vector. The networks were optimized on both accuracy and training time. The best found network was submitted to an extra step of hyper parameter optimization, in which the training schedule was optimized.

The first goal is to see if the SMS-MIP-EGO heuristic can find high performance networks using the *Skippy* construction method. The second goal is to analyze the architectural parameters and hyper parameters in order to find best practices for designing CNNs.

Contents

1 Introduction

Convolutional Neural Networks (CNNs) are used in a wide range of fields, ranging from computer vision to natural language processing [4]. A CNN consists of stacked layers, that contain nodes, which perform convolutions on an image or a matrix. For clarity the term "image" will be used from now on.
The nodes in such a layer are called kernels. A kernel is a matrix that contains learnable weights. A weight is a float point number that is used in a dot product.
One convolution layer consists of a series of these kernels. A convolution is a operation that performs dot products on the kernels and sections of the image, resulting in one output pixel per operation, as can be seen in figure 1. At first, the left top corner of an image is selected as a section, making one output pixel. In the next step, an adjacent section more to the right is taken as a selection, resulting in a different output pixel. When the end of the image is reached, the leftmost part, just below the first selection is selected to produce an output pixel. After this, a section to the right is selected, etcetera. The output pixels will form a new image, called a feature.
The reason why these convolutions are useful, is because of the fact that the weights in the kernels are trainable through supervised learning. Kernels tend to learn certain patterns like vertical or diagonal lines, which then can be detected in images. The resulting features show where in the image these patterns are present. The next layer can detect more abstract patterns in these features. A combination of certain line patterns can form a circle for example. When stacking more and more of these layers on top of each other, the learned patterns can become more and more abstract, up until the point where eyes, noses and ears can be detected. These features can then be used to detect faces, dogs or cats for example.
CNNs come in all sizes and shapes. Design choices include the type of activation function, the size of the kernels, the width of each layer, the depth of the network, what training schedule to use, etcetera. These choices are often based upon what is described in literature, or they require high expertise to make. Automated machine learning can be used to make these choices instead. It aims to automatically generate and evaluate these design choices. It requires less expertise, design time and has the potential to find novel solutions.
Training a CNN requires time. Training a CNN on a GPU for only ten epochs can last for multiple hours. It is not uncommon for a CNN to be trained for days. Because of these time constraints, evaluating every CNN that is being automatically generated is infeasible. Instead it is useful to construct a surrogate model. This surrogate model predicts the effectiveness of a CNN based on a vector of given parameters. These parameters would be used by the construction method to decide how to build the CNN. Such a surrogate model can predict the accuracy or training time in a fraction of the time of actually training the CNN. The downside is that a surrogate model is never 100% accurate. However, a surrogate model is useful for quickly evaluating a lot of configurations and selecting a high performance network with a reasonable amount of certainty. This selected network can then be evaluated using the expensive method of actually training it on a GPU. The accuracy and training time information from training the network can then be used to train a new surrogate model, continuing the cycle.
Deep neural networks are an increasingly popular machine learning method that is proven to be very powerful in the area of image classification. However, the deeper the network, the smaller the values in the gradient of the loss function get, making it hard to train the network. This is referred to as the "vanishing gradient problem" . The result is that making a network deeper will result in better performance up to a certain point. Once the network gets too deep,

the performance declines. A Residual Neural Network (RESnet) [10] tackles this problem by building a network out of residual blocks. A residual block consists of a stack of convolutions circumvented by a skip connection. As the name suggests, a skip connection skips convolutional layers, allowing for shorter paths in the network and thus better gradient preservation. Up until now, these RESnet structures could not be automatically constructed and optimized. The *Skippy* method is proposed to solve this problem. It is able to add skip connections to deep CNNs as specified by their own parameters. These parameters can then be optimized along with all other parameters of the network using automated machine learning. The heuristic that is used for automated machine learning in the proposed method is called S-Metric Selection Mixed Integer Parallel Efficient Global Optimization (SMS-MIP-EGO). This heuristic is explained in section 3.

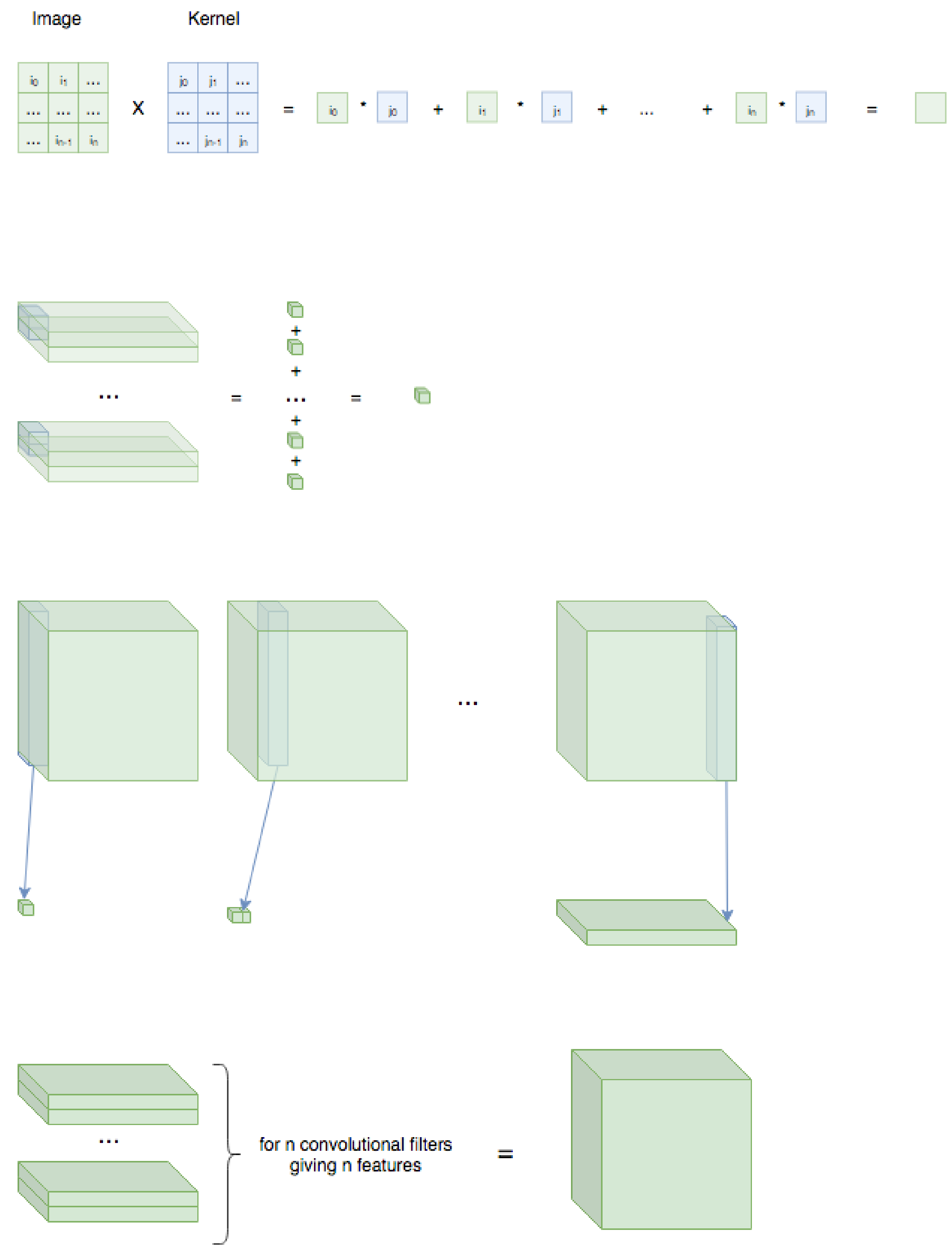


Image
Kernel
i0 i1 ...
... ...
... in-1 in
X
j0 j1 ...
... ...
... jn-1 jn
= i0 * j0 + i1 * j1 + ... + in * jn =
for n convolutional filters giving n features
=

Figure 1: How 2D-convolutions with multiple channels work: The dot product is taken of a section of a stack of images (depicted in green) and a 2D-convolutional filter (a stack of kernels, depicted in blue), which forms a pixel in the output feature. A stack of kernels is repeatedly offset by a number of pixels determined by the stride value, generating a new feature. Each convolutional filter generates a new output feature. All output features are stacked on top of each other to form a new stack of features, upon which the next layer of 2D-convolutions can act.

2 Related Work

Automated machine learning can be implemented in many ways. The methods differ in the way new configurations are generated as well as the order of configuration manipulation and training. Some methods alternately train networks and change their topology, while other methods keep the configuration manipulation and training separated. The method of Gaier et al. [9] bypasses the need for vectors of fixed length. It uses kernel-based surrogate models to allow predictions on variable topologies instead of on fixed-length vectors. Its surrogate model can predict performance by only using the distances between samples using a given distance metric.

In the method of Elsken et al. [8] effective Neural Networks are searched for by combining a hill climbing algorithm with the “NetMorph” operation. NetMorph generates a group of child networks from a parent network by altering the topology of the network, while keeping the functionality identical. These child networks and parent network are then trained for one epoch and the best is selected to be the new parent network. Thus alternating training and topology modification.

Similar to the method of Elsken et al, a method is described in Wei et al. [21] that alters the topology of a network while keeping the behaviour identical. This method is used to create a pool of altered child networks from a parent network. These child networks are then trained for one epoch. The best result is selected to be the parent network for the next iteration. This way a network topology is grown, while at the same time it is trained.

Another example of simultaneously training networks while evolving the topology is the method of Stanley and Miikkulainen [19]. It uses the NeuroEvolution of Augmenting Topologies (NEAT) to simultaneously evolve the topology of networks as well as their weights. Networks and their weights are represented as genetic information. Innovations are protected using speciation, which classifies networks into niches. Small networks are incrementally made more complex.

The method of Zoph and Le [3] uses a Recurrent Neural Network (RNN) as a controller for the hyper parameters, separating optimization and training. This method is more similar to the proposed method, where the S-Metric Selection Mixed Integer Parallel Efficient Global Optimization (SMS-MIP-EGO) heuristic selects parameters. Instead of a RNN to select parameters, SMS-MIP-EGO uses Mixed Integer Evolutionary Strategies (MIES) [14].

SMS-MIP-EGO uses multi-objective (bi-objective) optimization, just like the method of Tan et al. [20] uses multi-objective optimization to optimize CNNs. The CNNs it optimizes are intended for use on mobile devices, on accuracy as well as latency. For optimization, reinforcement learning approach is used, whereas SMS-MIP-EGO uses MIES.

The idea to add skip connections to the *Skippy* construction method came from the Residual Neural Network (RESnets) from He et al. [10]. Its findings are that RESnets suffer less from the vanishing gradient problem thanks to their use of skip connections. RESnets generally keep performing better the deeper they get.

RESnets are more prone to overfitting according to Ebrahimi et al. [7], therefore data augmentation was used in one *Skippy* experiment. The parameters for the data augmentation were part of the search space SMS-MIP-EGO optimized on. The results of Perez and Wang [16] show that traditional data augmentation is very effective in increasing the accuracy of a Neural Network.

3 Skippy

In this section the proposed method to automatically build CNNs, called *Skippy*, is introduced and explained. The heuristic used for automatically generating CNN configurations is called S-Metric Selection Mixed Integer Parallel Efficient Global Optimization (SMS-MIP-EGO). It is a bi-objective optimizer that is used to minimize the training time and the validation loss of CNNs. Minimizing the validation loss will maximize the validation accuracy. It constructs random forests [5] as surrogate models for both the training time and the validation loss. A random forest is a model that is trained on a set of data by building decision trees that split the data on a feature with maximal information gain out of a set of randomly selected features. These decision trees can perform regression in order to make predictions. A MIES algorithm [14] is used to generate new configurations. The MIES algorithm uses the surrogate models to evaluate the fitness of its generated configurations. The fitness is calculated using Ponweiser's [17] S-metric. After the MIES algorithm made its choice for a solution, this solution is evaluated by training it on a GPU. The resulting training time and validation loss are used to retrain the time and loss surrogate models. SMS-MIP-EGO is and adaptation of van Stein's [22] MIP-EGO method, which is a single objective optimizer.

3.1 CNN construction with skip connections

To construct a CNN from a given Mixed Integer (MI) vector, a construction method called *Skippy* is proposed. This construction method is based on the construction method of van Stein's [22] MIP-EGO method. Just like van Stein's construction method, *Skippy* builds a CNN that consists of a certain number of stacks. A stack consists of a convolutional part followed by a dimension reduction part, which decreases the feature size. Each stack has its own parameters, such as the number of convolutional layers, the width of the convolutional layers and the size of the kernels used by the convolutions, as well as the amount of reduction in feature size.

Skippy differs from van Stein's method in the sense that it uses more stacks in order to a allow a search space where a network exists similar to RESnet-34, as described in figure 3 of He et al. [10]. The method of van Stein uses no Max pooling layers, but convolutional layers with a stride larger than one. Van Stein's method does this because it is simpler, more elegant and because of the claim of Springenberg et al. [18] that Max pooling layers can be replaced by Convolutional layers. *Skippy* allows dimension reduction with either the use of convolutional layers or Max pooling layers.

The main difference with van Stein's method and *Skippy* is *Skippy's* use of skip connections. These skip connections create shortcuts between layers that are more than two layers apart in depth. This allows for more feature recombination options as well as better gradient preservation. Skip connections form chains that traverse the network. After a skip connection is attached, a new skip connection is formed immediately. Each CNN can have up to five chains of skip connections. The number of layers a skip connection skips and at what depth a chain of skip connections begins is determined by separate parameters per chain. These parameters are part of the search space of SMS-MIP-EGO.

Connecting layers using skip connections, and thereby combining features, is not trivial. When convolutions and strides act on features, they produce output features with different dimensions. The length and width of features can change as well as the number of features. For this

reason, most of the time, the dimensions of the skip connection feature does not match the dimensions of the feature it needs to be connected to.
The way convolutions act on features is depicted in figure 1. The top of the figure shows a kernel being aligned with one part of the incoming layer. The dot product of the kernel and the part of the incoming feature becomes a new pixel. One node of a convolutional layer is made of a stack of these kernels. Such a stack of kernels produces a stack of pixels. These pixels are added together forming one pixel of one output feature. A stack of kernels moves over the incoming feature, creating one pixel for each convolution. These pixels together form one outgoing feature. All n stacks of kernels in the convolutional layer perform such an operation, creating n outgoing features. These outgoing features are stacked on top of each other creating the input for the next layer.
In order to connect a skip connection feature neatly, the feature must match the size of the feature it connects to. Figure 2 shows an example of a CNN with a skip connection. The features at the input side are sent through an outgoing skip connection and are depicted as a red cube. At the point of the incoming skip connection, the feature to be connected to and the incoming feature must be concatenated, but their size does not match necessarily. If the length and width of the incoming skip connection feature is larger than the feature to be connected to, the incoming skip connection feature's size is reduced by Max pooling. Since Max pooling can result in less than perfectly matching length and width, it is opted to make the length and width of the incoming skip connection feature equal to or smaller than the width and length of the feature to be connected to.
After this, a Dropout layer can be added, when the skip connection skips any Dropout layer in the main part of the network. The value of Dropout is equal to the last main part Dropout layer skipped. If the length and width of the incoming skip connection feature are smaller due to rigorous Max pooling, or the incoming skip connection feature being too small in the first place, the incoming skip connection feature is zero padded to match the length and width of the feature to be connected to exactly. Then the feature to be connected to and the incoming skip connection feature are concatenated.
If multiple incoming skip connection features are connected at this point, they are all concatenated after their length and width have been altered in the same manner. In order to avoid an explosion of features and to promote feature recombination, the concatenation is sent through a convolutional layer with a kernel size of 1×1. This way no actual convolutions take place, but learning how to recombine features through weight updates is still possible. This makes it possible to recombine features in a more relevant way. This is an advantage over the addition operation, which recombines features in an arbitrary manner, and thus does not check which feature to recombine with which. Wistuba et al. [15] describe the method of a 1×1 convolution to alter the number of features between layers in figure 3 of their survey. This method can behave the way RESnet [10] connects skip connections. The weights selecting the feature to be connected to could all be one and the weights selecting the features of the incoming skip connection layer could act as a projection. Thereby adding a projection of the incoming skip connection layer to the layer to be connected to. If the weights selecting the incoming skip connection feature are all one and the rest is zero, a feature is reused completely.
Unfortunately, an unnecessary extra dropout layer was added after the global average pooling layer, in case global average pooling was specified by the MI vector. This error was removed before commencing the experiments that use data augmentation in section 10.

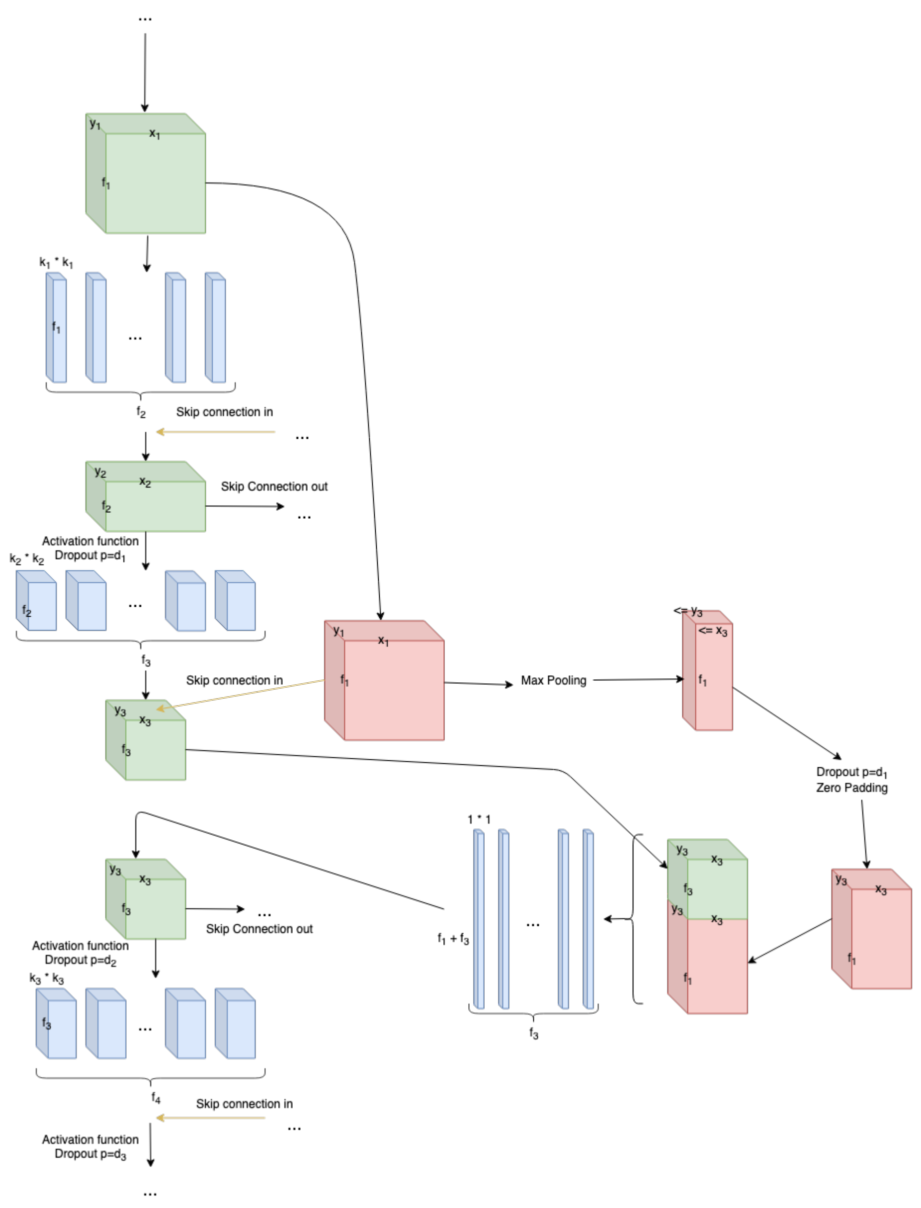


Figure 2: How *Skippy* connects skip connections: A stack of features a (green or red) consist of f_a features. Each feature is x_a by y_a pixels. A 2D-Convolutional layer b (blue) has f_{b+1} stacks of kernels. Each kernel is k_b by k_b pixels. Each stack of kernels consists of f_b kernels.
A skip connection carries over a stack of features from a higher layer (shown in red) to be connected to the "skip connection in" point. In order to concatenate both stacks of features, their dimensions must match. If the x_1 and y_1 dimensions of the skip connection features are larger than x_3 and y_3 dimensions of the features to be connected to, they are reduced using Max pooling, to make it at least as small as the x_3 and y_3 dimensions. A Dropout layer is added here, if the skip connection passes over a Dropout layer in the main network. The amount of dropout is equal to that of the last Dropout layer skipped, namely d_1. If the y_1 and x_1 dimensions are too small, Zero padding is performed in order to make x_1 and y_1 match x_3 and y_3 perfectly. Both stacks of features are concatenated. A 1×1 2D-Convolutional layer reduces the number of features to the number at the "skip connection in" point, namely f_3. The resulting feature is passed through the rest of the network as well as to a new skip connection.

Parameter	Explanation
filters	Number of filters for each stack's head and tail
kernel_size	Kernel dimensions for each stack's head ant tail
strides	Stride for dimension reduction part of each stack
stack_sizes	Number of layers per convolutional part for each stack
activation	Type of activation to be used after convolutions
activation_dense	Type of activation function for output layer
step	Whether or not to decrease learning rate during training
global_pooling	Whether or not to use global pooling in stead of flattening
skstart	Start points of each chain of skip connections
skstep	Number of layers to be skipped per skip connection (1 means no skip connection)
max_pooling	Whether to use Max pooling in stead of convolutions with stride > 1
dense_size	Width of the two dense layers before the output layer
drop_out	Dropout value for each dropout layer
lr	Learning rate
l2_regularizer	Amount of l2 kernel regularization

Table 1: Search space explanation

3.2 General architecture

Algorithm 1, 2 and figure 3 explain the general architectural rules *Skippy* uses to build a CNN. Algorithm 1 shows in pseudocode how a CNN is built by *Skippy* given a MI vector of parameters. Algorithm 2 shows in pseudocode what steps are taken to create and connect skip connections. Figure 3 is a visual representation of the pseudocode of algorithm 1. Table 1 shows a brief explanation of the parameters in the MI vector.
A dropout layer is added immediately after the input. Next, a series of stacks follows. Each stack consist of a convolutional part followed by a dimension reduction part. The convolutional part consists of a number of repetitions of the following trio: a convolutional layers, followed by an optional skip connection input/output, followed by an activation layer. The dimension reduction part consist of either one Max pooling layer, or one Convolutional layer with a stride larger of equal to one, followed by an optional skip connection input/output, followed by an activation layer. Associated with each stack are a number of parameters as stated in table 1. After these stacks, the output is flattened by either Global Pooling, or a Flatten operation. This is followed by two densely connected layers each followed by an Activation and Dropout layer. Finally, the output layer is added using a Softmax function.
The "Pruning one node layers" method, a variation of *Skippy*, is a method that prunes the last layers from a network. It is described in section 8. The feature's length and width dimensions tended to lower down to one somewhere along the depth of the network. Since this resembled a densely connected network more than a CNN, it was decided to try and cut this 1×1 feature part from the network. This method detects when the feature's length and width dimensions become one. As soon as this happens, the method stops building convolutional layers, flattens the features and immediately builds the final three fully connected layers. Figure 15 and 16 are an examples of using the standard *Skippy* method versus the "Pruning one node layers" method.

Algorithm 1 Skippy

```
 1: procedure SKIPPY
 2:     add Input layer
 3:     add Dropout layer
 4:     for all stack in stacks do
 5:         for all layers in stack do
 6:             add Conv2D layer
 7:             Connect_Skip(previous_layer)
 8:             add Activation layer
 9:         end for
10:         if Max pooling then
11:             add MaxPooling layer
12:         else
13:             add Conv2D layer with larger stride
14:             Connect_Skip(previous_layer)
15:             add Activation layer
16:         end if
17:         add Dropout layer
18:     end for
19:     if global pooling then
20:         add global pooling layer
21:         add Dropout layer
22:     else
23:         add Flatten layer
24:     end if
25:     add dense layer 0
26:     add Activation layer
27:     add Dropout layer
28:     add dense layer 1
29:     add Activation layer
30:     add Dropout layer
31:     add last dense layer
32:     add Softmax activation layer
33: end procedure
```

Algorithm 2 Connect_Skip

```
 1: procedure CONNECT_SKIP(layer)
 2:     for all skip connections in memory do          ▷ end skip connections if needed
 3:         pool skip connection layer if needed
 4:         add dropout layer if needed
 5:         pad skip connection layer if needed
 6:         Concatenate layer and skip connection layer
 7:         Convolute Concatenated layer with 1 × 1 kernel
 8:     end for
 9:     begin skip connections if needed
10: end procedure
```

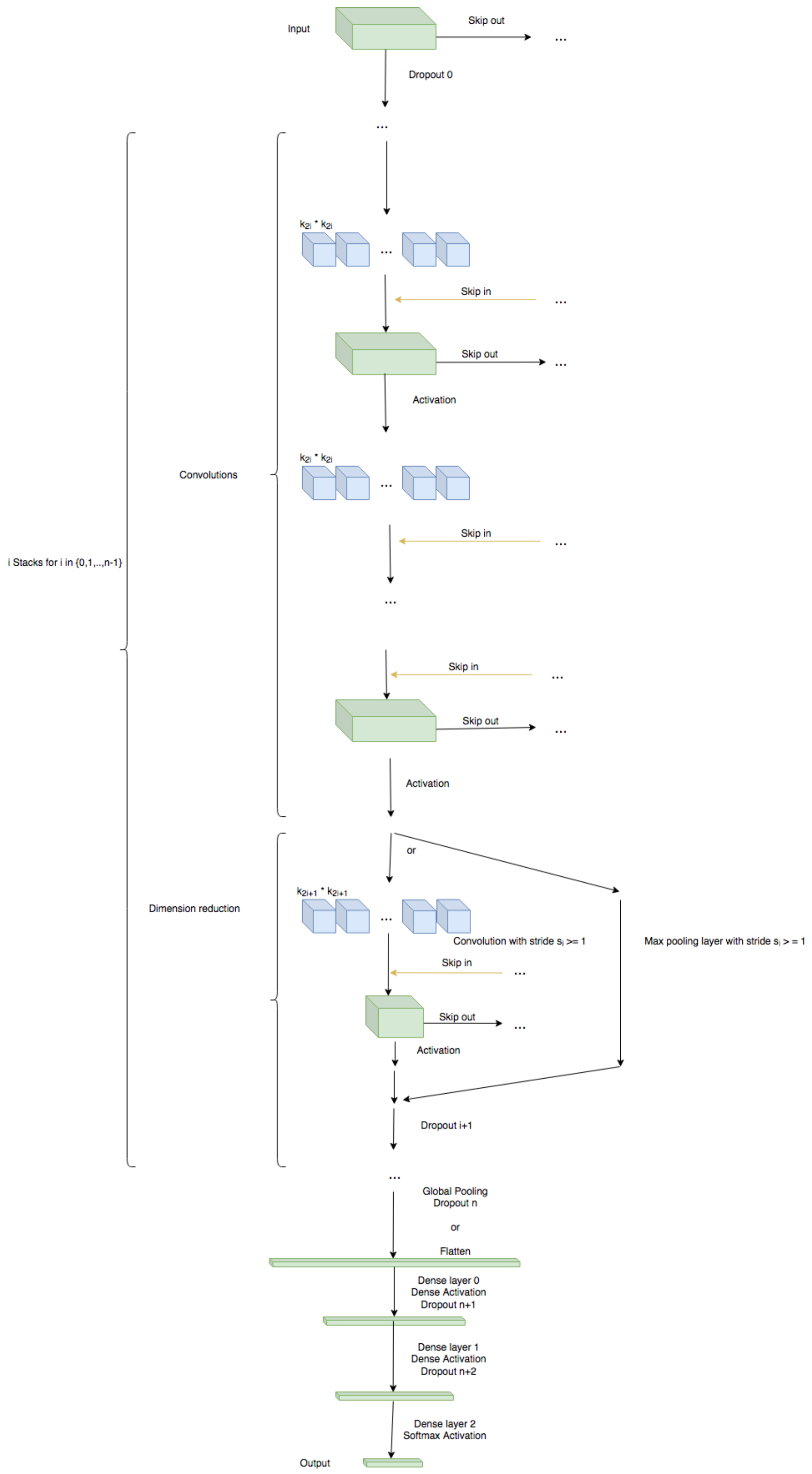


Figure 3: The general architectural rules *Skippy* uses to build a CNN: After the input, a dropout layer is added. After this n stacks follow. Each stack consists of a Convolutional part and a dimension reduction part. Each stack has its own set of parameters as denoted in table 1. The “Skip in ” point is where skip connections are connected to the network. The “Skip out” point is where skip connections split from the network. Dimension reduction can take place either using convolutions with a stride greater than one, or with a Max pooling layer. After all stacks, Global Pooling is performed if the MI vector specifies this. After this, two fully connected layers are connected, followed by an output layer using Softmax activation.

4 Research question

A series of experiments is performed, with the aim to answer the following question:

Can SMS-MIP-EGO, when combined with Skippy, successfully find CNNs in a given search space, minimizing training time and maximizing validation accuracy, and can these CNNs beat current state-of-the-art CNNs?

The second goal is to analyze the found configurations and to distill good design strategies or rules of thumb.
A series of iterative experiments were performed. They consisted of running SMS-MIP-EGO with the *Skippy* construction method. The goal was to find CNNs with good performance. During the experiments, data analysis was performed in order to improve the next experiments. All experiments used the CIFAR10 data set [12].
SMS-MIP-EGO uses a predefined search space to find MI vectors. Table 1 shows a brief explanation of what each parameter of the search space stands for.
The networks were trained on a cluster of sixteen Nvidia Tesla K80 GPUs. Each individual network was trained on one GPU at a time. Up to ten networks were trained in parallel.
The code of the SMS-MIP-EGO and *Skippy* methods can be found online [13].
Data analysis was performed on some experiments. The corresponding data can also be found online [13] in the "data_thesis" folder.

5 Pre-experiments

To validate the performance of *Skippy*, a series of early experiments was performed. The goal of these experiments was to test if the skip connections add to the functionality, and to see if certain design choices can be beneficial. The variations of the method are called *Skippy3* and *Skippy4*, simply because they were the third and fourth variation implemented. The first method implemented was vanilla *Skippy*. The second method implemented was *Skippy2*, this method sent features through an activation function, before connecting skip connections. This method was not further tested, since it was less like the method of He et al. [10], which first connected skip connections and then sent the combined features through an activation function, just like vanilla *Skippy*, *Skippy3* and *Skippy4*.
Table 2 shows the parameters used to build network variations, similar to RESnet-34 as described by He et al. [10], using the vanilla *Skippy* method. The only difference between He et al.'s version and this one is that skip connections are built using *Skippy's* method, thus skip connections are not connected using projection, padding and addition, but instead use pooling, padding, concatenation and dimension reduction. A schema of the networks, built by the vanilla *Skippy* method, is shown in figure 4a.
A network without skip connections was built. The goal is to see if skip connections add to the functionality of the network. A schema of the built network is shown in figure 4b.
Skippy3 does Max pooling and padding, just like vanilla *Skippy*, but after this it does dimension reduction using the 1×1 2D convolution, before concatenation. The dimension reduction is only done when the width of the skip layer is bigger than k times the width of the layer it is to be concatenated with. This is tested for $k \in \{1, 3, 7\}$. In older iterations of the *Skippy* method, no dimension reduction was done after connecting skip connections, thereby maximizing the

chance of feature reuse. However, this resulted in excessive growth of the features that were passed by the skip connections, often resulting in a network that did not fit in the memory of the GPU. This method of connecting skip connections allows for more feature re-use, while keeping the amount of excessive feature growth reduction variable. Four networks were built using the *Skippy3* method, using the parameters of table 2. Figure 4c shows a schema of this of network, for $k = 1$, figure 4d, for $k = 3$ and figure 5a, for $k = 7$. The goal is to see how variating k impacts the validation accuracy and memory requirements. Figure 5b shows a schema of this network, for $k = 1$, where no dropout layer is used in the skip connections. The goal was to see if these skip connection dropout layers impacted the network's validation accuracy.

Skippy4 does Max pooling and padding, after which it does dimension reduction using the 1×1 2D convolution. It does not use concatenation like the previous methods, but addition. This implementation is the closest to the method of He et al. [10]. Two networks were built with the *Skippy4* method, using the parameters of table 2. Figure 5c shows a schema of this network. Figure 5d shows a schema of this network, without dropout layers in the skip connections.

Table 3 shows the number of trainable parameters, the training accuracy and the validation accuracy for networks built by different *Skippy* versions and parameters. All networks were trained for 20 epochs.

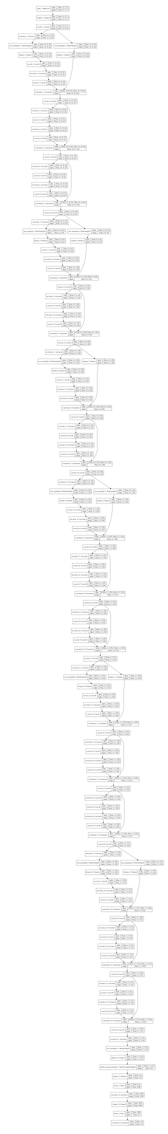
Comparing the results of the vanilla *Skippy* method with the *Skippy* method that uses no skip connections, it can be seen that omitting the skip connections reduces the validation accuracy to the level of random guessing: namely 0.1, where CIFAR10 has ten classes. Thus is it a good choice to use the skip connections.

The *Skippy3* method shows some validation accuracy improvement over vanilla *Skippy*, but the number of trainable parameters is also increased, meaning more GPU memory is needed. This means that networks in the search space tend to be bigger, thus the chance is higher that a built network does not fit in the GPU memory. When increasing k from one to seven, the validation accuracy increases from 0.7359 to only 0.7832, while the number of trainable parameters more than doubles from 3.37e7 to 7.57e7, giving diminishing returns. Therefore, vanilla *Skippy* was deemed more simple and elegant, and thus chosen as the method for the big experiments. Omitting the dropout in the skip connections showed a slight but insignificant increase in validation accuracy. In order to counteract overfitting through the skip connections, it was opted to keep the dropout layers in the skip connections.

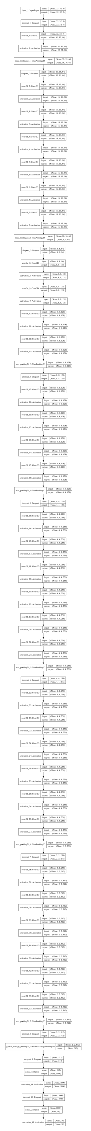
Skippy4 shows a slightly higher validation accuracy than vanilla *Skippy*, but not enough to be significant. It was opted to use vanilla *Skippy*, instead of *Skippy4*, because vanilla *Skippy* allows for complete feature reuse and recombination of every feature with every feature per layer, before the 1×1 projection layer, as is described in subsection 3.1. Thus, vanilla *Skippy* is the most flexible method and the most interesting to be tested.

Parameter	Value
stack_0	1
stack_1	6
stack_2	4
stack_3	4
stack_4	6
stack_5	6
stack_6	6
s_0	2
s_1	2
s_2	1
s_3	2
s_4	1
s_5	2
s_6	1
filters_0	64
filters_1	64
filters_2	64
filters_3	64
filters_4	128
filters_5	128
filters_6	128
filters_7	128
filters_8	256
filters_9	256
filters_10	256
filters_11	256
filters_12	512
filters_13	512
k_0	7
k_1	1
k_2	3
k_3	1
k_4	3
k_5	1
k_6	3
k_7	1
k_8	3
k_9	1
k_10	3
k_11	1
k_12	3
k_13	1
activation	'relu'
activ_dense	'softmax'
dropout_0	0.001
dropout_1	0.001
dropout_2	0.001
dropout_3	0.001
dropout_4	0.001
dropout_5	0.001
dropout_6	0.001
dropout_7	0.001
dropout_8	0.001
dropout_9	0.001
lr	0.01
l2	0.0001
step	False
global_pooling	True
skstart_0	1
skstart_1	1
skstart_2	1
skstart_3	1
skstart_4	1
skstep_0	2
skstep_1	1
skstep_2	1
skstep_3	1
skstep_4	1
max_pooling	True
dense_size_0	1000
dense_size_1	0

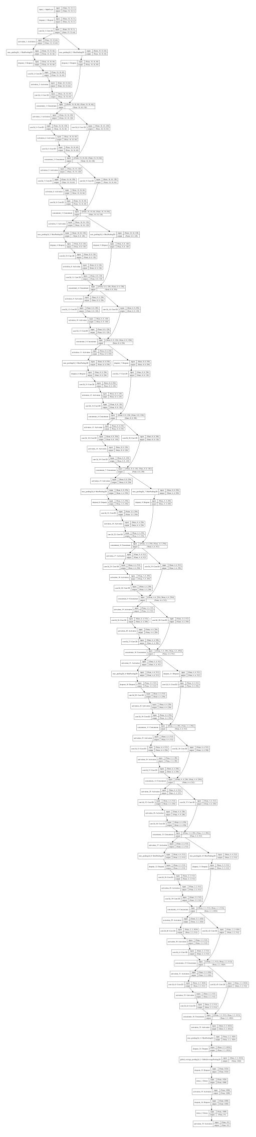
Table 2: Parameters of the network similar to RESnet-34 used in the early experiments



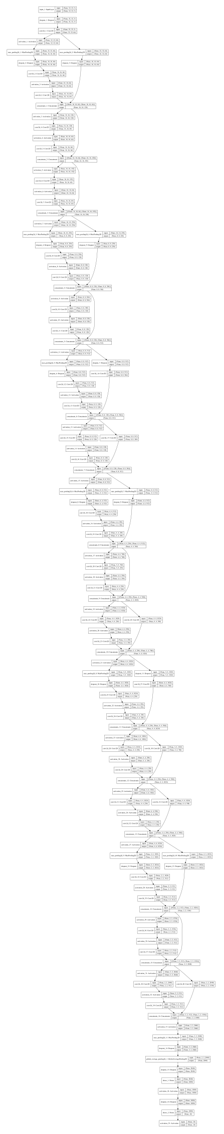
(a) CNN similar to RESnet-34 built by *Skippy*



(b) CNN similar to RESnet-34 built by *Skippy* but without skip connections



(c) CNN similar to RESnet-34 built by *Skippy3* using one times current layer width as dimension reduction trigger

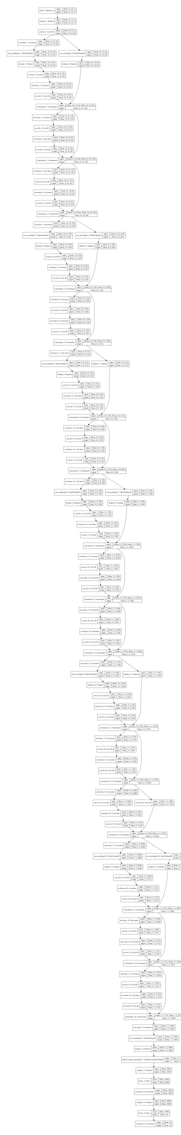


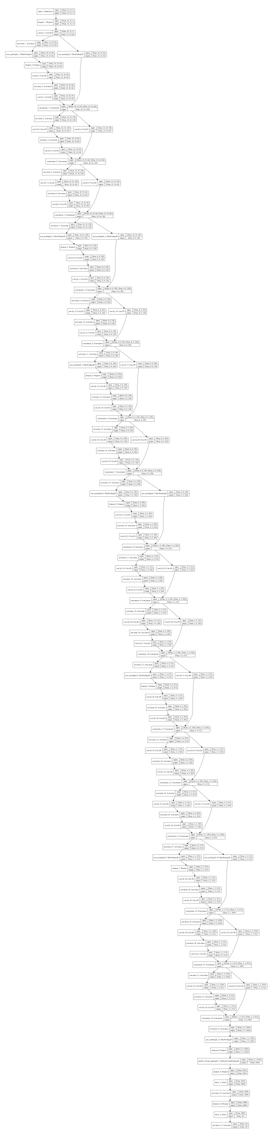
(d) CNN similar to RESnet-34 built by *Skippy3* using three times current layer width as dimension reduction trigger

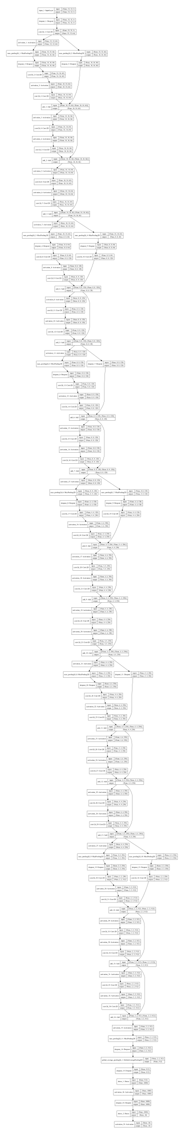
Figure 4: RESnet34 variations

Method	# Trainable Parameters	Train acc.	Validation acc.
Skippy	23972930	0.9422	0.7219
Skippy no skip connections	**21626242**	0.0980	0.1000
Skippy3 max layer width 1	33691778	0.9620	0.7359
Skippy3 max layer width 3	55841410	0.9757	0.7601
Skippy3 max layer width 7	75740290	0.9837	**0.7832**
Skippy3 max layer width 1 no dropout in skip	33691778	0.9646	0.7376
Skippy4	21799170	**0.9901**	0.7305
Skippy4 no dropout in skip	21799170	0.9885	0.7396

Table 3: Early experiments results. Networks were trained for 20 epochs


(a) CNN similar to RESnet-34 built by *Skippy3* using seven times current layer width as dimension reduction trigger. Note that almost no skip connection, a 1×1 2D Convolution is used. This means that every feature appears at almost every depth in the network.


(b) CNN similar to RESnet-34 built by *Skippy3* using one times current layer width as dimension reduction trigger, using no dropout in the skip connections


(c) CNN similar to RESnet-34 built by *Skippy4*

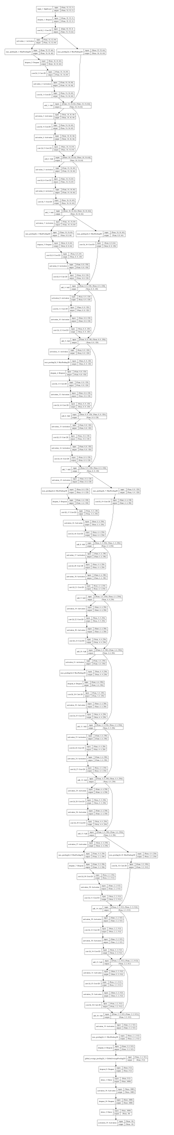

(d) CNN similar to RESnet-34 built by *Skippy4*, using no dropout in the skip connections

Figure 5: RESnet34 variations

Parameter	Type	Bounds	# Dimensions
filters	Discrete	[10, 600]	14
kernel_size	Discrete	[1, 8]	14
strides	Discrete	[1, 5]	7
stack_sizes	Discrete	[0, 7]	7
activation	Nominal	[”elu”,”relu”,”tanh”,”sigmoid”,”selu”]	1
activation_dense	Nominal	[”softmax”]	1
step	Nominal	[True, False]	1
global_pooling	NominalSpace	[True,False]	1
skstart	Discrete	[0, 7]	5
skstep	Discrete	[1, 10]	5
max_pooling	Nominal	[True, False]	1
dense_size	Discrete	[0,2000]	2
drop_out	Continuous	[1e-5, 0.9]	10
lr	Continuous	[1e-4, 1.0e-0]	1
l2_regularizer	Continuous	[1e-5, 1e-2]	1

Table 4: search space of the “Base” experiment

Time (s)	Loss	Acc
316.16	2.30	0.10
395.52	2.30	0.10
401.37	1.47	0.23
951.97	1.33	0.27
952.83	0.90	0.41
1007.84	0.31	0.74

Table 5: Paretofront of the “Base” experiment: each entry is trained for ten epochs.

6 Base experiment

The first big experiment evaluated 410 networks on CIFAR10 by training them for ten epochs. MIES used a maximum of 500 iterations, with random forests containing ten trees as surrogate models. The search space of this experiment can be seen in table 4.
Table 23 in the appendices shows the file name information of the data file and the file name of the construction script used in this experiment. The data file can be found online [13].
Figure 6 shows the performance of networks evaluated by SMS-MIP-EGO. Table 5 shows the training time, validation loss and validation accuracy of the Pareto optimal solutions. Of this experiment, 4.15% of the produced networks did not fit in the memory of the GPU.
After this experiment, the configurations were split into two groups based on their validation accuracy. Configurations with a validation accuracy greater or equal to 0.4 were put into the group of “good” configuration, while the ones with a validation accuracy smaller than 0.4 were put into the group of “bad” configurations. The networks that were too large to fit in memory, and thus received a large penalty value, were filtered out for all analyses. The “Base” experiment results contain only one network in the “good” section, so it was not used. It contained 392 networks in the “bad” section. The found trends, as shown in figure 7, 8a, 8b, 9a and 9b were used to improve the bounds of the search space.

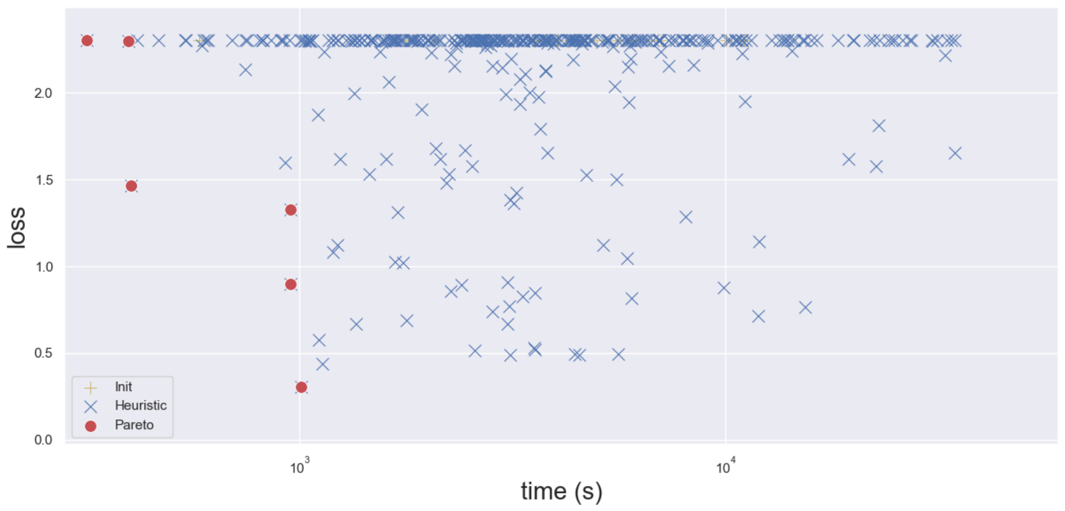


Figure 6: The "Base" experiment on CIFAR10. 410 samples taken, using ten evaluation epochs, ten trees in the random forest. Init: 20 solutions sampled to start off SMS-MIP-EGO, Heuristic: solutions evaluated by SMS-MIP-EGO, Pareto: Pareto optimal solutions.

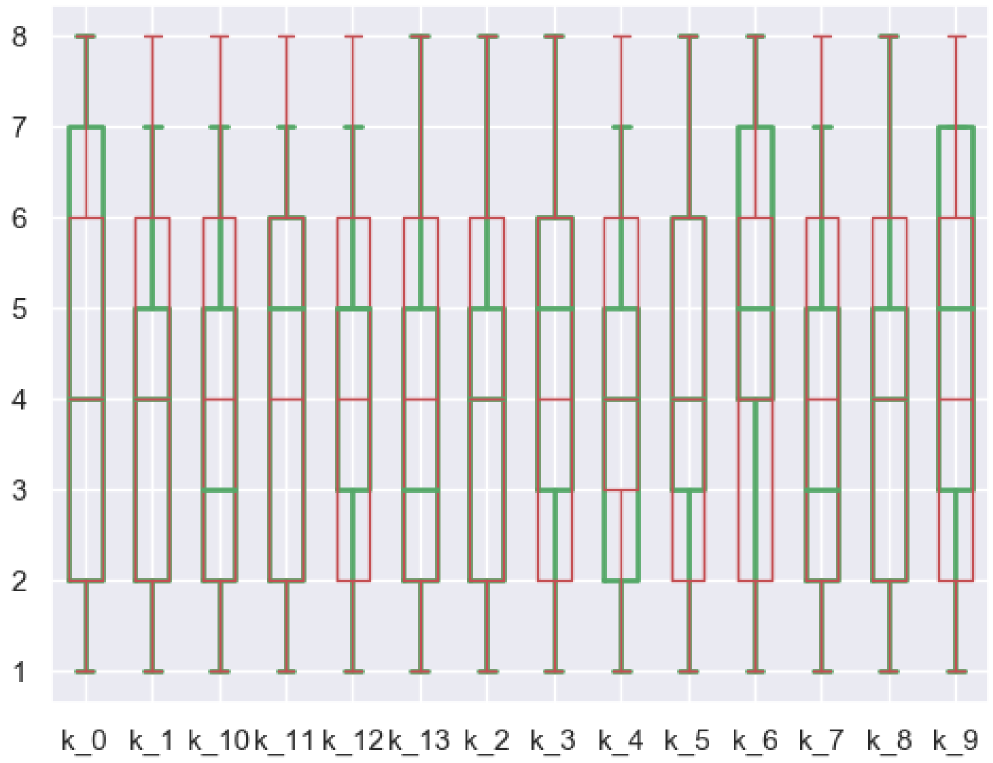
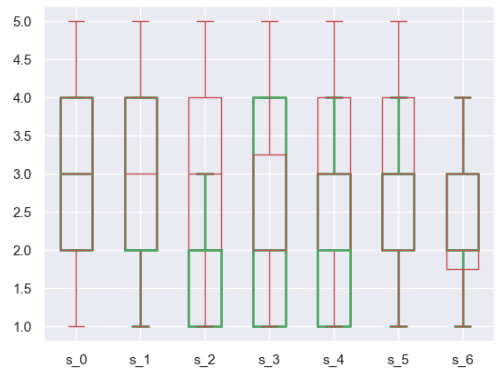
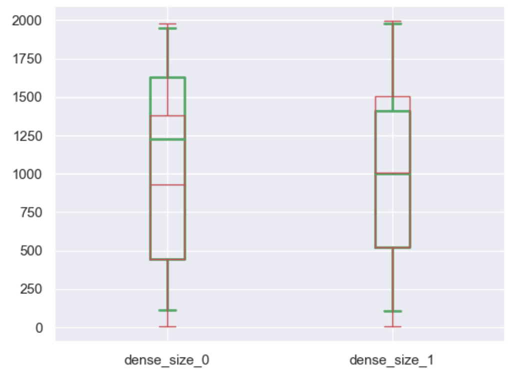


Figure 7: The distribution of the kernel sizes of configurations of the "Base" experiment, split on validation accuracy greater or equal to 0.4 (green) and smaller than 0.4 (red).

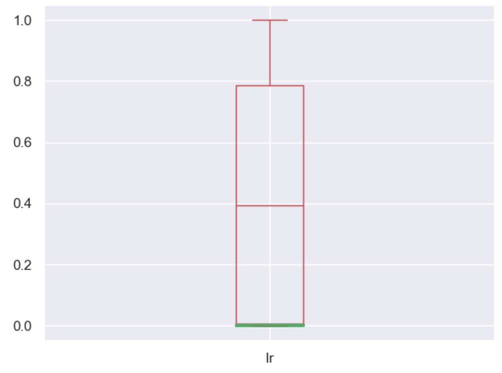


(a) The distribution of the strides of configurations of the "Base" experiment, split on validation accuracy greater or equal to 0.4 (green) and smaller than 0.4 (red).

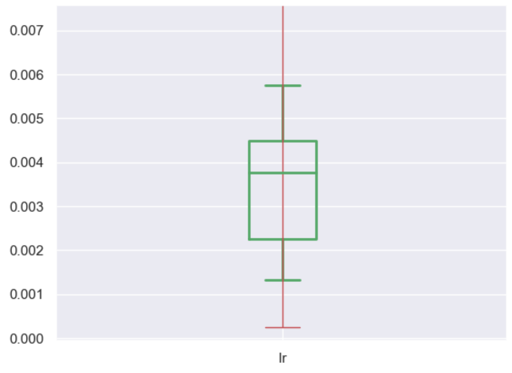


(b) The distribution of the dense layer sizes of configurations of the "Base" experiment, split on validation accuracy greater or equal to 0.4 (green) and smaller than 0.4 (red).

Figure 8: Distributions of features of the "Base" experiment



(a) The distribution of the learning rate of configurations of the "Base" experiment, split on validation accuracy greater or equal to 0.4 (green) and smaller than 0.4 (red).



(b) Close up of he distribution of the learning rate of configurations of the "Base" experiment, split on validation accuracy greater or equal to 0.4 (green) and smaller than 0.4 (red).

Figure 9: Distributions of features of the "Base" experiment

Parameter	Type	Bounds	# Dimensions
filters	Discrete	[10, 600]	14
kernel_size	Discrete	[1, 16]	14
strides	Discrete	[1, 10]	7
stack_sizes	Discrete	[0, 7]	7
activation	Nominal	[”elu”,”relu”,”tanh”,”sigmoid”,”selu”]	1
activation_dense	Nominal	[”softmax”]	1
step	Nominal	[True, False]	1
global_pooling	Nominal	[True,False]	1
skstart	Discrete	[0, 7]	5
skstep	Discrete	[1, 10]	5
max_pooling	Nominal	[True, False]	1
dense_size	Discrete	[0,4000]	2
drop_out	Continuous	[1e-5, .9]	10
lr	Continuous	[1e-4, 1.0e-2]	1
l2_regularizer	Continuous	[1e-5, 1e-2]	1

Table 6: search space of the “Fine-tuned search space” experiment

7 Fine-tuned search space experiment

To follow up the “Base” experiment, the search space was altered according to the trends shown in figure 7, 8a, 8b, 9a and 9b, so that it contained more useful configurations. The *kernel_size* parameter was increased from [1, 8] to [1,16], since kernel size zero, six and nine were on the higher side in the group of “good” solutions. The *strides* parameter interval was increased form [1, 5] to [1,10], because the value of stride three was more widely spread out in the group of“good”solutions. The *dense_size* parameter interval was increased form [0,2000] to [0,4000], because the group of“good”solutions had a slightly higher *dense_size_0*. The *lr* parameter interval was decreased from [1e-4, 1.0e-0] to [1e-4, 1.0e-2], because all“good”solutions lie in this drastically smaller interval.
810 networks were evaluated, where 7.41% did not fit in the memory of the GPU. Table 7 shows the train time, validation loss and validation accuracy of the Pareto optimal solutions. Table 24 in the appendices shows the file name information of the data file and the file name of the construction script used in this experiment. The data file can be found online [13]. Figure 10 shows a plot of the train time and validation accuracy of all evaluated networks.

Time (s)	Loss	Acc
451.33	2.16	0.11
469.70	0.50	0.61
580.81	0.49	0.62
717.48	0.46	0.63
822.89	0.39	0.68
1093.18	0.37	0.69
1235.70	0.31	0.73
1643.25	0.28	0.75
2701.50	0.27	0.76
2938.25	0.25	0.78
5448.46	0.22	0.81

Table 7: Paretofront of the "Fine-tuned search space" experiment each entry is trained for ten epochs

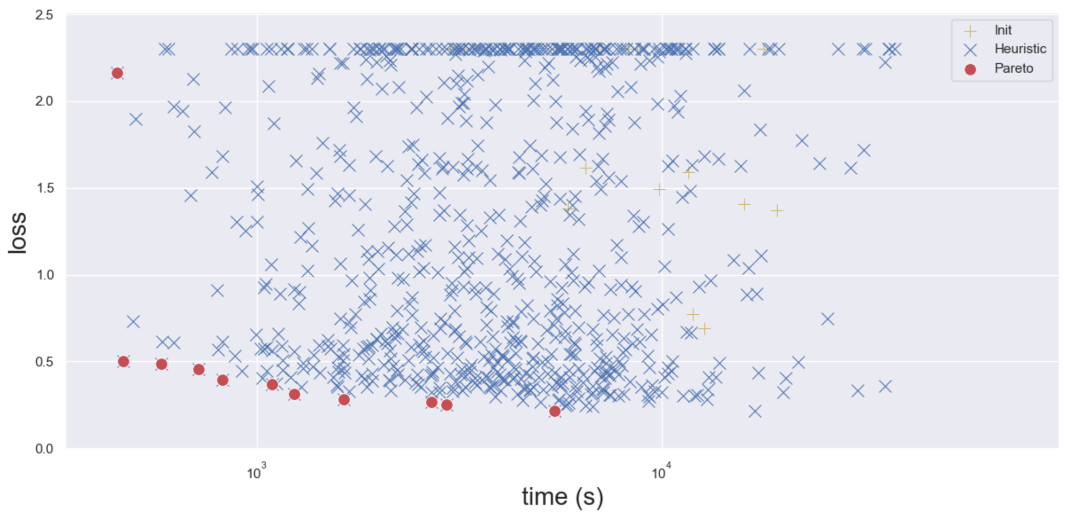


Figure 10: the "Fine-tuned search space" experiment on CIFAR10. 810 samples taken, using ten evaluation epochs, ten trees in the random forest

7.1 Data analysis

In order to be able to analyze the data, the data was split on a validation accuracy of 0.7 into a group of "good" networks with an accuracy greater or equal to 0.7 and a group of "bad" networks with an accuracy smaller than 0.7. The "Fine-tuned search space" experiment results contains 65 "good" networks and 685 "bad" networks.

7.1.1 High vs. low accuracy

In order to discover trends in "good" versus "bad" networks, a box plot was created for both the "good" and "bad" sets, using Pandas' box plot method [2]. This can be seen in the figures 11, 12, 13 and 14 . In these figures, green boxes indicate the distribution for "good" networks, red for "bad" networks. The boxes indicate the $Q1$ to $Q3$ quartile, with a line in the middle at $Q2$. The whiskers extend by $1.5 * (Q3 - Q1)$ from the edges of the box. Values that lie beyond these whiskers are considered to be outliers, and are not plotted. Boolean values are casted to float point values, where "True" is 1.0 and "False" is 0.0. The following trends were found:

- The "good" networks tend to have a larger average kernel size (*avg_kernel_size*) and a larger number of trainable features (*num_features*).

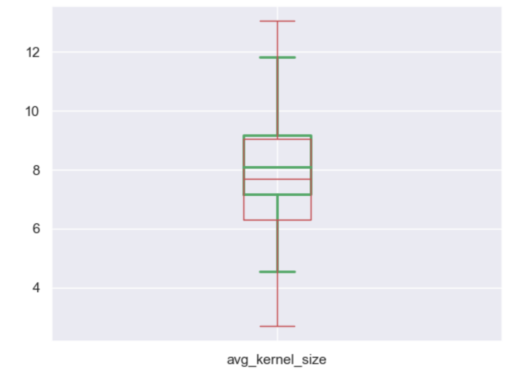
- A longer training time per epoch seems to yield higher accuracy (*time*). The "good" networks all have a *dropout_0* value close to zero, indicating it is not a good idea to add a dropout layer right after the input layer. This will not make the network more robust, it will just make it blind.

- The average dropout (*avg_dropout*) over all layers is significantly lower among the "good" networks. This is partly due to the dropout in layer zero being ideally close to zero, but in other dropout layers, the "good" networks also tended to have less dropout than the "bad" networks. This indicates that the best dropout rate is in the lower sections of our search space. Among the "good" networks, it averages around 0.23, indicating this is the optimal value.

- The *l2* kernel regularization tends to be close to zero in the "good" networks, averaging around 0.0001. This shows that a low value is preferred for a high accuracy. It might mean it is better not to do *l2* kernel regularization at all.

- All the "good" networks use the elu activation function [6], as can be seen by the boolean value *elu* being True for all "good" networks. It indicates that this function works best for this dataset and the general network architecture.

- The "good" networks tend to have a higher learning rate (*lr*). This indicates the ideal learning rate lies in the higher regions of the search space, averaging around 0.0079.



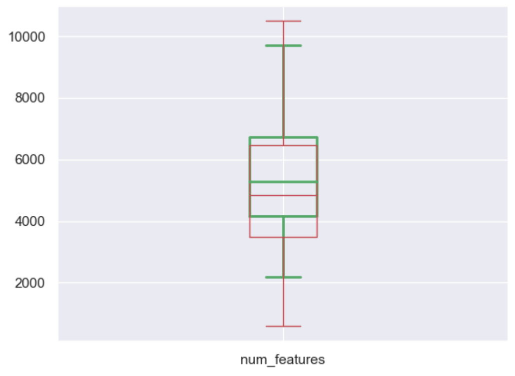
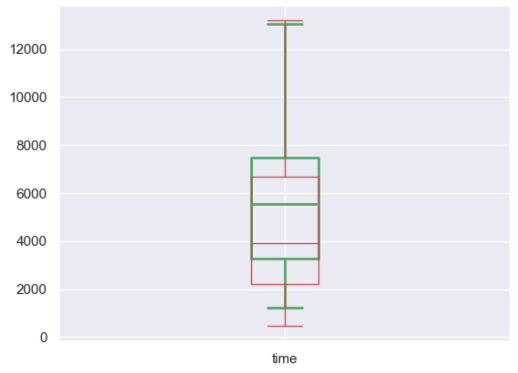


Figure 11: Fine-tuned search space experiment: Comparison of parameters between "good" (accuracy ≥ 0.7) in green and "bad" (accuracy < 0.7) in red.



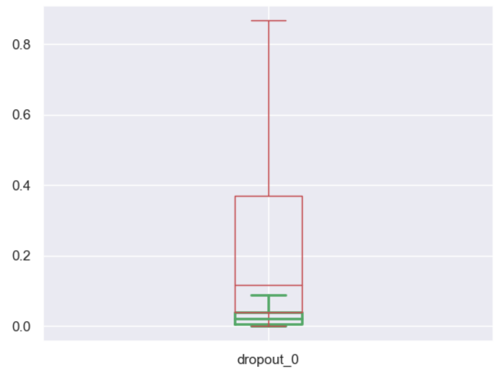
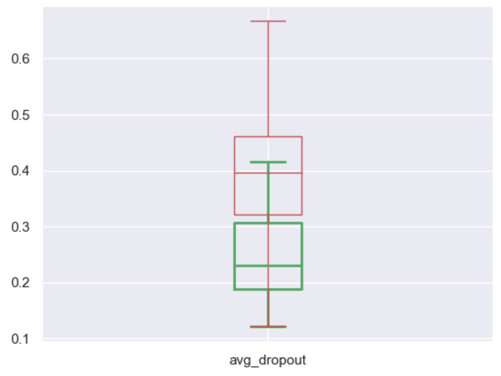


Figure 12: Fine-tuned search space experiment: Comparison of parameters between "good" (accuracy ≥ 0.7) in green and "bad" (accuracy < 0.7) in red.



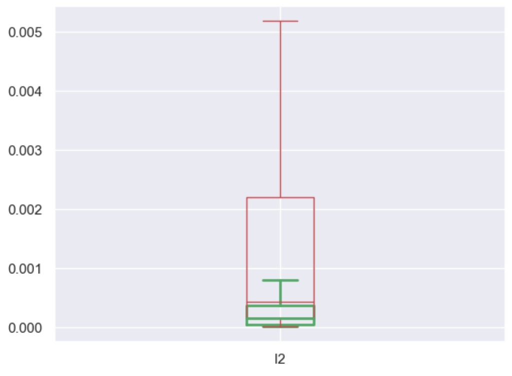


Figure 13: Fine-tuned search space experiment: Comparison of parameters between "good" (accuracy ≥ 0.7) in green and "bad" (accuracy < 0.7) in red.

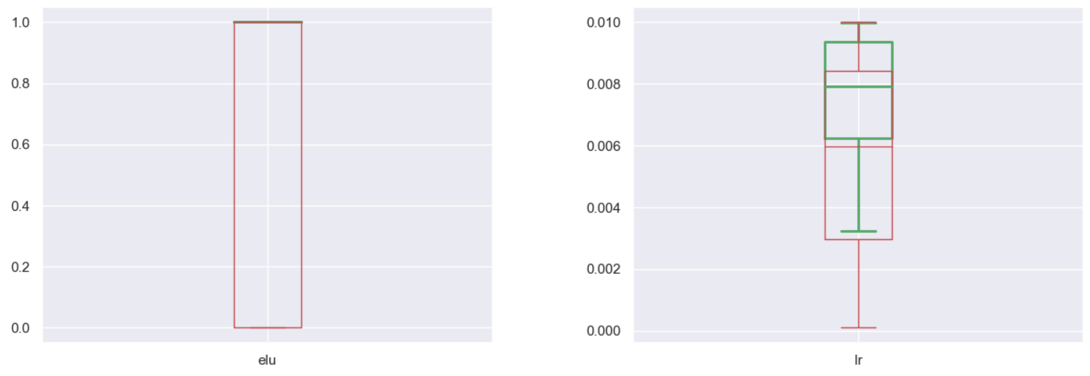


Figure 14: Fine-tuned search space experiment: Comparison of parameters between "good" (accuracy ≥ 0.7) in green and "bad" (accuracy < 0.7) in red.

Feature	Importance
activation	0.31
dropout_9	0.09
skstep_3	0.05
dropout_7	0.04
dropout_5	0.04
dropout_8	0.03
dropout_6	0.03
dropout_4	0.02
dropout_0	0.02
skstep_1	0.02

Table 8: Top ten feature importance of the "Fine-tuned search space" experiment

7.1.2 Feature importance

In order to get more insight into how a surrogate model predicts the validation accuracy based upon a MI vector of parameters, a feature importance analysis is performed. If a parameter has a high feature importance, it is considered to greatly influence the performance of a network. On the data of the "Fine-tuned search space" experiment, a random forest was built with 1000 trees, using the parameters of the MI vector as input and the loss value as output. Table 8 shows which parameters are considered to be the most relevant by the random forest model, trained on validation accuracy. The activation function was the most indicative of the performance. This was to be expected, since all "good" networks of the "Fine-tuned search space" experiment used the *elu* activation function. The other important parameters are mostly dropout values, indicating that correctly setting the dropout is important for the performance of a network. The skip-step three and one values are also considered to be important. The fact that skip-step three is considered to be more important than skip-step one must be due to random chance, because they both control the step size of an arbitrary skip connection chain.

8 Pruning one node layers experiment

Configurations tended to reduce the feature dimensions to one by one before the network reached its fully connected part. Technically this is no convolution anymore and actually the same as a fully connected layer, which is undesirable. Therefore, the "Pruning one node layers" method was implemented. This method cuts the one by one part from the network and immediately attaches the dense fully connected part.

8.1 Pre-experiment

A test run was done on the configuration denoted by table 9. The network without the cut operation can be seen in figure 15 and with the cut operation in figure 16. Both networks were trained for 20 epochs. The network with the cut operation reached an accuracy of 0.61 and a validation accuracy of 0.63 in 1734.69 seconds. The network without the cut operation reached an accuracy of 0.61 and a validation accuracy of 0.61 in 1659.75 seconds. Since the cut operation did not seem to impact the performance it was considered to be fit for testing. The goal was to see if using the cut operation resulted in higher accuracy and lower memory requirement, which would decrease the percentage of networks that would not fit in memory.

8.2 Main experiment

After testing if the "Pruning one node layers" method produces reasonable results, an experiment was performed. As an extra feature, the epochs to train in the evaluation function (*epoch_sp*) and the batch size (*batch_size_sp*) were added as parameters in the search space. Epochs were varied between five and 25. The batch size was varied between 50 and 200. In order to prevent any kind of information leakage, the CIFAR10 data set was split into a test set, a train set and a validation set. The original train set was split into a train set and a validation set.The validation set consist of 2000 samples picked from the original train set. It contained roughly the same amount of samples per class. Networks were trained on the train set. SMS-MIP-EGO optimized on test set accuracy and training time. Separately from this experiment, the network with the highest accuracy was trained again and evaluated on the validation set, as is presented in section 12. All the following experiments use this threefold separation of data sets.
For the experiment 804 networks were evaluated, where 6.09% did not fit in the memory of the GPU. This was a small decrease from the 7.41% that did not fit in memory during the "Fine-tuned search space" experiment. From this small difference, it can be concluded that the "Pruning one node layers" method does not significantly reduce the memory needed compared to the standard *Skippy* method. On the other hand, data analysis in subsection 8.3 shows that the networks generated by the "Pruning one node layers" method have a significantly smaller number of features. However, the features that were cut from the network by the "Pruning one node layers" method all had a size of 1×1, meaning they used almost no memory in the first place. Table 11 shows the training time, validation loss and validation accuracy of the Pareto optimal solutions. Table 25 in the appendices shows the file name information of the data file and the file name of the construction script used in this experiment. The data file can be found online [13]. Figure 17 shows the training time and validation accuracy of the evaluated networks.

Paramter	Value
stack_0	2
stack_1	4
stack_2	2
stack_3	6
stack_4	6
stack_5	4
stack_6	0
s_0	3
s_1	7
s_2	7
s_3	1
s_4	6
s_5	1
s_6	9
filters_0	108
filters_1	374
filters_2	55
filters_3	579
filters_4	409
filters_5	112
filters_6	246
filters_7	350
filters_8	554
filters_9	502
filters_10	351
filters_11	493
filters_12	239
filters_13	141
k_0	8
k_1	10
k_2	1
k_3	8
k_4	5
k_5	11
k_6	13
k_7	5
k_8	12
k_9	9
k_10	14
k_11	4
k_12	15
k_13	3
activation	'tanh'
activ_dense	'softmax'
dropout_0	0.12495090785008392
dropout_1	0.00244978236390256
dropout_2	0.10503529076687178
dropout_3	0.4258548492846925
dropout_4	0.5454956319314442
dropout_5	0.7148625620753165
dropout_6	0.09603825810959132
dropout_7	0.7656562067402091
dropout_8	0.6713796132485909
dropout_9	0.3676723784593959
lr	0.004860286083081829
l2	0.0018527840854743725
step	False
global_pooling	False
skstart_0	6
skstart_1	1
skstart_2	1
skstart_3	3
skstart_4	2
skstep_0	9
skstep_1	4
skstep_2	3
skstep_3	5
skstep_4	8
max_pooling	True
dense_size_0	3884
dense_size_1	3232
batch_size_sp	170
epoch_sp	33

Table 9: Parameters of the network used to compare using the “Pruning one node layers” method versus not using it

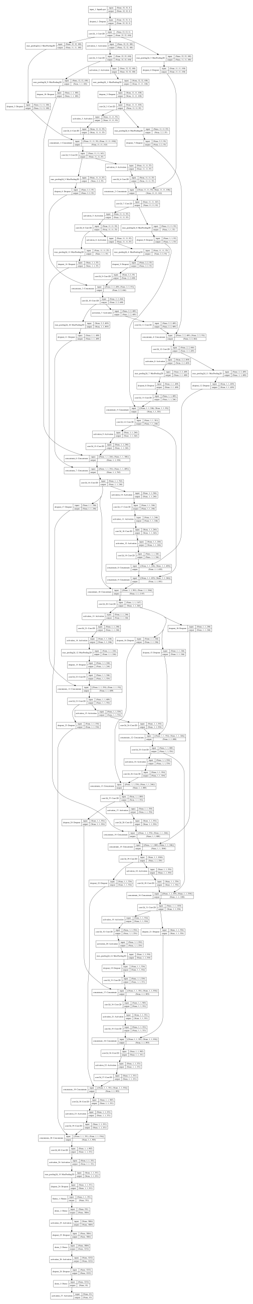


Figure 15: network without non-convolutional part cut

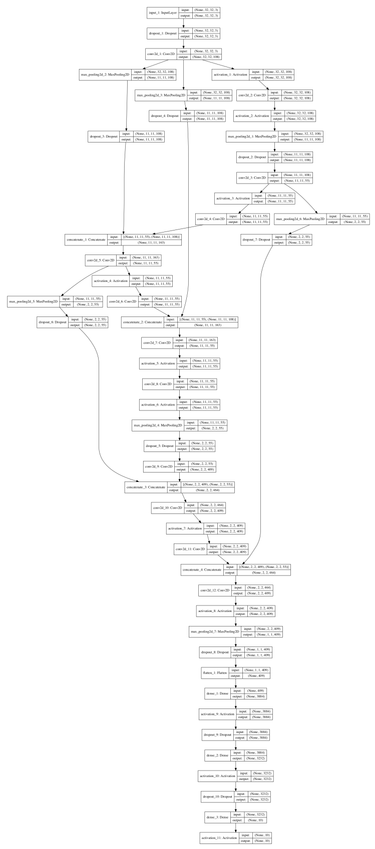


Figure 16: network with non-convolutional part cut

Parameter	Type	Bounds	# Dimensions
filters	Discrete	[10, 600]	14
kernel_size	Discrete	[1, 16]	14
strides	Discrete	[1, 10]	7
stack_sizes	Discrete	[0, 7]	7
activation	Nominal	[”elu”,”relu”,”tanh”,”sigmoid”,”selu”]	1
activation_dense	Nominal	[”softmax”]	1
step	Nominal	[True, False]	1
global_pooling	Nominal	[True,False]	1
skstart	Discrete	[0, 7]	5
skstep	Discrete	[1, 10]	5
max_pooling	Nominal	[True, False]	1
dense_size	Discrete	[0,4000]	2
drop_out	Continuous	[1e-5, 0.9]	10
lr	Continuous	[1e-4, 1.0e-2]	1
l2_regularizer	Continuous	[1e-5, 1e-2]	1
epoch_sp	Discrete	[5, 25]	1
batch_size_sp	Discrete	[50, 200]	1

Table 10: search space of the “Pruning one node layers” experiment

Time (s)	Loss	Acc
90.84	2.02	0.13
99.43	0.72	0.49
159.84	0.60	0.55
217.65	0.46	0.63
258.27	0.45	0.64
408.72	0.43	0.65
451.312	0.41	0.67
461.42	0.40	0.67
529.44	0.40	0.67
563.65	0.33	0.72
1189.82	0.27	0.76
1890.37	0.25	0.78
6998.60	0.25	0.78
9858.69	0.24	0.78
12237.98	0.22	0.80

Table 11: Paretofront of the "Pruning one node layers" experiment. Training epochs vary per entry.

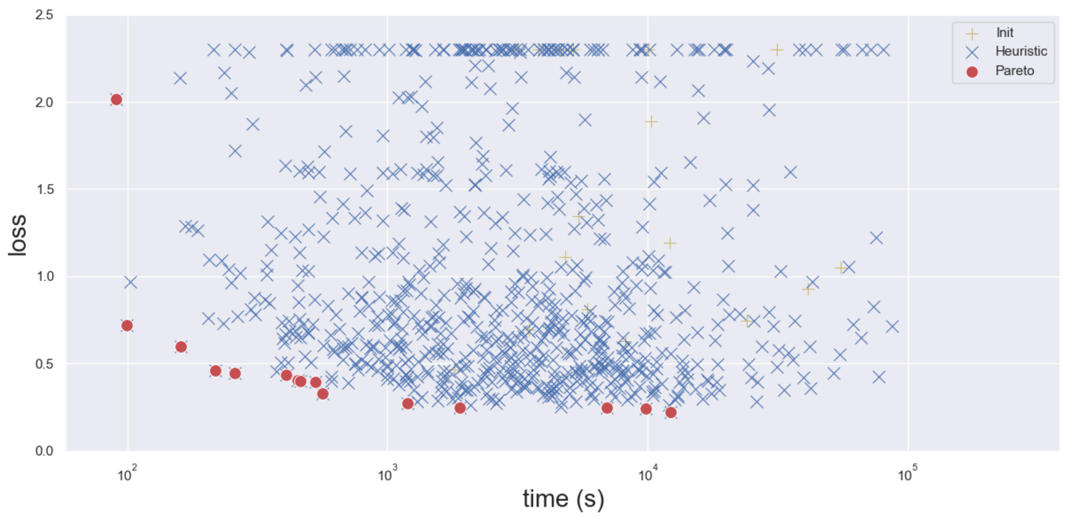


Figure 17: The "Pruning one node layers" experiment on CIFAR10. 804 samples taken, using epochs as search parameter (5 - 25), using batch size as search parameter (50 - 200), ten trees in the random forest

8.3 Data analysis

The data was split into a set "good" and "bad" networks on an evaluation accuracy of 0.7. An evaluation accuracy greater or equal to 0.7 was considered to be "good", smaller than 0.7 was considered to be "bad". It resulted in 57 "good" networks and 698 "bad" networks.
The addition of adding a variable batch size and epochs to the search space did not seem to impact the performance. It did not improve the hyper volume as can be seen in Table 22. However it must be mentioned that the training set is now smaller, due to the evaluation set being split from it.

8.3.1 High vs. low accuracy

All networks had a zero layer size from filters eight to thirteen, meaning there were no networks with four to six stacks. This was due to all networks reaching a feature size of 1×1 before this layer, thus instructing the "Pruning one node layers" method to immediately attach the dense layers and output layer. Since the last layers were never used, this method created a loss of diversity. Therefore in the following experiment, the "Pruning one node layers" method was not used, but instead the maximum possible stride value was reduced.
Figures 18, 19, 20, 21, 22 and 23 show a box plot of feature values of the "good" and "bad" networks. The following trends were found:

- The *avg_dropout* tends to be lower for the "good" networks, indicating that the optimal average dropout value is at the lower side of the search space. The average for the "good" networks is around 0.24, indicating this is the optimal value. For the "Fine-tuned search space" experiment, the average was around 0.23, which is very close to 0.24. This indicates that this dropout value might be a good rule of thumb.

- The *avg_kernel_size* is almost identical among the "good" and "bad" networks. It indicates that the average kernel size does not seem to matter much for the performance of a network. In the "Fine-tuned search space" experiment, a slightly higher average kernel size seemed to benefit accuracy.

- The *batch_size_sp* value is slightly lower among the "good" networks. It averages at around 120. For the "bad" networks, it averages around 130.

- The *dropout_0* value tends to be around zero for the "good" networks. This indicates that adding a dropout layer directly after the input is not a good idea. The "Fine-tuned search space" experiment showed the same result.

- The *elu* activation function is used by the "good" and "bad" networks. This box plot does not indicate any trends, where the "Fine-tuned search space" experiment showed that the elu activation function was to be preferred.

- The *epoch_sp* value tends to be higher among the "good" networks. This was to be expected, since more epochs means more training opportunity, thus a higher accuracy.

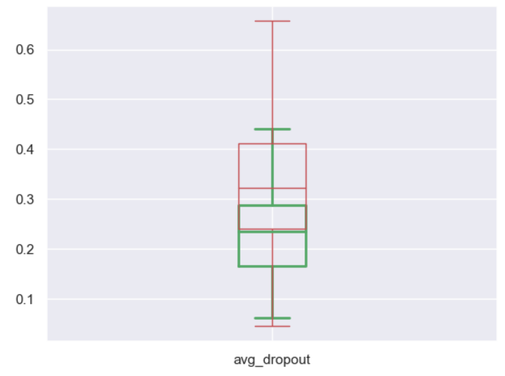
- The *l2* kernel regularization was lower among the "good" networks, indicating that there was an optimal value on the lower side of the search space. The average value among the "good" networks lies around 0.0016, indicating this is the optimal value. In the "Fine-tuned search space" experiment it averaged even lower around 0.0001 for the "good" networks. These results indicate that l2 kernel regularization might not be needed, and if needed, its value should be close to zero.

- The *lr* (learning rate) value was slightly higher among the "good" networks. It averaged around 0.0066 among the "good" networks, indicating an optimal value. In the previous "Fine-tuned search space" experiment, it averaged around 0.0079 among the "good" networks.

- All "good" networks used the *max_pooling* option. This means Max pooling layers have an added benefit over just using 2D convolutions in a network.

- The *num_features* range was smaller among the "good" networks. Its average was 2000 among the "good" networks. In the previous "Fine-tuned search space" experiment, the number of features was slightly higher among the "good" solutions versus the "bad" solutions. The "good" solutions' number of features averaged around 5400 in the "Fine-tuned search space" experiment, more than twice the amount of features as the average amount of features in the "good" solutions of the "Pruning one node layers" experiment. This was to be expected, since the "Pruning one node layers" method cuts all features of size 1×1. Since these features do not take up a lot of memory in the first place, the "Pruning one node layers" method unfortunately does not save a lot of memory.

- The *time* values were greater among the "good" networks. This indicates that more training time means higher accuracy. This can be partly explained by the fact that more epochs tend to lead to a higher accuracy and that more epochs require more training time. However, the "Fine-tuned search space" experiment shows higher accuracy for longer training times even when the number of epochs is fixed in this experiment.



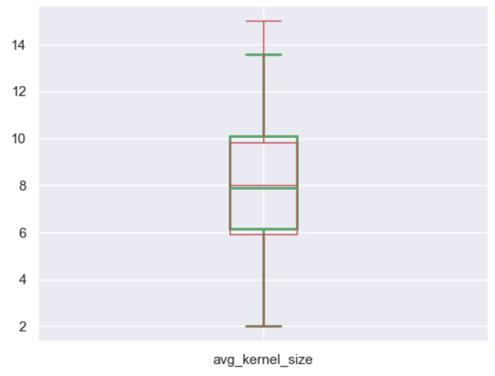
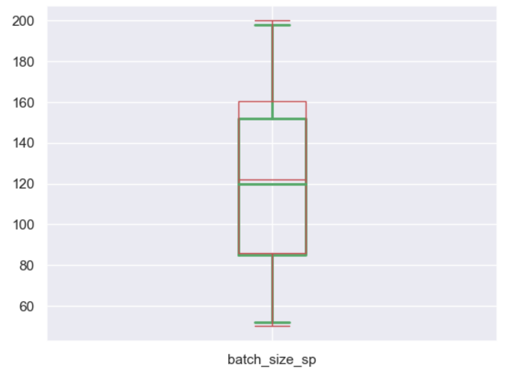


Figure 18: Pruning one node layers: Comparison of parameters between “good” (accuracy ≥ 0.7) in green and “bad” (accuracy < 0.7) in red.



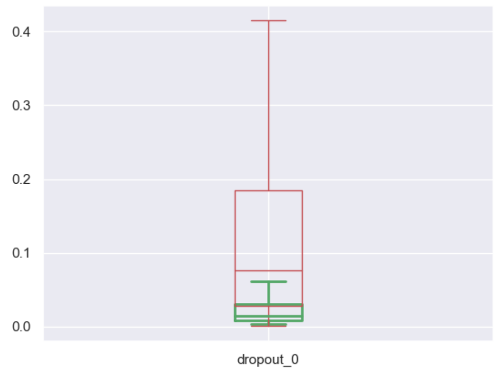
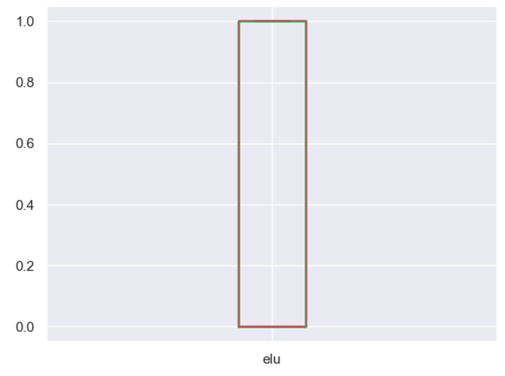


Figure 19: Pruning one node layers: Comparison of parameters between “good” (accuracy ≥ 0.7) in green and “bad” (accuracy < 0.7) in red.



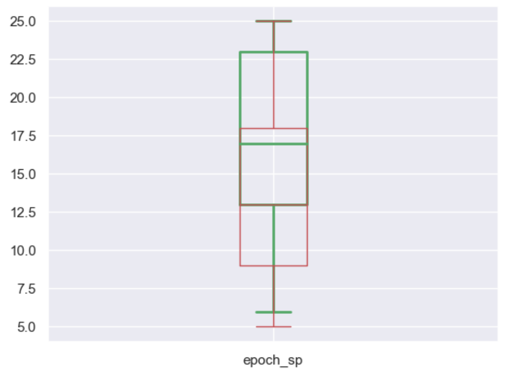


Figure 20: Pruning one node layers: Comparison of parameters between “good” (accuracy ≥ 0.7) in green and “bad” (accuracy < 0.7) in red.

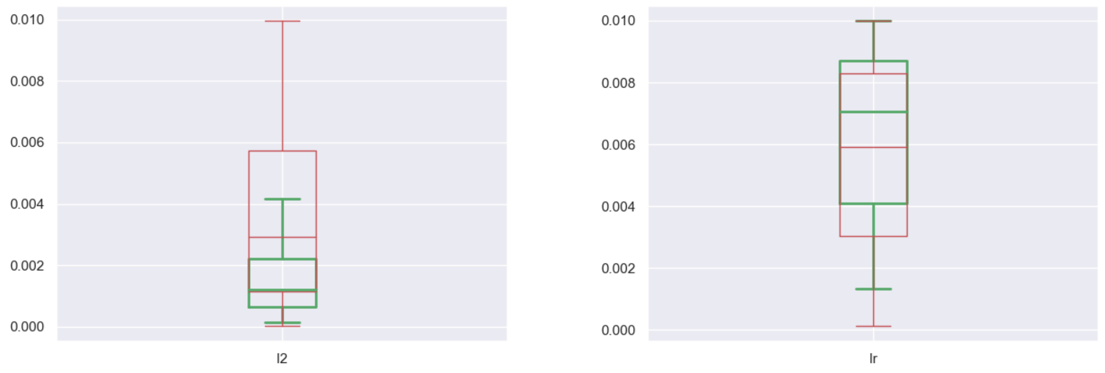


Figure 21: Pruning one node layers: Comparison of parameters between "good" (accuracy ≥ 0.7) in green and "bad" (accuracy < 0.7) in red.

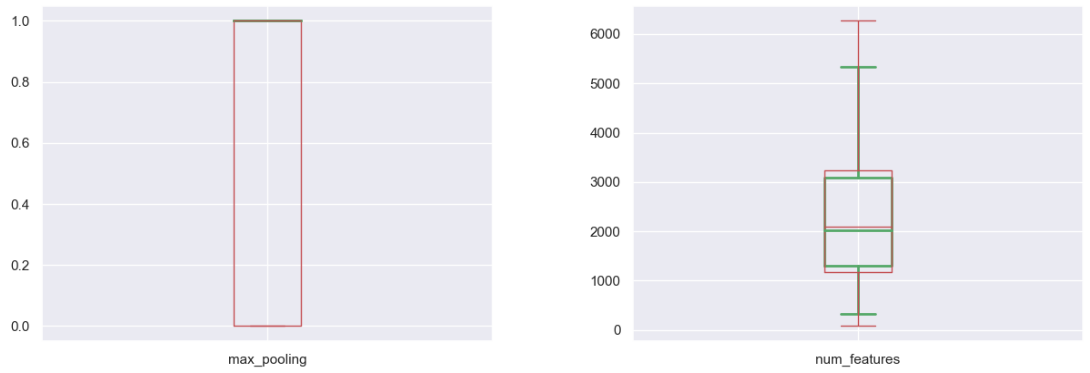


Figure 22: Pruning one node layers: Comparison of parameters between "good" (accuracy ≥ 0.7) in green and "bad" (accuracy < 0.7) in red.

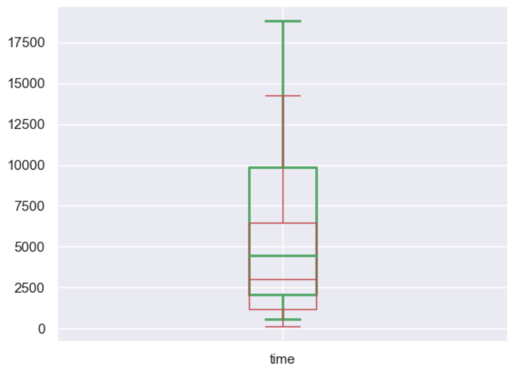


Figure 23: Pruning one node layers: Comparison of parameters between "good" (accuracy ≥ 0.7) in green and "bad" (accuracy < 0.7) in red.

Feature	Importance
activation	0.17
dropout_9	0.07
skstep_3	0.04
dropout_0	0.03
dropout_7	0.03
dropout_1	0.03
dropout_6	0.03
filters_11	0.02
filters_3	0.02
dropout_3	0.02

Table 12: Top ten feature importance of the “Pruning one node layers” experiment

8.3.2 Feature importance

A random forest was trained with 1000 trees on the data of the “Pruning one node layers” experiment, using the parameters of the MI vector as input and the loss value as the training goal. Table 12 shows the top ten feature importance. Just like in the “Fine-tuned search space” experiment, the activation function is considered to be the most important. After this, the choice for the dropout values is most important, just like in the “Fine-tuned search space” experiment. Apart from the activation function, all importance values are small. Because it did not add to any gained insights, it is not used further on.

9 Comparison with RESnet-30

The proposed method was compared with a state-of-the-art RESnet-30 architecture that is publicly available [23]. The best network from the “Fine-tuned search space” experiment and the state-of-the-art RESnet-30 network were trained for 200 epochs and their results were compared as can be seen in figure 24. For the first 80 epochs, the best network from the “Fine-tuned search space” experiment performs similarly to the state-of-the-art RESnet-30 network. At around 80 epochs, the state-of-the-art network jumps up significantly in performance. This is at the same time when the learning rate drops due to its training schedule. The state-of-the-art network used data augmentation, where the best network from the “Fine-tuned search space” experiment does not. Also the “Fine-tuned search space” experiment evaluated its networks for only ten epochs. It is highly unlikely that it could predict a jump at 80 epochs. Therefore two last experiments were proposed. In the first step, called the “Data augmentation experiment”, data augmentation parameters were added to the search space. In the second step, called the “Training schedule optimization” experiment, the network with the highest validation accuracy was selected and this network’s training schedule was optimized and trained for 100 epochs. SMS-MIP-EGO was used as an optimizer.

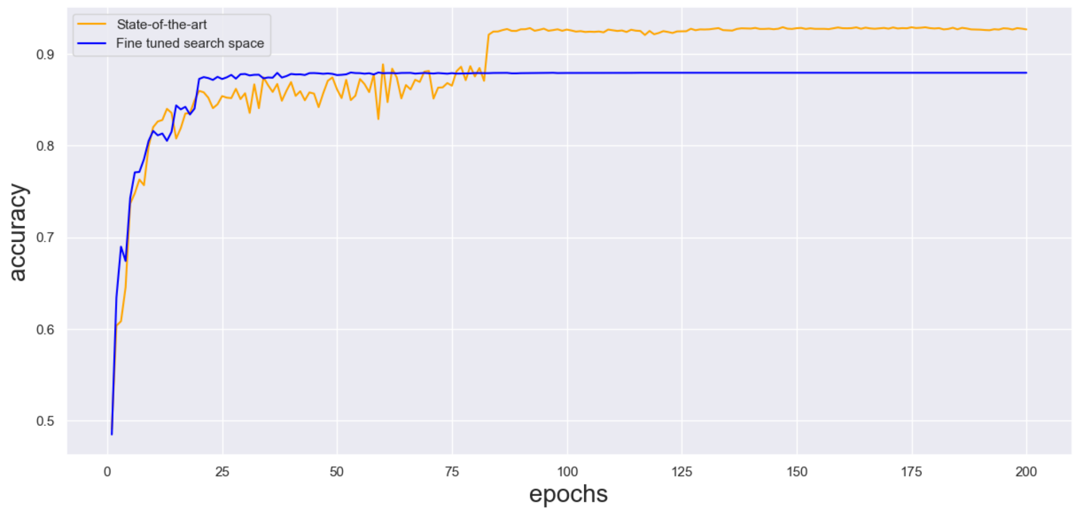


Figure 24: Comparison of state-of-the-art RESnet-30 [23] and the best network from the "Fine-tuned search space" experiment, trained for 200 epochs

10 Data augmentation experiment

In the "Pruning one node layers" experiment, almost no networks used the last stacks. This was a waste of possible diversity per stack, since the network could also be made shallower by decreasing the stack sizes. Therefore the maximum stride was decreased to four. The maximum value for dropout was capped at 0.42 because in the "Pruning one node layers" experiment, all "good" networks, except for the outliers, had an average dropout in this range. Because training for more epochs resulted in higher accuracy in the "Pruning one node layers" experiment, it was harder to distinguish which architecture was better suited for training and which accuracy was simply trained for more epochs. Therefore, the *epoch_sp* parameter was removed from the search space.

Table 13 explains the features that were added to the search space in order to perform data augmentation. Table 14 shows the ranges of all parameters that make up the search space. Note that the *featurewise_center*, *samplewise_center*, *featurewise_std_normalization* and *samplewise_std_normalization* parameters are always False. This is done because it interfered with the normalization that was already in place and resulted in low accuracy

In this experiment, 807 solutions were chosen to be evaluated for ten epochs. 13.14% of the solutions did not fit in memory and were not evaluated.

Table 15 shows the training time, evaluation loss value and evaluation accuracy of the Pareto optimal solutions. Table 26 in the appendices shows the file name information of the data file and the file name of the construction script used in this experiment. The data file can be found online [13]. Figure 25 shows a plot of the training time and the evaluation accuracy of all evaluated networks.

Parameter	Explanation
featurewise_center	Set input mean to 0 over the dataset, feature-wise.
samplewise_center	Set each sample mean to 0.
featurewise_std_normalization	Divide inputs by std of the dataset, feature-wise.
samplewise_std_normalization	Divide each input by its std.
zca_epsilon	epsilon for ZCA whitening.
zca_whitening	Apply ZCA whitening.
rotation_range	Degree range for random rotations.
width_shift_range	fraction of total width, if < 1, or pixels if ≥ 1.
height_shift_range	fraction of total height, if < 1, or pixels if ≥ 1.
shear_range	Shear Intensity (Shear angle in counter-clockwise direction in degrees)
zoom_range	Range for random zoom. [lower, upper] = [1-zoom_range, 1+zoom_range].
channel_shift_range	Range for random channel shifts.
fill_mode	One of {"constant", "nearest", "reflect" or "wrap"}.
cval	Value used for points outside the boundaries when fill_mode = "constant".
horizontal_flip	Randomly flip inputs horizontally.
vertical_flip	Randomly flip inputs vertically.

Table 13: Extra search space parameters for the "Data augmentation" experiment explanation, which uses the Keras' "keras.preprocessing.image.ImageDataGenerator" function. Explanations are according to the Keras documentation [1].

Parameter	Type	Bounds	# Dimensions
filters	Discrete	[10, 600]	14
kernel_size	Discrete	[1, 16]	14
strides	Discrete	[1, 4]	7
stack_sizes	Discrete	[0, 7]	7
activation	Nominal	["elu","relu","tanh","sigmoid","selu"]	1
activation_dense	Nominal	["softmax"]	1
step	Nominal	[True, False]	1
global_pooling	Nominal	[True,False]	1
skstart	Discrete	[0, 7]	5
skstep	Discrete	[1, 10]	5
max_pooling	Nominal	[True, False]	1
dense_size	Discrete	[0,4000]	2
drop_out	Continuous	[0.0, 0.42]	10
lr	Continuous	[1e-4, 1.0e-2]	1
l2_regularizer	Continuous	[1e-5, 1e-2]	1
batch_size_sp	Discrete	[50, 200]	1
featurewise_center	Nominal	[False]	1
samplewise_center	Nominal	[False]	1
featurewise_std_normalization	Nominal	[False]	1
samplewise_std_normalization	Nominal	[False]	1
zca_epsilon	Continuous	[0.5e-6, 2e-6]	1
zca_whitening	Nominal	[True,False]	1
rotation_range	Ordinal	[0, 360]	1
width_shift_range	Continuous	[0.0, 1.0]	1
height_shift_range	ContinuousSpace	[0.0, 1.0]	1
shear_range	Continuous	[0.0, 45.0]	1
zoom_range	Continuous	[0.0, 1.0]	1
channel_shift_range	Continuous	[0.0, 1.0]	1
fill_mode	Nominal	["constant","nearest","reflect","wrap"]	1
cval	Continuous	[0.0, 1.0]	1
horizontal_flip	Nominal	[True,False]	1
vertical_flip	Nominal	[True,False]	1

Table 14: search space of the "augmented" experiment

Time (s)	Loss	Acc
754.83	1.85	0.16
891.18	1.19	0.31
1072.66	0.81	0.44
1574.26	0.46	0.63
6163.14	0.35	0.71

Table 15: Paretofront of the "Data augmentation" experiment each entry is trained for ten epochs

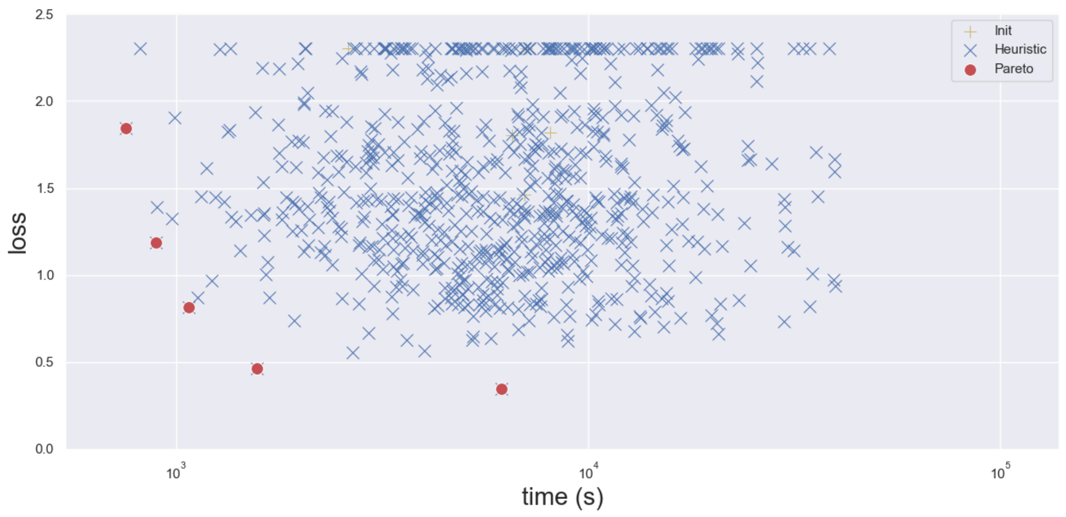


Figure 25: The "Data augmentation" experiment on CIFAR10. 807 samples taken, using batch size as search parameter (50 - 200), ten trees in the random forest

10.1 Data analysis

The configurations of the "Data augmentation" experiment were split on a validation accuracy of 0.4. Configurations with a validation accuracy greater or equal to 0.4 were considered to be "good" solutions, while those with a validation accuracy smaller than 0.4 were considered to be "bad" solutions. This split the resulted in 77 "good" solutions and 624 "bad" solutions.

10.1.1 High vs. low accuracy

Figure 26, 27, 28, 29, 30, 31, 32, 33, 34, 35 and 36 show box plots of these features. The following trends were found:

- The *avg_kernel_size* is slightly smaller in the "good" solutions, but this is insignificant. The average of the average kernel size per network lies around 7.5. In the "Fine-tuned search space" experiment, "good" solutions tended to have slightly bigger kernel sizes than the "bad" solutions, averaging around 8 for the "good" networks. In the "Pruning one node layers" experiment, average kernel size was almost equal for "good" and "bad" networks, averaging around 8 for the "good" networks. It can be concluded that there is no clear optimal kernel size that can lead to a high accuracy. In the experiments, the average kernel size tends to be around 8.

- The *num_features* is slightly larger among the "good" networks, averaging around 5400. Indicating more features in a network improve the accuracy. The same result can be seen in the "Fine-tuned search space" experiment. In this experiment, it averaged around 5400. The "Pruning one node layers" experiment, this result is not present. In this experiment, it averaged around 2000. It can be concluded that there might be a positive correlation between the number of features and the validation accuracy.

- The *time* value tends to be slightly larger among the "good" solutions, although the average training time value is almost the same among the "good" and "bad" solutions. Since the amount of epochs trained for is the same for all networks, this parameter indicates how long it takes to train for each epoch. It is to be expected that a network with more trainable features will yield better results and also needs more training time per epoch. Therefore it is to be expected that a longer training time per epoch will yield better results on average. The "Fine-tuned search space" experiment and "Pruning one node layers" experiment also show more training time leading to a higher accuracy. Therefore it can be concluded that more training time will lead to a higher accuracy, even when the number of epochs is fixed.

- The *dropout_0* value is close to zero among the "good" solutions. This result was also present in the "Fine-tuned search space" experiment and "Pruning one node layers" experiment. So even with data augmentation, it is good practice to omit a dropout layer between the input and the network. It can be concluded that it is bad practice to put a dropout layer directly after the input.

- The *avg_dropout* value is slightly lower in among the “good” solutions and centred around 0.16. This indicates this is the optimal value for most networks. In the “Fine-tuned search space” experiment this value was centred around 0.23, where it was centred around 0.24 for the “Pruning one node layers” experiment. This difference might be explained due to the fact that dropout protects for overfitting. However, data augmentation also protects from overfitting. Therefore when using data augmentation, less dropout is needed. From these experiments it can be concluded that it is advisable is to use dropout values around 0.23 as a start point for tweaking when not using data augmentation and around 0.16 when using data augmentation.

- The *l2* regularization value is centred around 0.001 for the “good” solutions. This is considered the optimal value for this experiment. In the “Fine-tuned search space” experiment, it was centred around 0.0001, while in the “Pruning one node layers” experiment it was centred around 0.0016. It can be concluded that l2 regularization should only be used with small values under 0.002, or that it should not be used at all.

- None of the “bad” solutions use the *elu* activation function. In the “Fine-tuned search space” experiment, all the “good” solutions used the elu activation function. The “Pruning one node layers” experiment showed no clear trend. However, there is an indication that the elu activation function works well with *Skippy*. It might mean that the elu activation function is the preferred function when building CNNs.

- The *lr* value is slightly higher among the “good” solutions. The learning rate value is centred around 0.006 for the “good” solutions. This is considered to be the optimal value for this experiment. For the “Fine-tuned search space” experiment, it centred around 0.0079, while for the “Pruning one node layers” experiment it centred around 0.0066. Thus, a learning rate of about 0.007 is a good starting point for this type of network and dataset.

- The *batch_size_sp* value is centred around 140 for the “good” solutions and around 130 for the “bad” solutions. For the “Pruning one node layers” experiment, the average value was centred around 120 for the “good” networks and around 130 for the “bad” networks. In conclusion, the batch size has little impact on the accuracy of a network.

- The *channel_shift_range* is lower among the “good” solutions. This indicates that channel shifting in the data augmentation should only be done with small values between zero and 0.05, or be omitted.

- No "good" solutions used the *constant* infill method. This indicates that using a constant value to fill in the gaps created by shearing an image is not a good idea. None of the "bad" solutions use the *nearest* infill value when using shearing, while some of the "good" do. This indicates that it not a bad idea to use the nearest pixel as an infill value when using a shear transformation in data augmentation. The *reflect* infill method is used by the "good" and the "bad" solutions, while no solutions used the *wrap* infill method. Thus, the nearest pixel or the reflect method can be used as an infill value, but avoid the constant value and do not use the wrap method.

- None of the "bad" solutions use *global_pooling*. Since the "Data augmentation" experiment uses a smaller stride range, the chance of features being reduced to a size of 1×1 pixels is small, in which case global pooling can help in reducing the complexity of the dense layers of the network. This result indicates that global pooling is also beneficial for the accuracy or a network. The "Fine-tuned search space" and "Pruning one node layers" experiment's networks did not show this trend. It can be concluded that it is good practice to reduce the feature size to 1×1 before connecting the dense layer, and if this is not the case, to use global pooling.

- The *height_shift_range* is cantered around 0.35 for the "good" networks, while at 0.45 for the "bad" solutions. This indicates that data augmentation works best for a height shift range between zero and 0.35.

- All of the "bad" solutions use the *horizontal_flip* data augmentation parameter, while some of the "good" solutions use it and some don't. This indicates that the random horizontal flipping of images is not necessary for high performance.

- All networks use the *max_pooling* layer, with the exception of some outliers, instead of a 2D convolutional layer with a stride larger than one for reducing the size of images. It is unclear why this is the case, but it might be that SMS-MIP-EGO senses this is good design practice. In the "Pruning one node layers" experiment, all "good" networks use Max pooling. These results challenge the findings of Springenberg et al. [18], that question the necessity of Max pooling layers.

- The *rotation_range* is centred around 100 for the "good" solutions and around 180 for the "bad" solutions. This indicates better results are yielded when images are rotated for smaller amounts, during data augmentation, preferably between zero and 100 degrees.

- All of the "bad" solutions use the *vertical_flip*, while some of the "good" solutions use it and some don't. This indicates that flipping images among the vertical axis during data augmentation is not needed for good results.

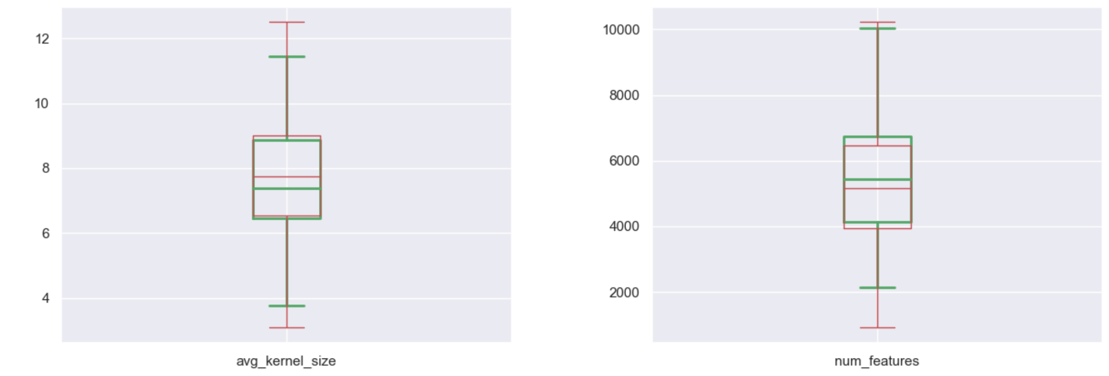


Figure 26: Data augmentation: Comparison of parameters between "good" (accuracy $\geq$ 0.4) in green and "bad" (accuracy < 0.4) in red.

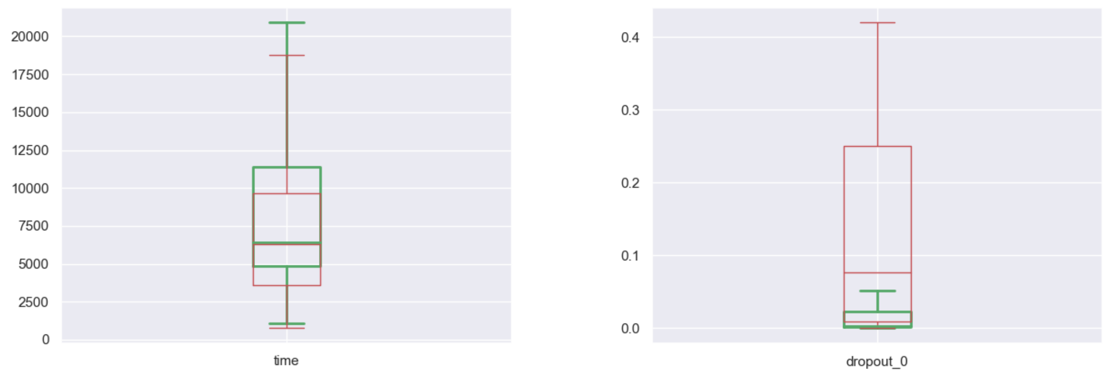


Figure 27: Data augmentation: Comparison of parameters between "good" (accuracy $\geq$ 0.4) in green and "bad" (accuracy < 0.4) in red.

- The *width_shift_range* is centred around 0.18 for the "good" solutions and around 0.5 among the "bad" solutions. This indicates that it is best not to shift the width too much when using data augmentation, ideally only between zero and 0.18.

- The *zoom_range* is centred around 0.35 for the "good" solutions, while it is centred around 0.55 for the "bad" solutions. This indicates that images should not be zoomed too drastically during data augmentation. Ideally they should stay within a range between zero and 0.35.

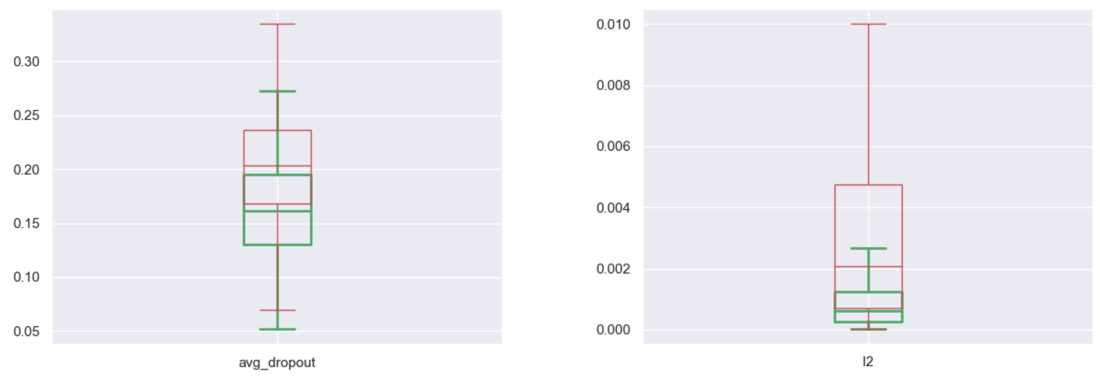


Figure 28: Data augmentation: Comparison of parameters between "good" (accuracy $\geq$ 0.4) in green and "bad" (accuracy $<$ 0.4) in red.

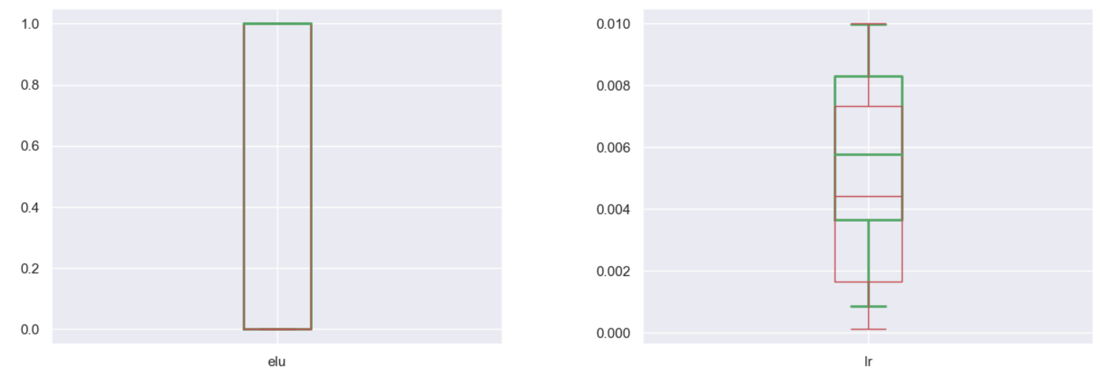


Figure 29: Data augmentation: Comparison of parameters between "good" (accuracy $\geq$ 0.4) in green and "bad" (accuracy $<$ 0.4) in red.

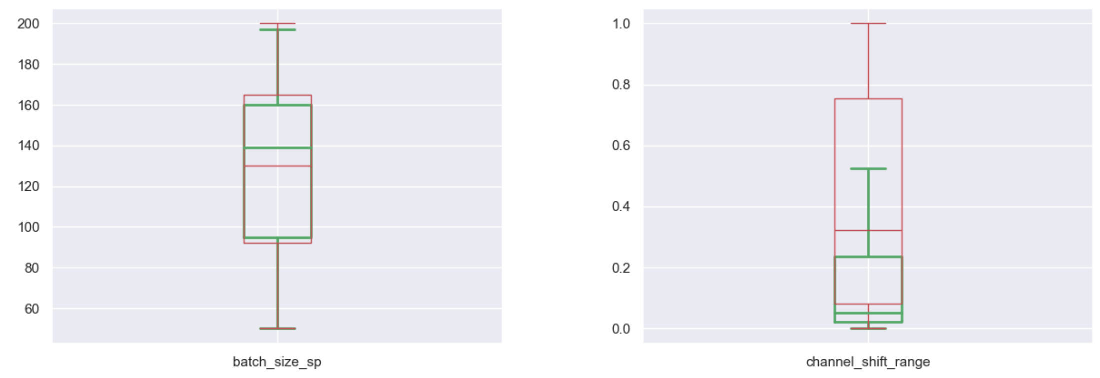


Figure 30: Data augmentation: Comparison of parameters between "good" (accuracy $\geq$ 0.4) in green and "bad" (accuracy $<$ 0.4) in red.

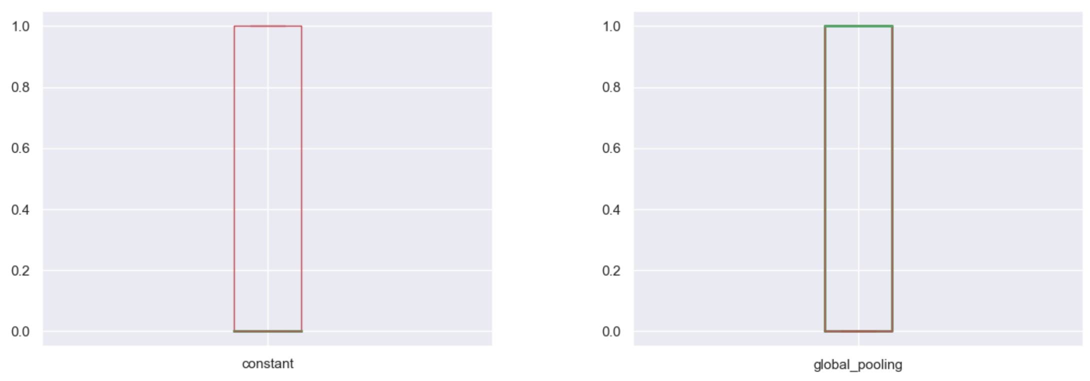


Figure 31: Data augmentation: Comparison of parameters between "good" (accuracy $\geq$ 0.4) in green and "bad" (accuracy < 0.4) in red.

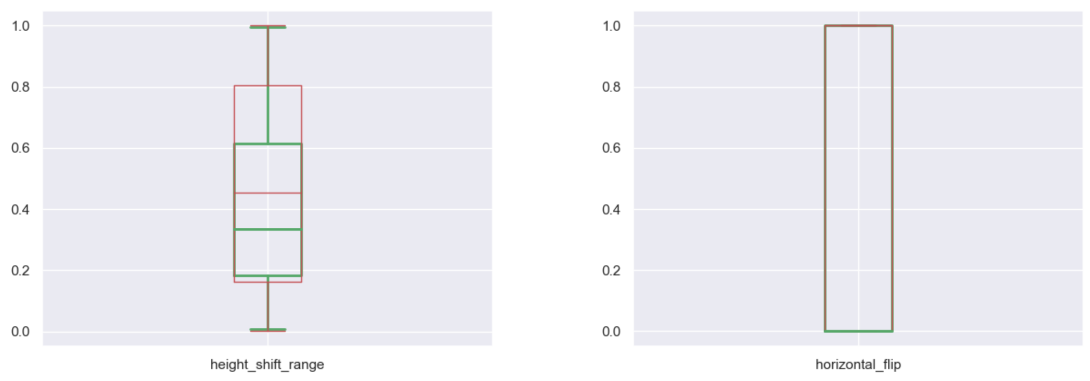


Figure 32: Data augmentation: Comparison of parameters between "good" (accuracy $\geq$ 0.4) in green and "bad" (accuracy < 0.4) in red.

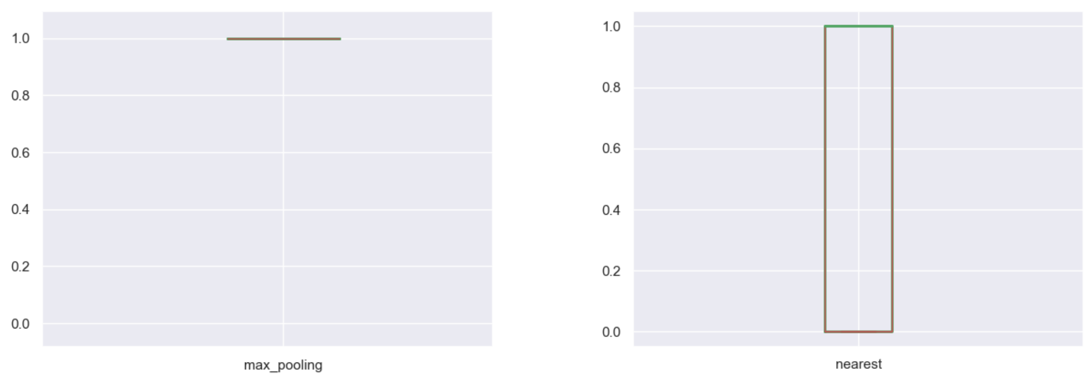


Figure 33: Data augmentation: Comparison of parameters between "good" (accuracy $\geq$ 0.4) in green and "bad" (accuracy < 0.4) in red.

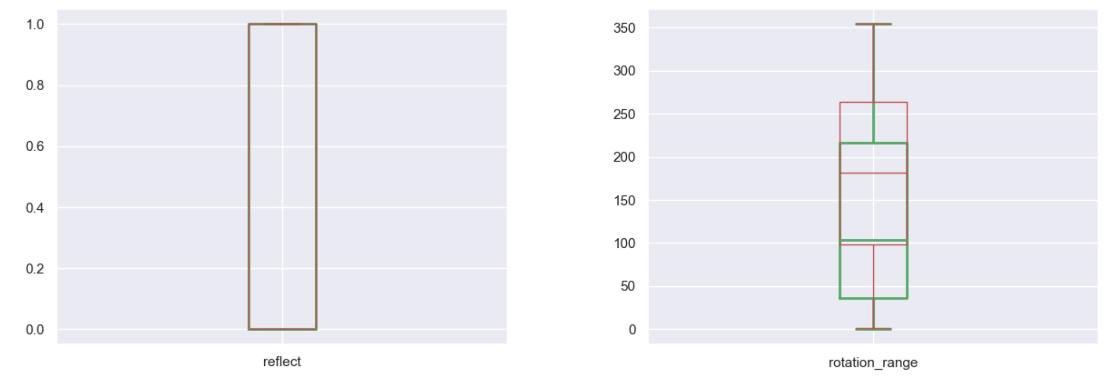


Figure 34: Data augmentation: Comparison of parameters between "good" (accuracy $\geq$ 0.4) in green and "bad" (accuracy $<$ 0.4) in red.

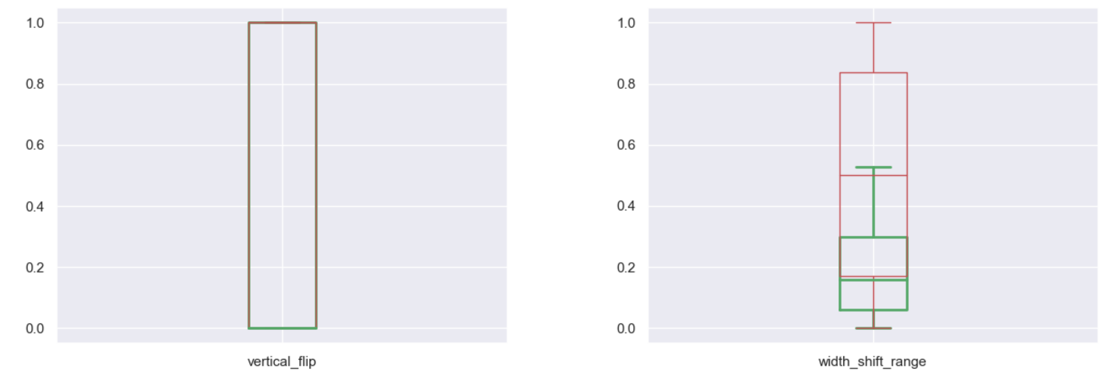


Figure 35: Data augmentation: Comparison of parameters between "good" (accuracy $\geq$ 0.4) in green and "bad" (accuracy $<$ 0.4) in red.

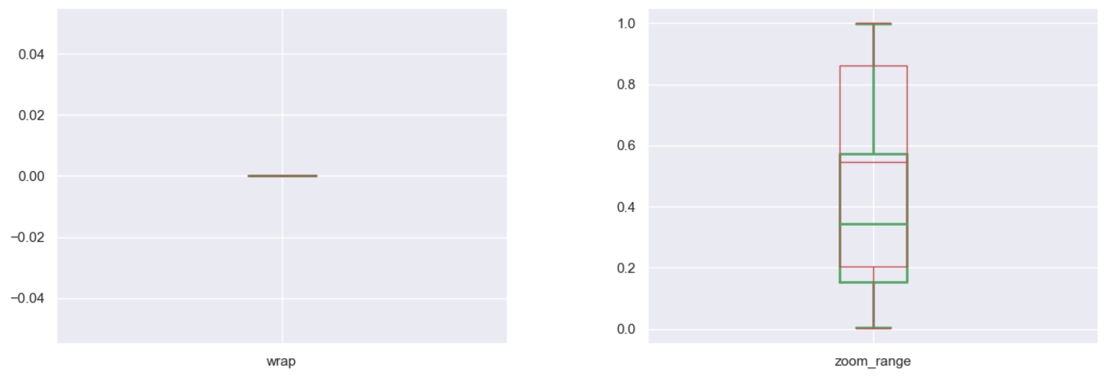


Figure 36: Data augmentation: Comparison of parameters between "good" (accuracy $\geq$ 0.4) in green and "bad" (accuracy $<$ 0.4) in red.

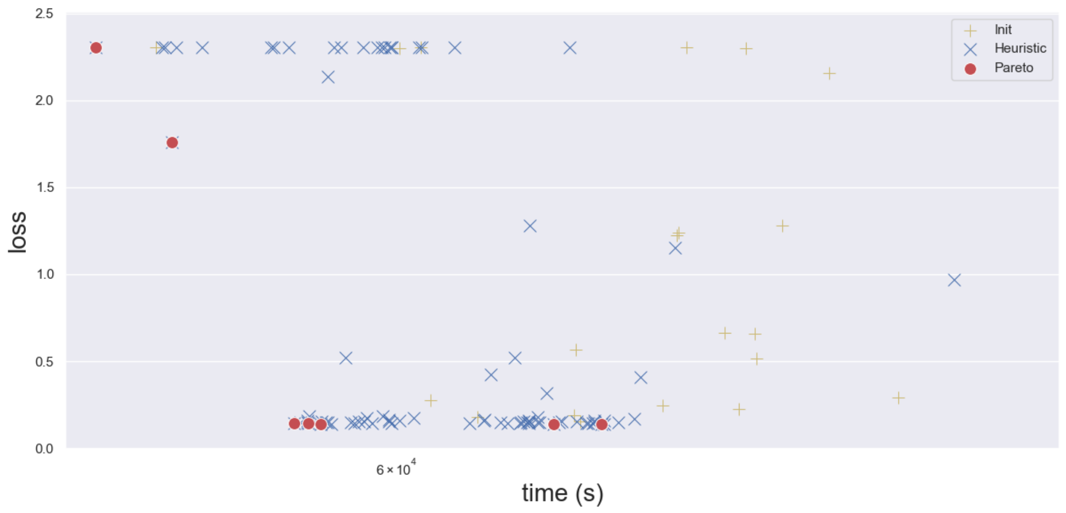


Figure 37: The "Training schedule optimization" experiment on CIFAR10. 104 samples taken, ten trees in the random forest

11 Training schedule optimization experiment

The network with the highest validation accuracy of the "Data augmentation" experiment's results had its training schedule optimized. Table 17 explains the parameters that are optimized. Note that the SMS-MIP-EGO optimizer searches for the optimal training optimizer, thus an optimizer optimizes over a set of optimizers. Table 18 shows the ranges upon which SMS-MIP-EGO optimizes these parameters. Its parameters are listed in table 16. The upper bound of the learning rate search space was chosen to be two times the learning rate of this best network as found by the "Data augmentation" experiment, in order to give SMS-MIP-EGO some room to work with, without straying to far from a possible optimum. The basic structure of the network is fixed at this point, assuming that the "Data augmentation" experiment found a near optimal network. The network was trained for 100 epochs for each training schedule and 104 training schedules were evaluated. Table 27 in the appendices shows the file name information of the data file and the file name of the construction script used in this experiment. The data file can be found online [13].

11.1 Data analysis

To analyze the training schedules, the results were split into "good" and "bad" configurations, splitting on a validation accuracy of 0.86. Networks with a validation accuracy greater or equal to 0.86 were considered to be "good", those with a validation smaller than 0.86 were considered to be "bad". This resulted in 28 "good" training schedules and 76 "bad" training schedules. Almost all training schedules used the Stochastic Gradient Descent (SGD) training optimizer. This can be explained by the fact that during the "Data augmentation" experiment, the SGD optimizer was used. Therefore, the networks were optimized to work good with this optimizer. SMS-MIP-EGO has the ability to sense this optimizer is a good choice, thereby almost always selecting this optimizer.

Paramter	Value
stack_0	3
stack_1	1
stack_2	2
stack_3	1
stack_4	5
stack_5	0
stack_6	0
s_0	2
s_1	1
s_2	3
s_3	3
s_4	3
s_5	3
s_6	3
filters_0	246
filters_1	120
filters_2	397
filters_3	473
filters_4	191
filters_5	109
filters_6	535
filters_7	202
filters_8	353
filters_9	339
filters_10	305
filters_11	507
filters_12	474
filters_13	350
k_0	2
k_1	11
k_2	4
k_3	4
k_4	14
k_5	10
k_6	8
k_7	9
k_8	9
k_9	5
k_10	10
k_11	9
k_12	4
k_13	4
activation	“selu”
activ_dense	“softmax”
dropout_0	0.005004155479145995
dropout_1	0.16597601970646955
dropout_2	0.3496803660826153
dropout_3	0.11567654065541401
dropout_4	0.025329970168830974
dropout_5	0.09773198911653828
dropout_6	0.19656398582242512
dropout_7	0.2567508735733037
dropout_8	0.10220859898802954
dropout_9	0.282536761776477
l2	0.00043366714416766863
global_pooling	True
skstart_0	6
skstart_1	0
skstart_2	6
skstart_3	3
skstart_4	1
skstep_0	2
skstep_1	8
skstep_2	2
skstep_3	2
skstep_4	1
max_pooling	True
dense_size_0	1344
dense_size_1	1216
batch_size_sp	75
featurewise_center	False
samplewise_center	False
featurewise_std_normalization	False
samplewise_std_normalization	False
zca_epsilon	1.2393513955305375e-06
zca_whitening	False
rotation_range	31
width_shift_range	0.11326574574565945
height_shift_range	0.5512549395731117
shear_range	4.413108635288765
zoom_range	0.02446592218470434
channel_shift_range	0.002134671459292783
fill_mode	“nearest”
cval	0.24779638415638786
horizontal_flip	True
vertical_flip	False

Table 16: Parameters of the network used optimize the training schedule on.

Parameter	Explanation
lr	The learning rate
drop	Constant the learning rate is multiplied by after a number of epochs specified by “epochs_drop”
epochs_drop	The number of epochs it takes for the learning rate to drop
momentum	This parameter is used by the “SGD” optimizer.
optimizer	The optimizer that optimizes the training process
rho	This parameter is used by the “RMSprop” and “Adadelta” optimizer.

Table 17: Extra search space parameters for the “Training schedule optimization” experiment explanation.

Parameter	Type	Bounds	# Dimensions
lr	Continuous	[1e-4, 2 * 0.003521543292982737]	1
drop	Continuous	[0.1, 0.95]	1
epochs_drop	Discrete	[1, 40]	1
momentum	Continuous	[0.8,0.99]	1
optimizer	Nominal	[“SGD”,“RMSprop”,“Adagrad”,“Adadelta”,“Adam”,“Adamax”,“Nadam”]	1
rho	Continuous	[0.8,0.99]	1

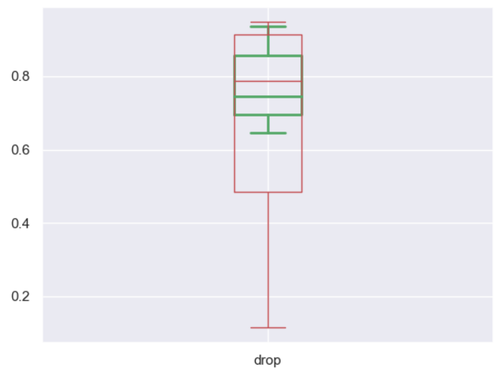
Table 18: search space of the “Training schedule optimization” experiment

Time (s)	Loss	Acc
56003.90	2.30	0.10
56993.21	1.76	0.17
58603.34	0.15	0.86
58802.00	0.14	0.87
58969.00	0.14	0.87
62196.51	0.14	0.87
62881.96	0.14	0.87

Table 19: Paretofront of the “Training schedule optimization” experiment each entry is trained for 100 epochs

Parameter	Value
lr	0.0016
drop	0.82
optimizer	“SGD”
epochs_drop	13
rho	0.92
momentum	0.96

Table 20: The best configuration of the “Training schedule optimization” experiment



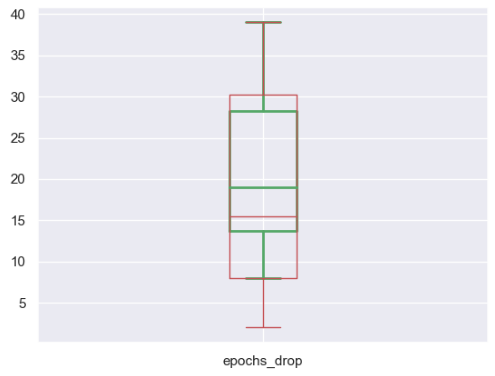


Figure 38: Training schedule optimization: Comparison of parameters between "good" (accuracy ≥ 0.86) in green and "bad" (accuracy < 0.86) in red.

11.1.1 High vs. low accuracy

Figure 38, 39 and 39 show box plots of these sets of "good" and "bad" training schedules. The following trends were found:

- The "good" training schedules tend to have a higher *drop* value than the "bad" training schedules. This indicates that the learning rate should be gradually reduced. The average lies around 0.75 for the "good" schedules
- The "good" training schedules tend to have a higher *epochs_drop* value than the "bad" training schedules. The average is around 19 epochs for the "good" schedules.
- The averaged *lr* for the "good" training schedules lies around 0.0016, which is lower than the 0.0035 that was found by the "Data augmentation" experiment.
- The average *momentum* value lies around 0.93 for the "good" training schedules. The "good" and "bad" training schedules do not differ a lot on this point, with the exception that more "good" schedules have a lower momentum value.
- Since almost all training schedules use the SGD optimizer, the *rho* value does not matter, since SGD does not use this value.

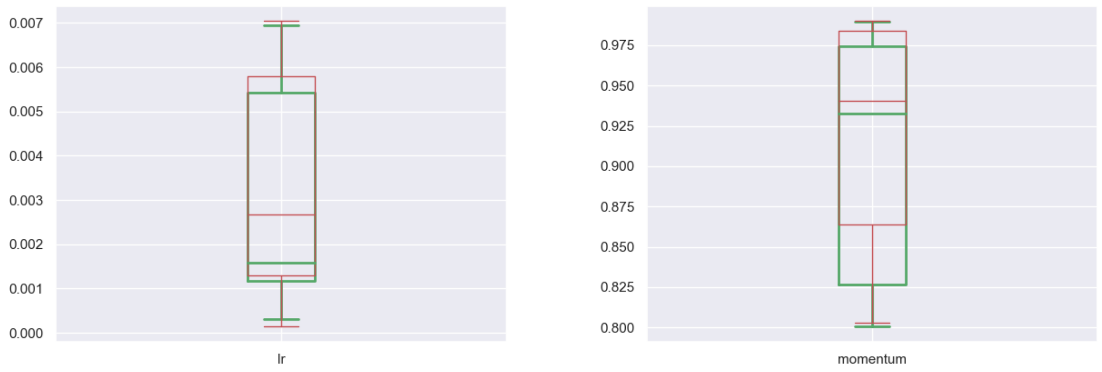


Figure 39: Training schedule optimization: Comparison of parameters between "good" (accuracy ≥ 0.86) in green and "bad" (accuracy < 0.86) in red.

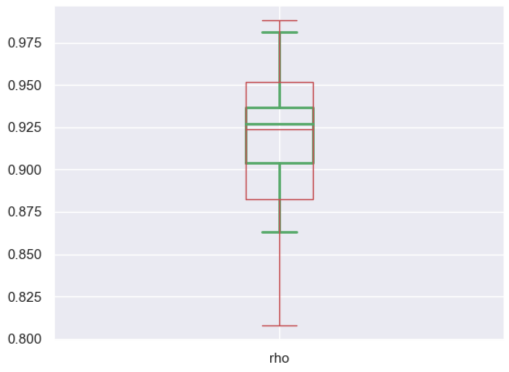


Figure 40: Training schedule optimization: Comparison of parameters between "good" (accuracy ≥ 0.86) in green and "bad" (accuracy < 0.86) in red.

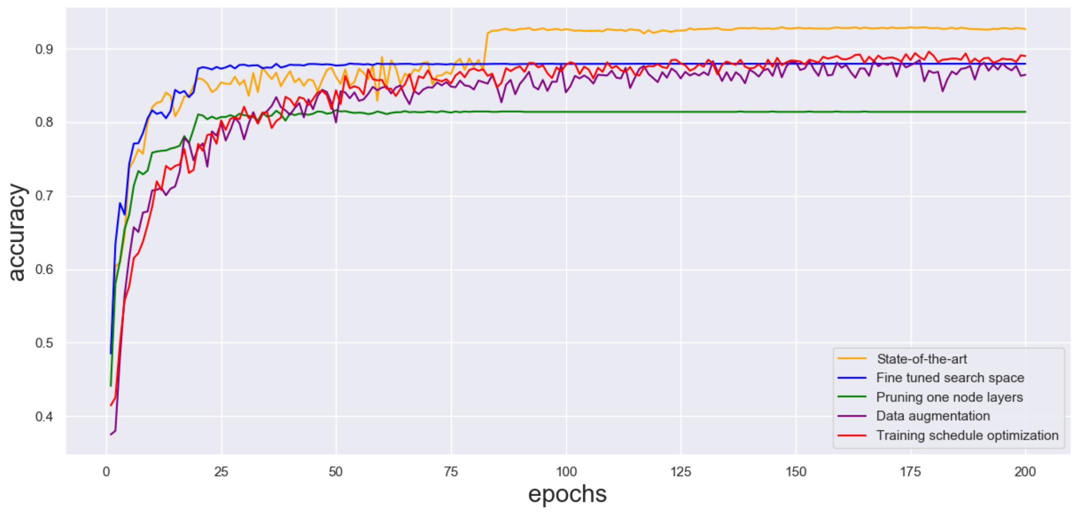


Figure 41: Overview of the evaluation accuracy of the best networks trained for 200 epochs.

12 Extra training of best networks

Figure 41 and its zoomed in version, figure 42, show the validation accuracy of the configurations that had the highest validation accuracy during the SMS-MIP-EGO step, when trained for 200 epochs. Note that the highest validation accuracy "Fine-tuned search space" experiment result used an evaluation set that was also used during the optimization of SMS-MIP-EGO, thereby having the potential to leak information and overfit on the validation set. The configurations with the highest validation accuracy of the "Pruning one node layers", "Data augmentation" and "Training schedule optimization" experiment however, used a validation set independent from the validation set that was used in the optimization step. The original train set was split into a new train set and a validation set, containing 2000 samples, with roughly the same amount of samples per layer. During the search performed by SMS-MIP-EGO, the configurations were trained on the new train set and evaluated on the original test set. In the last training session, networks were again trained on the new train set but evaluated on the validation set.
Table 21 shows the maximum validation accuracy reached during the training of the networks with the highest validation accuracy from the different methods. The state-of-the-art RESnet-30 [23] has the highest validation accuracy with 0.93, closely followed by the network with the highest validation accuracy from the "Data augmentation" experiment combined with the best training schedule from the "Training schedule optimization" experiment, which had resulted in an accuracy of 0.90.

13 Summary

A summary of all experiments can be seen in table 22. The "Fine-tuned search space" and "Pruning one node layers" experiment have the highest hypervolume value. The "Data augmentation" experiment showed the smallest hypervolume, the highest percent dysfunctional networks and the smallest amount of Pareto optimal solutions. Nonetheless it was deemed to be the method with the highest potential, since it uses data augmentation.

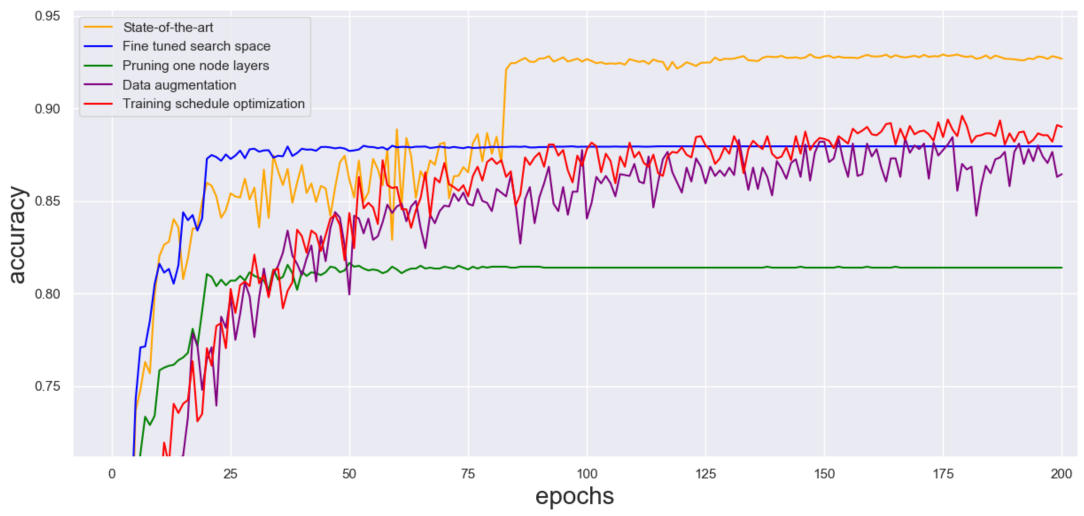


Figure 42: Close up of the overview of the evaluation accuracy of the best networks trained for 200 epochs.

Method	Max validation accuracy
State-of-the-art	**0.93**
Fine-tuned search space	0.88
Pruning one node layers	0.82
Data-aug	0.88
Training schedule optimization	0.90

Table 21: A comparison of the maximum validation accuracy of the best networks when trained for 200 epochs

Experiment	Iterations	Hypervolume	% dysfunctional	# paretofront	highest acc.
Base	410	5.37 e5	4.15	6	0.74
Fine-tuned search space	810	5.55 e5	7.41	11	0.81
Pruning one node layers	804	5.55 e5	6.09	15	0.80
Data augmentation	807	5.27 e5	13.14	5	0.71

Table 22: Results of different experiments using the *Skippy* method. During the evaluation step, networks were trained for ten epochs. The reference point for hypervolume calculation is (200000, 3). The “Base” experiment was run for only 410 iterations, since the search space needed tweaking, before spending too much resources. The rest of the experiments were run for at least 800 iterations. The slight difference in the number of iterations in these “800 plus” experiments was caused by a different amount of parallel execution for each experiment, resulting in stop criteria being triggered at a different amount of iterations. This slight difference was considered to be irrelevant.

14 Conclusion

After the experiments, it can be concluded that SMS-MIP-EGO, when combined with *Skippy*, can successfully find CNNs in a given search space, minimizing training time and maximizing validation accuracy. The found CNNs did not beat the current state-of-the-art CNNs, as can be seen in the comparison in section 12.
Results that are able to compete with the current state-of-the-art CNNs were found in the "Fine-tuned search space" experiment in the first ten epochs as is described in section 7. Adding data augmentation and further training schedule optimization resulted in a validation accuracy close to the state of the art (0.90) when trained for 200 epochs as is described in section 12.
SMS-MIP-EGO could be applied as a general black box optimization algorithm, because it can optimize CNN architectures as well as training schedules as shown in section 10 and 11.
The validation accuracy of deep networks is improved by adding skip connections, as can be seen in section 5. The tested network's performance deteriorated to random guessing when omitting skip connections, whereas the version with skip connections reached an evaluation accuracy of 0.72 in 20 epochs.
Section 8 shows that pruning the last non-convolutional part of a network did not matter significantly. It did not impact the overall performance of networks and also did not significantly impact the memory requirements. In section 10, the amount of stride in the network was constricted to not make this reduction to a 1×1 feature size happen.
According to section 7 and 8, the choice of activation function matters the most for the accuracy of a network.
Data analysis in subsections 7.1, 8.3, 10.1 and 11.1 led to the discovery of the following best practices. However, these best practices do not necessarily generalize to other architectures or data sets other than CIFAR10. The practices found were:

- It is bad practice to put a dropout layer directly after the input. This does not benefit training a network, it merely muddles the input.

- More training time generally means higher accuracy, even when the amount of epochs is fixed. The more complex the network, the more training time is needed. Thus, the complexer the network, the more potential it has for reaching a high accuracy.

- A learning rate around 0.007 is a good starting point to train a CNN with skip connections for CIFAR10.

- Another rule of thumb is to use dropout values around 0.23 as a starting point for tweaking when not using data augmentation and around 0.16 when using data augmentation.

- L2 regularization should only be used with very small values under 0.002, or not at all.

- The results indicate that the elu activation function works well with *Skippy*.

- The batch size has little impact on the accuracy of a network.

- It is good practice to reduce the feature size to 1×1 before connecting the dense layer, or else to use global pooling.

- The “good” networks in the results tend to use the option to do Max pooling. These results challenge the findings of Springenberg et al. [18], that question the necessity of Max pooling layers.

- In data augmentation, channel shifting should only be done with small values between zero and 0.05, or be omitted.

- The results show that, when using a shear operation as a data augmentation technique, use the nearest pixel as an infill value, or the reflect method for infill, but avoid the constant value and do not use the wrap method.

- The results show that when shifting the height of images during data augmentation, it works best to use a height shift range between zero and 0.35.

- When using image rotation as a data augmentation technique, the highest validation accuracy is yielded when images are rotated for smaller amounts, preferably between zero and 100 degrees.

- Flipping images among the vertical axis during data augmentation is not needed for good results.

- For a high validation accuracy, it is best not to shift the width too much when using data augmentation, ideally only between values of zero and 0.18

- Images should not be zoomed too drastically during data augmentation, because this can negatively impact the validation accuracy. Ideally stay within a range between zero and 0.35.

In the configuration datasets, correlations were analyzed. No interesting or meaningful correlations were present, aside from obvious ones. These results are therefore omitted.
An a priori rule finding algorithm was used to find rules in discretized versions of the configuration datasets. No clear rules could be found by this algorithm. Its results are therefore omitted.

15 Future Work

It would be interesting to use the *Skippy3* method, as described in section 5, and vary the amount of feature reduction by putting it as a variable in the search space. This method showed potential, but was not used due to the amount of GPU memory it needed. It can be interesting to see how this method performs with an increased amount of memory, for example to run it on multiple GPUs, or to wait for GPUs with more memory to be developed.
Skippy uses a fixed number of stacks. It might be an option to make the number of stacks variable. Parameters like layer width and kernel size can be varied by picking a value for the first stack and then increase or decrease it by a constant factor, or use a function with its own parameters to increase or decrease the values.
Networks could be trained for 100 epochs in the evaluation step to get better estimates of the networks' performance. Training schedule optimization can be added to the search space immediately. Evaluating 800 networks that were trained for ten epochs took two weeks when using ten Nvidia Tesla K80 GPUs simultaneously. Scaling up the experiment by a factor of ten would take roughly 20 weeks for it to complete. Halving the amount of GPUs reduced to five so as not to take up too many resources would double this to 40 weeks.
It could be interesting how SMS-MIP-EGO with *Skippy* performs on less commonly used data sets. The state of the art results are so highly optimized on CIFAR10, that they are hard to beat.
Since SMS-MIP-EGO is not the best MI optimizer available, it would be interesting to use different state of the art optimizers, like Yang's method [11] for example, with *Skippy* to see if it can then beat state of the art networks.

References

[1] Keras documentation: Image preprocessing. `https://keras.io/preprocessing/image/`. Accessed: 2019-07-20.

[2] pandas.dataframe.boxplot. `https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.boxplot.html`. Accessed: 2019-07-20.

[3] Quoc V. Le Barret Zoph. Neural architecture search with reinforcement learning. *ICLR 2017 conference submission*, 2017.

[4] Ashwin Bhandare, Maithili Bhide, Pranav Gokhale, and Rohan Chandavarkar. Applications of convolutional neural networks. *International Journal of Computer Science and Information Technologies*, pages 2206–2215, 2016.

[5] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[6] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[7] Mohammad Sadegh Ebrahimi and Hossein Karkeh Abadi. Study of residual networks for image recognition. *arXiv preprint arXiv:1805.00325*, 2018.

[8] Thomas Elsken, Jan-Hendrik Metzen, and Frank Hutter. Simple and efficient architecture search for convolutional neural networks. *arXiv preprint arXiv:1711.04528*, 2017.

[9] Adam Gaier, Alexander Asteroth, and Jean-Baptiste Mouret. Data-efficient neuroevolution with kernel-based surrogate models. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 85–92. ACM, 2018.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[11] Thomas Bäck Kaifeng Yang, Koen van der Blom and Michael Emmerich. Towards single- and multiobjective bayesian global optimization for mixed integer problems. 2018.

[12] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 dataset. `https://www.cs.toronto.edu/~kriz/cifar.html`. Accessed: 2019-07-20.

[13] Christiaan Lamers. Data and the code of method. `https://github.com/christiaanlamers/sms-mip-ego`. Accessed: 2019-08-03.

[14] Rui Li, Michael TM Emmerich, Jeroen Eggermont, Thomas Bäck, Martin Schütz, Jouke Dijkstra, and Johan HC Reiber. Mixed integer evolution strategies for parameter optimization. *Evolutionary computation*, 21(1):29–64, 2013.

[15] Tejaswini Pedapati Martin Wistuba, Ambrish Rawat. A survey on neural architecture search. *arXiv:1905.01392*, 2019.

[16] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017.

[17] Wolfgang Ponweiser, Tobias Wagner, Dirk Biermann, and Markus Vincze. Multiobjective optimization on a limited budget of evaluations using model-assisted s-metric selection. In *International Conference on Parallel Problem Solving from Nature*, pages 784–794. Springer, 2008.

[18] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[19] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[20] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. *arXiv preprint arXiv:1807.11626*, 2018.

[21] Yong Rui Tao Wei, Changhu Wang and Chang Wen Chen. Network morphism. *arXiv:1603.01670*, 2016.

[22] Bas van Stein, Hao Wang, and Thomas Bäck. Automatic configuration of deep neural networks with ego. *arXiv preprint arXiv:1810.05526*, 2018.

[23] Yuxin Wu. cifar10-resnet.py. `https://github.com/tensorpack/tensorpack/blob/master/examples/ResNet/cifar10-resnet.py`. Accessed: 2019-07-20.

Appendices

Parameter	Value	Explanation
save_name	"data_skippy_cifar10_big_one"	Head of save file names
objective	"./all_cnn_bi_skippy.py"	Name of construction method program

Table 23: File-name information of the "Base" experiment. The file of the objective function can be found online[13]. The data file can also be found online [13] in the "data_thesis" folder.

Parameter	Value	Explanation
save_name	"data_skippy_cifar10_big_one_tweaked_restarted"	Head of save file names
objective	"./all_cnn_bi_skippy.py"	Name of construction method program

Table 24: File-name information of the "Fine-tuned search space" experiment. The file of the objective function can be found online[13]. The data file can also be found online [13] in the "data_thesis" folder.

Parameter	Value	Explanation
save_name	"data_skippy_cifar10_big_one_cut_smaller_restart_2"	Head of save file names
objective	"./all_cnn_bi_skippy_cut.py"	Name of construction method program

Table 25: File-name information of the "Pruning one node layers" experiment. The file of the objective function can be found online[13]. The data file can also be found online [13] in the "data_thesis" folder.

Parameter	Value	Explanation
save_name	"data_skippy_cifar10_better_data_augmentation_big_one_restarted1"	Head of save file names
objective	"./all_cnn_bi_skippy_aug.py"	Name of construction method program

Table 26: File-name information of the "Data augmentation" experiment. The file of the objective function can be found online[13]. The data file can also be found online [13] in the "data_thesis" folder.

Parameter	Value	Explanation
save_name	"data_skippy_cifar10_better_data_augmentation_train_tweak_big_one_restarted1"	Head of save file names
objective	"./all_cnn_bi_skippy_aug_tr_tw.py"	Name of construction method program

Table 27: File-name information of the "Training schedule optimization" experiment. The file of the objective function can be found online[13]. The data file can also be found online [13] in the "data_thesis" folder.