CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Data Specific Neural Architecture Search with Minimal Training

Analysing the Dependency of Local Linear Operators

Master's thesis in Computer science and engineering

LEON SIEGERT

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Master's thesis 2022

Data Specific Neural Architecture Search with Minimal Training

Analysing the Dependency of Local Linear Operators

LEON SIEGERT

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

Data Specific NAS with Minimal Training
Analysing the Dependency of Local Linear Operators
LEON SIEGERT

Supervisor: Yinan Yu, Computer Science and Engineering
Advisor: Ralph Grothmann, Siemens AG
Examiner: Mary Sheeran, Computer Science and Engineering

Master's Thesis 2022
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2022

Data Specific NAS with Minimal Training
Analysing the Dependency of Local Linear Operators
LEON SIEGERT
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Conventional neural architecture search is computationally expensive. Training-free alternatives are not yet capable of designing architectures which are on par with hand-crafted architectures. This work conducts research in the domain of training-free neural architecture search towards closing the performance gap to hand-crafted architectures. The best training-free neural architecture search methods to date compute a proxy score for the validation accuracy of architectures which can be computed before the architecture is trained based on the local linear operators of the architectures. This work proposes, to improve over existing methods by basing the score on the mutual dependency of the local linear operators instead of directly on the local linear operators. The work proposes further, to quantify the mutual dependency of the local linear operators in terms of the parameters of the network. A procedure is implemented to obtain the dependency of the local linear operators based on the network parameters. The results of the experiments in this work indicate that the mutual dependency of the local linear operators is conclusive in regard to the network's validation accuracy already before the architecture is trained and therefore a promising base for a training-free neural architecture search.

Keywords: Neural Architecture Search Local Linear Operators, Training-Free Neural Architecture Search, Neural Architecture Search Without Training.

Acknowledgements

I would like to thank Yinan Yu for the exceptional supervision and immense contribution to this thesis. Thank you, Yinan!

I would further like to thank Joel Svensson and Ralph Grothmann for their valuable inputs during the status meetings and beyond. Thank you!

Leon Siegert, Gothenburg, June 2022

Contents

List of Figures

List of Tables

1

Introduction

Machine learning (ML), to a certain extent, frees the developer from having to understand the exact input-output relationship of a system when modelling it. Instead, ML algorithms find a parameterization of the model in an automated fashion. There are, however, important design choices, such as the model family or hyperparameter choice, left to make for the developer. These choices are typically made by a trade-off between expert knowledge and a computationally expensive search over the hyperparameter and model space. Small and medium sized ML projects usually neither have access to the expert knowledge nor to the computational capacity to find reasonable solutions to these problems. *Automated Machine Learning* (AutoML) is a discipline that aims to automate design choices involved when developing and training ML models in an efficient way. The work presented here focuses on automating the model choice in particular for the model class of *convolutional neural networks* (CNN). This task falls into the domain of *Neural Architecture Search* (NAS) which is one of the sub-domains of AutoML.

Research into NAS has been conducted since 2002 based on *evolutionary algorithms* (EA) [6]. The first relevant successes in NAS for CNNs have been achieved in 2017, by *reinforcement learning* (RL) based algorithms [1][7]. The underlying principle shared by these algorithms is the iteration over the following steps: (1) an algorithm specific controller samples an architecture. (2) The architecture is trained. (3) The resulting validation accuracy of the architecture is fed back into the controller as a reward signal and (4) the controller is updated based on the reward signal. The controller learns which architectures are likely to perform well and samples such architectures with a higher probability. Figure 1.1 visualizes this process. The set of candidate architectures from which the controller samples an architecture is the search space of the algorithm.

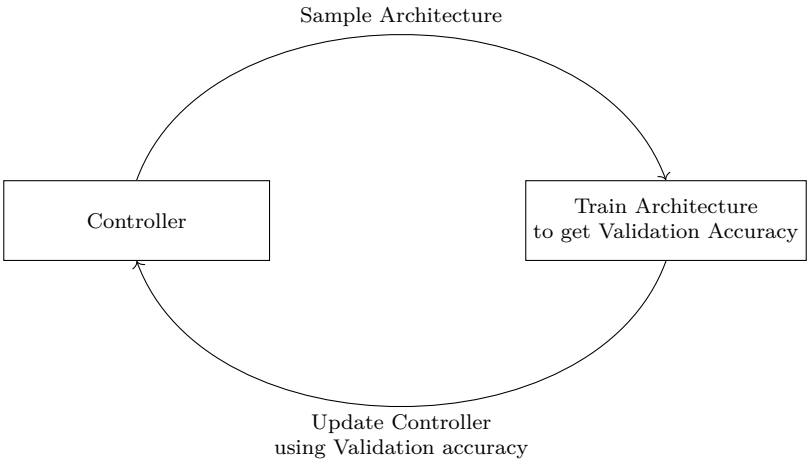

Figure 1.1: Principal of conventional NAS. Modified from [1].

Conventional NAS has one important drawback: The search spaces for NAS problems are usually enormous, even when only including a subset of the design choices involved when creating a neural network architecture. Common NAS search spaces comprise between 10^4 and 10^{18} different architectures [2][8][9][3]. Training a controller on these search spaces requires a large number of iterations. Each iteration requires a complete training run of a CNN. This makes conventional NAS resource consuming and thus expensive. Beyond the computational cost, the energy required to power the computational resources for a conventional NAS run causes potentially a lot of greenhouse gas emissions. Only a single run of conventional NAS can produce 500 kg of carbon dioxide emissions[1]. Consequently, research has been conducted in how to reduce the resource demand of NAS by training the candidate architectures only partially. This sub-domain of NAS is referred to as *NAS with partial training*. The approaches in NAS with partial training succeed in reducing the resource demand, however, in order to reach the performance of handcrafted architectures, one still needs to train the architectures for a long time [12].

The high resource demand of conventional NAS and NAS without training motivate research towards NAS without training. The first attempts in this field have been made in 2020 [13][14]. A common approach in NAS without training is to compute a scalar score for the architectures before the training that is correlated with the architecture's validation accuracy after training. The purpose of this score is to predict the accuracy of a trained network with the same architecture as the untrained, scored network. A well designed score can be used to identify high performing architectures. One way to perform NAS with such a score is to replace the actual validation accuracy by the score as a reward signal for the controller in the conventional NAS cycle (ref. figure 1.1).

The objective of research in the NAS-without-training domain is to optimize the cor-

[1] Assuming NVIDIA Titan RTX GPUs with a power consumption of 280 W [10], 12800 trained networks [1], 30 minutes training time per network, and 0.301 kg of CO_2 emissions per kWh [11]

relation of the score with the validation accuracy of the architectures. The best NAS scores to date achieve correlations of around 0.6 on common NAS benchmarks[15]. This correlation leaves vast room for improvement.

The aim of this thesis is to conduct research towards a NAS score with a higher correlation with the final validation accuracy of the architectures than existing scores. The best existing approaches are based on analysing the *Local Linear Operators*[2] (LLOs) of the candidate networks [14][15][16]. This thesis conducts research into improving the best existing NAS scores in two aspects:

1. The best existing NAS scores are based on the LLOs. The LLOs are subject to changes during the training. This thesis hypothesizes that a score computed solely based on the LLOs before training can only have a limited correlation with the performance of an architecture after training. This thesis proposes to evaluate the network's capability to adjust the LLOs during the training instead of only evaluating the LLOs before the training. This thesis proposes to analyse this capability of the network by the pairwise dependency of the LLOs.
2. The best existing NAS scores either do not take the annotations of the data into account at all [14][15] or do so only to a limited extent [16]. This is hypothesized to be sub-optimal: the annotations of the data define the task and different architectures are optimal for different tasks. Following this logic, a NAS score can only find the optimal architecture for a given task when taking the annotations of the task into account. Therefore, this thesis incorporates annotations into the proposed NAS score.

The contributions of this thesis to the field of NAS without training are the following:

1. The work proposes a novel method to quantify the pairwise dependency of LLOs of neural networks with ReLU activation.
2. The work describes an algorithm that implements the above mentioned method to compute the dependency of LLOs.
3. The work shows insights, relevant to NAS, that are gained by analysing the dependency of LLO.
4. The work proposes to base a NAS score on the dependency of LLOs.
5. The work proposes to incoorporate the labels into the computation of the NAS score to a larger extent then previous work.
6. The work proposes a novel NAS score, based on the pairwise dependency of local linear operators, that takes the annotation of the data into account.

The proposed NAS score is based on the work of [14], [15] and [16]. A detailed discussion of this work as well as how the proposed NAS score aims to improve over [14], [15] and [16] is provided in chapter 2.3.

[2]A Local Linear Operator is a linear mapping that represents the computation of a neural network with ReLU activation, locally, for a specific activation pattern. For an intuitive example of local linear operators see section 2.

1.1 Scope

This thesis proposes a novel method to compute a NAS score. The method is implemented and applied to a selection of small architectures from a common NAS search space. The implementation of the proposed method, used in this work is functional. However, it is not part of this work to optimize the implementation in terms of computational and memory efficiency. This thesis will thus not be able to make a final statement whether the method can outperform existing models or not. Such a final evaluation of the method requires to score a large number of architectures from a NAS benchmark. This is considered infeasible with the current inefficient, experimental implementation of the method. Instead this work provides results of first experiments as an indication for the potential of the method to motivate further work.

Beyond the above described limitations of the validation of the method, this section states and motivates constraints of the architecture search space on which the method is developed. The following constraints are common in NAS research [14][15][16]. This thesis considers NAS over CNN architectures for *image classification*. Image classification is among the disciplines that utilize very large neural architectures that often have several million parameters [17][18][19]. Due to the high training cost, NAS without training is particularly beneficial to this discipline. The reason for focusing on classification is that the ideas can be developed in a comparatively simple theoretical framework while it is hypothesized that the gained insights scale to regression tasks like *object localization* or even combinations of classification and regression tasks such as *object detection*.

The architectures are further assumed to have identity activation in the output layer and ReLU activation in all hidden layers. ReLU activation is considered to be so dominant in image recognition tasks that developing methods specific to networks with ReLU activations does not constrain the scope of the methods to a relevant extent.

Networks with a single output unit are considered during method development. This simplifies the method development as the influence of the architecture on the output is a relevant aspect and easier to describe with a scalar output. It is assumed that the method still scales well to multi-class tasks. The method is validated on data sets with multiple classes to show that this assumption holds.

1.2 Thesis Structure

The remaining report is structured in 5 chapters: chapter *Theory* explains an important concept that the NAS score proposed in this work is based on; *Local Linear Operators*. It follows a brief overview of the history of NAS in general as well as a summary of existing approaches in NAS without training. A discussion is provided

of how the proposed NAS approach aims to improve over previous work. Further the most common benchmarks for NAS algorithms are introduced. Chapter *Methods* explains the score design proposed in this work as well as the experiments conducted to evaluate the approach. The proposed NAS score is based on the pairwise dependency of local linear operators, which is also discussed in chapter *Methods*. Computing the pairwise dependencies of local linear operators is not trivial. Chapter *Implementation* is dedicated to the algorithmic implementation of the computation of pairwise local linear operator dependencies. Results and observations from the experiments are stated in the chapter *Results*. Chapter *Conclusions* summarizes the insight gained from the conducted research and gives suggestions for further work.

2

Theory

This chapter provides the theoretical background as well as a brief summary of the NAS history. Section 2.1 introduces the concept of *Local Linear Operators* (LLOs). Section 2.2 introduces different NAS benchmarks. A summary of the historic development of NAS is provided in Section 2.3. Section 2.3 is concluded by placing the research conducted in this work in the context of previous work.

2.1 Local Linear Operators and Local Linear Regions

Local linear operators can be derived for networks with linear or piece-wise linear activation functions such as for instance the ReLU activation function:

$$ReLU(x) = \begin{cases} x & if\ x > 0 \\ 0 & Otherwise \end{cases} \tag{2.1}$$

A ReLU is called *active* when its argument is positive and *dead* otherwise. The point where its argument is zero is called a *corner*. A corner of a ReLU in a neural network maps to a graph in the input space of the network. This graph is piece-wise linear and non-differentiable[1]. We call these graphs the *separators* of the ReLUs in a neural network. The separators divide the input space of a neural network into *local linear regions* (LLRs). Within a LLR the computation of the neural network can be described by just a linear mapping [20][2]. This mapping is called the *local linear operator* (LLO) of the LLR. Since no point in an LLR is mapped to the corner of any ReLU in the network, the pattern of active and dead ReLUs is fixed for the entire LLR. This pattern is called the *activation pattern* of the network at the referring LLR. Note that it is unique for each LLR and can thus serve to identify the LLRs. The notion *dependence of the LLOs* is used to discuss how dependently or independently the LLOs of a network can be changed during the training.

To provide an intuition for LLOs and their referring LLRs one might consider a fully-connected feed-forward architecture as shown in figure 2.1.

[1] ReLUs in the first layer of the network form an exception: Here the graphs that map to the corners of the ReLUs are *linear* hyperplanes.

[2] Note that besides ReLUs, max-pooling layers can introduce non-linearity in the network. For simplicity the effect of max-pooling layers is excluded in subsequent examples.

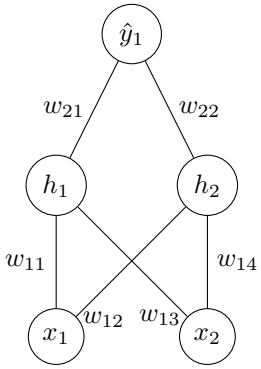

Figure 2.1: Simple fully connected feed-forward network with two inputs x_1 and x_2, two hidden units h_1 and h_2, one output $\hat{y}_1$, and its weights w.

Assuming the two hidden units have ReLU activation the computation of the network can be expressed as

$$\hat{y}_1(x_1, x_2) = w_{21} \cdot \text{ReLU}(w_{11} \cdot x_1 + w_{13} \cdot x_2) + w_{22} \cdot \text{ReLU}(w_{12} \cdot x_1 + w_{14} \cdot x_2). \quad (2.2)$$

Potential biases of the neurons are neglected to simplify the example. To transform this computation towards local linear operators one needs to explicitly express the non-linear ReLUs. This can be achieved by expressing the computation separately for every activation pattern of the network. If the activation pattern does not change, one can substitute each ReLU in equation 2.2 either by its argument if this is positive or by zero otherwise. To distinguish between different activation patterns depending on the input one can define conditions for the arguments of the ReLUs for a given input. For example illustrated in figure 2.1, ReLU h_1 is active if

$$x_1 > -\frac{w_{13}}{w_{11}} \cdot x_2 \quad (2.3)$$

and dead otherwise. Analogously, ReLU h_2 is active if

$$x_1 > -\frac{w_{14}}{w_{12}} \cdot x_2 \quad (2.4)$$

and dead otherwise. In total the network has at most four activation patterns:

$$\textbf{Case 1 :}\ h_1 : \text{active}, h_2 : \text{active} \quad \text{if} \left(x_1 > -\frac{w_{13}}{w_{11}} \cdot x_2\right) \wedge \left(x_1 > -\frac{w_{14}}{w_{12}} \cdot x_2\right) \quad (2.5)$$

$$\textbf{Case 2 :}\ h_1 : \text{active}, h_2 : \text{dead} \quad \text{if} \left(x_1 > -\frac{w_{13}}{w_{11}} \cdot x_2\right) \wedge \left(x_1 \leq -\frac{w_{14}}{w_{12}} \cdot x_2\right) \quad (2.6)$$

$$\textbf{Case 3 :}\ h_1 : \text{dead}, h_2 : \text{active} \quad \text{if} \left(x_1 \leq -\frac{w_{13}}{w_{11}} \cdot x_2\right) \wedge \left(x_1 > -\frac{w_{14}}{w_{12}} \cdot x_2\right) \quad (2.7)$$

$$\textbf{Case 4 :}\ h_1 : \text{dead}, h_2 : \text{dead} \quad \text{if} \left(x_1 \leq -\frac{w_{13}}{w_{11}} \cdot x_2\right) \wedge \left(x_1 \leq -\frac{w_{14}}{w_{12}} \cdot x_2\right) \quad (2.8)$$

Based on these conditions one can explicitly express the computation of the network:

$$\hat{y}_1(x_1, x_2) = \begin{cases} w_{21} \cdot (w_{11} \cdot x_1 + w_{13} \cdot x_2) + w_{22} \cdot (w_{12} \cdot x_1 + w_{14} \cdot x_2) & \textbf{Case 1} \\ w_{21} \cdot (w_{11} \cdot x_1 + w_{13} \cdot x_2) & \textbf{Case 2} \\ w_{22} \cdot (w_{12} \cdot x_1 + w_{14} \cdot x_2) & \textbf{Case 3} \\ 0 & \textbf{Case 4} \end{cases} \quad (2.9)$$

Which can be expressed as four linear functions of the inputs x_1 and x_2:

$$\hat{y}_1(x_1, x_2) = \begin{cases} (w_{21} \cdot w_{11} + w_{12} \cdot w_{22}) \cdot x_1 + (w_{13} \cdot w_{21} + w_{14} \cdot w_{22}) \cdot x_2 & \text{if } \textbf{Case 1} \\ w_{11} \cdot w_{21} \cdot x_1 + w_{13} \cdot w_{21} \cdot x_2 & \text{if } \textbf{Case 2} \\ w_{12} \cdot w_{22} \cdot x_1 + w_{14} \cdot w_{22} \cdot x_2 & \text{if } \textbf{Case 3} \\ 0 & \text{if } \textbf{Case 4}. \end{cases} \tag{2.10}$$

Expressing the computation of the network in this form shows that for each of the four activation patterns the output depends linearly on the input. The four linear mappings are the local linear operators of the network. Returning to the graph notation one can now represent the computation of the non-linear network in figure 2.1 as four linear models depending on the input values as shown in figure 2.2.

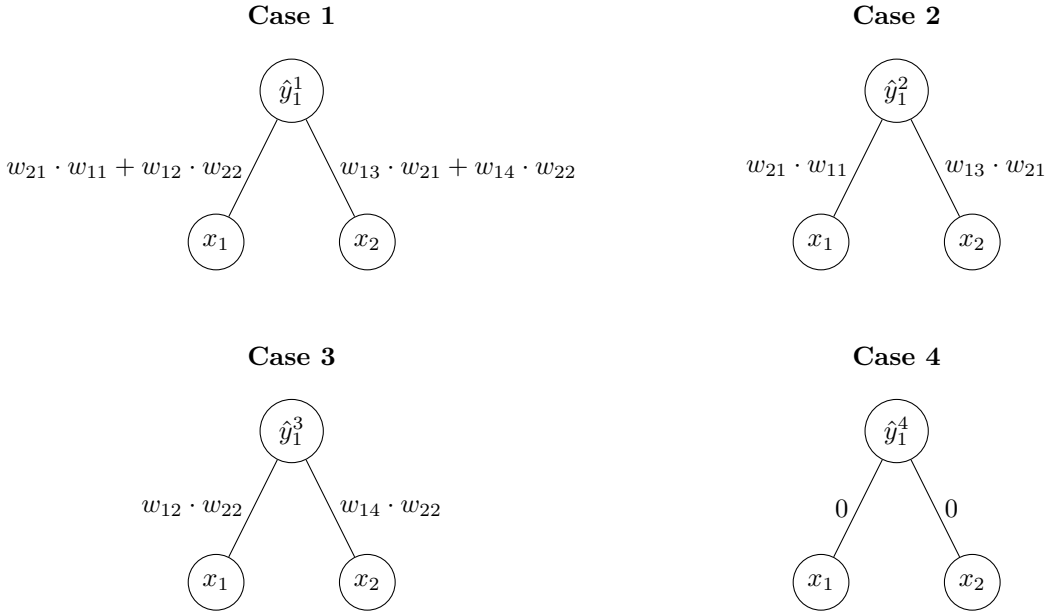

Figure 2.2: Decomposition of the network in figure 2.1 into four linear models. The superscript of the output $\hat{y}^n$ indicates the case index defined in the nth case from equation 2.5 to 2.8.

Local linear operators will subsequently be written as vectors of the coefficients of the input. Taking the region of the input space for which equation 2.5 holds as an example the local linear operator for case 1 $\lambda^{(C1)}$ can be noted as the vector

$$\lambda^{(C1)} = \left[\lambda_1^{(C1)}, \lambda_2^{(C1)}\right] = [w_{11} \cdot w_{21} + w_{12} \cdot w_{22}, w_{13} \cdot w_{21} + w_{14} \cdot w_{22}] \tag{2.11}$$

where the subscript of the λ indicates the index in the vector as well as of the referring input and the superscript the case to which the LLO is associated. During the following, however, the superscript is mostly used to associate an LLO with an instance of the training data set rather than the actual activation pattern. The LLO for the nth instance of the training data set is then denoted as $\lambda^{(n)}$.

One can obtain the same output as the network by the inner product of the input x and the LLO $\lambda^{(C1)}$, given that x lies in the LLR that fulfills equation 2.5.

Note that linear operators depend on the activation pattern. The region of the input space in which the architecture is activated in this particular pattern is the local linear region of the local linear operator. Figure 2.3 shows an example of how the input space of the fully connected network in figure 2.1 could be divided in the four local linear regions corresponding to the cases from equations 2.5 to 2.8.

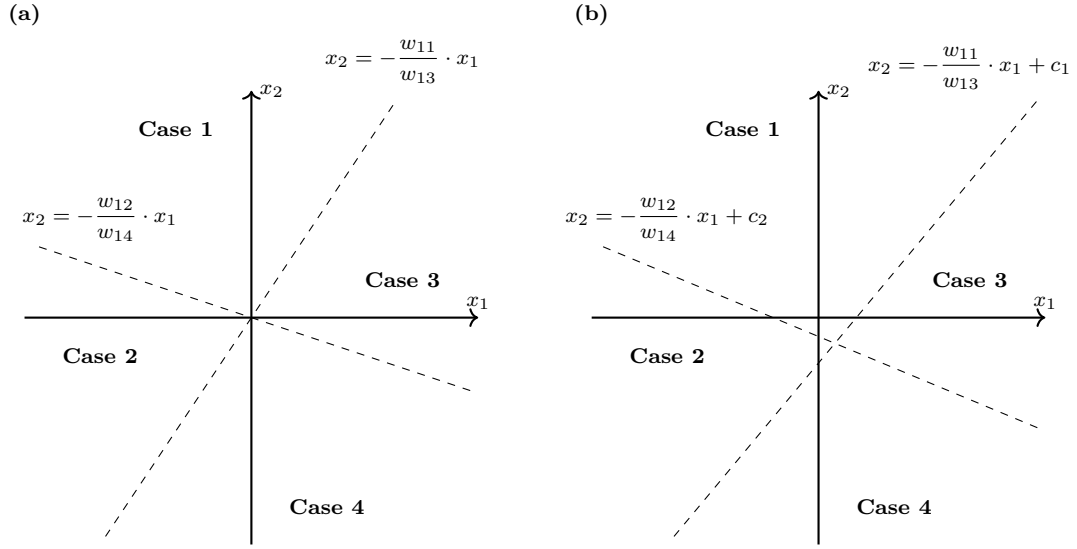

Figure 2.3: Input space of the network in figure 2.1. The dashed lines are exemplary instances of the separators which map to the corners of the ReLUs h_1 and h_2. That means all points on this line lead to one of the ReLUs having an argument equal to zero. The line equations are derived from the conditions in equation 2.5 to 2.8. (a) shows a possible setting of separators without biases in the network, as shown in figure 2.1. (b) shows the effect of potential biases by adding the constants c_1 and c_2 to the separator line equations. With biases the separators are not constrained to going through the origin of the input space. The four regions in which the separators divide the input space are the four local linear regions of the network corresponding to the four cases from equations 2.5 to 2.8 as indicated by the labels in bold between the separators. The slopes of the separators depend on the parameters of the network and thus change during the training.

The lines that separate the LLRs are the corners of the ReLUs h_1 and h_2 mapped to the input space.

Note that one convenient way to obtain these LLOs is to compute the local gradient of the output with respect to the input. For the network depicted in figure 2.1 the LLO of the nth case can thus be obtained as

$$\lambda^{(Cn)} = \nabla_x \hat{y}_1|_{Cn} = \left[\frac{\partial \hat{y}_1}{\partial x_1}, \frac{\partial \hat{y}_1}{\partial x_2}\right]_{Cn}, \tag{2.12}$$

where the subscript indicates that the input x fulfills case n. The fact that LLOs can be obtained by differentiating is particularly useful in the implementation of LLO based algorithms as most deep learning libraries have implemented functionalities

to compute gradients [21][22].

Considering a particular data set, it is hypothesised that the LLOs and referring LLRs of a network are approximately data point specific. That is, it is highly probable that at most one data point falls into each LLR. This hypothesis is founded based on two premises: (1) It is assumed that the data is normalized and the network is initialized appropriately. Then the corners of the ReLUs will approximately be mapped to the same region of the input space, in which the elements of the data set are placed and (2) the number of LLRs is significantly higher then the number of data set elements. The latter can be supported by estimating the number of LLRs n_{LLR} in an architecture as an exponential function of the number of neurons n_n. Assuming the number of LLOs approximately doubles with each neuron, one can approximate the number of LLOs as

$$n_{LLR}(n_n) \approx 2^{n_n} \tag{2.13}$$

The most common image recognition data sets have between a couple of thousands and a few million instances [23][4][5][24]. If the approximation 2.13 holds, a network with only $n_n = 100$ neurons would have already $n_{LLR}(n_n) \approx 1.2 \cdot 10^{30}$ LLRs and thus multiple orders of magnitudes higher then the number of instances in any feasible image recognition data set. Given both above stated premises, it would thus be very unlikely for two data points to fall into the same LLR. This implies that the LLO are in most cases unique for a data point.

2.2 Benchmarks in Neural Architecture Search

So called NAS benches are data bases with a large number of neural architectures and their corresponding final validation accuracy. Often NAS benches also provide further meta data concerning the architecture and the training beyond the validation accuracy. Typically, NAS benches are tabular data bases similar to the fictive database shown in table 2.1

Table 2.1: Example of a fictive NAS bench with the first column holding architecture identifiers and the second column holding meta data concerning the architecture and the training. The architecture identifier in this example is a list of layers with their referring hyperparameters. E.g. *conv(16,(3,3),...)* stands for a convolution layer with 16 3×3 filters. The meta data in this example is the accuracy on the benchmark image classification data set Cifar10 [4] in percent.

Architecture Identifier	**Performance Metrics**
[conv(16,(3,3),...), ..., softmax]	{Cifar10 acc: 71%, ...}
[conv(32,(3,3),...), ..., softmax]	{Cifar10 acc: 76%, ...}
[conv(16,(5,5),...), ..., softmax]	{Cifar10 acc: 72%, ...}
[conv(32,(5,5),...), ..., softmax]	{Cifar10 acc: 75%, ...}
⋮	⋮

The benches spare the researchers the expensive training of often multiple thousands of architectures needed to develop and evaluate novel NAS approaches. Instead of

training the architectures the researcher just has to query the data base for the desired validation accuracy. Another benefit of NAS benches is that they allow the comparison of different NAS algorithms on the same architecture search space. In this way, they serve as benchmarks in the NAS domain.

A drawback of tabular NAS benches is that every architecture in the database needs to be trained and evaluated. This makes the creation process of tabular NAS benches expensive and limits the number of architectures in the database. An approach to overcome this problem is a *surrogate NAS bench*. In a surrogate NAS benchmark, the results for most architecture instances are obtained by interpolating between a few actually trained instances. So far surrogate benches use smaller data sets and are less commonly used as a benchmark for NAS algorithms. Therefore this work uses the tabular benches described below. A detailed description of the surrogate bench *NAS-Bench-301* [9], however, can be found in appendix A.1.

For the research documented in this report, NAS benches are relevant for both validating the proposed approach and comparing the results to previous work. This section briefly introduces the NAS benches *NAS-Bench-101* [2], *NAS-Bench-201* [8], and *NATS-Bench* [3].

2.2.1 NAS-Bench-101

NAS-Bench-101 is the first NAS bench that was made publicly available. All architectures in NAS-Bench-101 are trained and evaluated only on the CIFAR10 data set [4]. The architectures in the bench all have the same skeleton which is shown in figure 2.4.

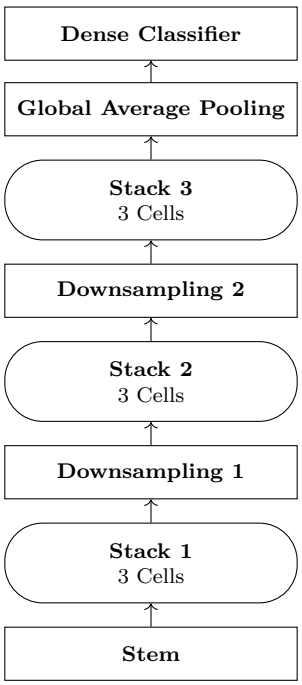

Figure 2.4: Skeleton of the NAS-Bench-101 architectures with the following building blocks: (1) stem: 3×3 convolution layer with 128 channels (2) stack 1: 3 cells, (3) first downsampling layer, (4) stack 2: 3 cells, (5) second downsampling layer, (6) stack 3: 3 cells, (7) global average pooling, (8) dense layer for the final classification. Modified from [2].

The downsampling layer halves the width and the height of the feature maps by average pooling and doubles the depth of the feature map by a 1×1 convolution layer.

A cell is a directed acyclic graph (DAG) with E edges, V nodes and L different labels of the nodes. The nodes can each be one of the following operations: 3×3 convolution, 1×1 convolution and 3×3 max-pooling. All convolutions use batch norm and ReLU activation. Two of the V nodes are the input and output node. The edges represent the feature maps between the nodes. Figure 2.5 shows an example cell from the NAS-Bench-101 cell space.

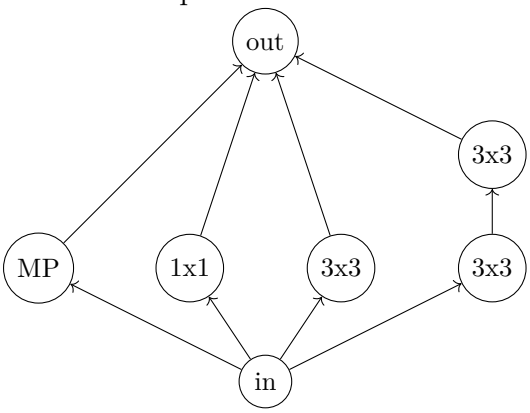

Figure 2.5: One example cell of the cells in the NAS-Bench-101 cell space. *MP* stands for max-pooling, *1x1* for a convolution with a 1×1 filter, *3x3* for a convolution with a 3×3 filter, *in* for the input node and *out* for the output node. The edges are the feature maps between the operations. Modified from [2].

In case of multiple inputs to a single node the following rules are in place to aggregate these: (1) tensors going to the output node are concatenated and (2) those going into other nodes are summed. (3) The output tensors from the input vertex are projected in order to match the expected input channel counts of the subsequent operations. The DAGs have at most E = 9 edges, V = 7 nodes and L = 3 labels of the nodes to limit the search space. This yields 510 million different DAGs in the cell space. Removing DAGs with no connection between input and output node as well as computationally equivalent DAGs leaves 423 000 unique DAGs. Hyperparameters are chosen by a coarse grid search utilizing the mean accuracy over 50 randomly sampled architectures from the space. To provide a proxy for the dispersion of the model performance metrics over different initializations each model is trained 3 times. Each model is further trained with four different numbers of epochs; 4, 12, 36, 108. The learning rate is annealed to 0 via cosine decay, each for the final epoch. Thus $3 \cdot 4 \cdot 423\ 000 \approx 5$ million models are trained in total. For

each training run the data set holds (1) the training accuracy, (2) the validation accuracy, (3) the test accuracy, (4) training time and (5) the number of trainable parameters. [2]

Even though the NAS-Bench-101 comprises a large number of trained architectures, the bench has strong limitations; The models of the bench do not reach state of the art performance on CIFAR10. According to [2] this is mostly because the search space is constrained, no advanced augmentation is used, and no advanced regularization is applied.

2.2.2 NAS-Bench-201 and NATS-Bench

NAS-Bench-201 [8] comprises fewer architectures than NAS-Bench-101, but more fine-grained metrics. Furthermore, the metrics are reported for three datasets: CIFAR10 [4], CIFAR100 [4], and ImageNet16-120 [24] instead of only on CIFAR10. *NATS-Bench* [3] is an extension of the NAS-Bench-201; while NAS-Bench-201 comprises only a *topology search space* (TSS) similar to NAS-Bench-101, NATS-Bench comprises an additional *size search space* (SSS). The architectures in the SSS vary in the number of channels in the convolution layers. Both search spaces are briefly described below. Even though NATS-Bench completely contains NAS-Bench-201, the latter is mentioned here as many publications in the domain still report their results on NAS-Bench-201.

The architectures in the NATS-bench follow almost the same skeleton as NAS-bench-101. The only difference in the skeleton is that the NATS-bench use architectures with one and five cells per stack in the SSS and the TSS, respectively. The downsampling blocks consist of a 2×2 average pooling and a 1×1 convolution to half the spatial dimensions and double the channels of the feature map. The cells of the NATS-bench architectures differ significantly from the cells of NAS-bench-101. In contrast to the NAS-Bench-101 representation, the edges of the cell DAGs are the operations and the nodes are summations of the tensors of all incoming edges. The number of possible operations is $\mathrm{L} = 5$. These are: (1) zeroize[3], (2) skip connection[4], (3) 1×1 convolution, (4) 3×3 convolution, and (5) 3×3 average pooling. Figure 2.6 shows an example of a cell from the NATS cell space.

[3]Multiplication of the input with zero.
[4]Identity operation with the input

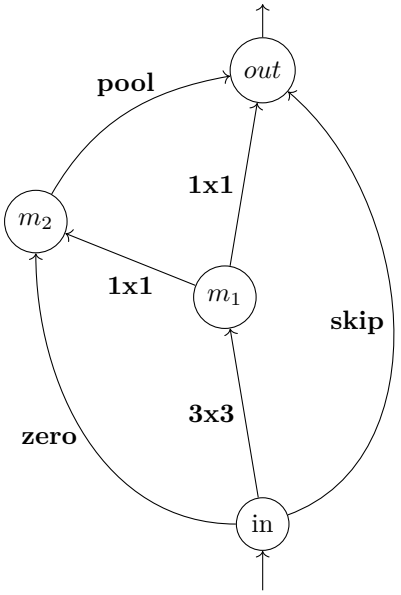

Figure 2.6: One example cell of the cells in the NATS-Bench cell space. The edges are the operations and the nodes are the feature maps between the operations. In case of multiple incoming edges the results from the operations are summed element-wise. *Zero* stands for the zeroize operation, *pool* for a average pooling operation, *skip* for a identity operation and *1x1* and *3x3* for a convolution operation with a filter size of 1×1 and 3×3, respectively. Modified from [3].

Each convolution is followed by batch normalization and a ReLU activation. Each DAG has V = 4 Nodes. This results in 15 625 different architectures in the TSS.

The SSS use the same skeleton and the same cell structure as the TSS. However, as stated above, each stack comprises only a $N = 1$ cell and the six operations in the cell are fixed to be four 3×3 convolutions, one 1×1 convolution and one skip-connect. The cell used in the SSS is the cell with the best performance on CIFAR100 from the TSS. Each of the five convolutions can assume one of the channel numbers in $\{8, 16, 24, 32, 40, 48, 56, 64\}$. Therefore the SSS comprises $8^5 = 32\ 768$ different architectures.

For each architecture the number of parameters, FLOPs, latency as well as the loss end accuracy for training, validation and test data after every epoch are reported. The bench also provides the parameters of the trained architectures. Each architecture is trained for 200 epochs.

2.2.3 Summary: NAS Benches

NAS benches are crucial for this work and for NAS research in general since they (1) make different NAS algorithms comparable and (2) reduce the need for vast computational resources. However, even the best architectures in the above described NAS benches do not reach state of the art performance on image classification benchmarks

such as CIFAR10. The best performing architecture from the 32 768 architectures of the SSS of NATS-Bench, for instance, reaches a test accuracy on CIFAR10 of 93.65 %. One of the highest performing architectures on CIFAR10 in general, *EfficientNet*[25] reaches a test accuracy of 98.90 %[5].

For the validation of the NAS approach proposed in this work, the NATS bench will be use. The primary reason for this choice is that it provides architecture performance metrics not only for CIFAR10 but also for CIFAR100 and ImageNet16-120.

2.3 Previous Work

This section provides an overview of the history of NAS. The different NAS methods in this section are divided into three categories: section 2.3.1 discusses conventional NAS approaches and motivates research into NAS methods that require less training than conventional approaches. Section 2.3.2 discusses NAS approaches that work with partial training of the candidate architectures in order to reduce the computational cost of NAS. The section also points out that these approaches are still computationally expensive, which motivates research into NAS without training of the candidate architectures. NAS without training is discussed in section 2.3.3. The section introduces three of the best performing approaches in NAS without training in detail. The method proposed in this thesis is based on these three approaches. The section is concluded by stating in which aspects the method proposed in this thesis aims to improve over the best performing existing approaches in NAS without training.

2.3.1 Neural Architecture Search with Training

The first implementations of NAS that achieved good results were based on genetic programming [6]: The algorithm randomly creates a generation of candidate architectures which are trained and evaluated. The best performing architectures are then used to create a next generation of candidate architectures by 1.) applying small modifications (*mutation*), 2.) combining them (*cross-over*) and 3.) keeping them to the next generation (*selection*). Iteratively producing new generations of candidate solutions in this fashion yields good architectures for certain problems. However, the approach turned out not to scale to tasks that require deep architectures, such as image recognition [26]. The first NAS algorithms that were on par with deep, handcrafted architectures were reinforcement learning (RL) based: the NAS algorithm *MetaQNN*[7] uses a Q-learning agent to build neural architectures by iteratively picking layers and hyperparameters. Another reinforcement learning based algorithm, *NAS-RL*[1], generates architecture descriptions with a recurrent neural network that was trained using a policy gradient reinforcement learning algorithm. Both approaches have two drawbacks: they are severely limited in the selection of layers and hyperparameters to constrain the state and action space

[5]Note that this comparison is not completely fair since the training settings are not identically. However, it is still considered to serve as a good indication for the performance of the two approaches relative to each other.

for the agent and they are computationally expensive with multiple thousands of GPU hours per data set. Table 2.2 shows the training time and the performance of the resulting architectures on the image classification benchmarks CIFAR10[4] and ImageNet[5], for MetaQNN and NAS-RL. The enormous computational cost of these conventional NAS algorithms motivates research into NAS methods that train the architectures only partially. Some representatives of such approaches are presented in the subsequent section.

Table 2.2: Test accuracy of different NAS algorithms on the CIFAR10 [4] and ImageNet [5] image classification benchmarks.

	Algorithm	**CIFAR10 Test Acc**	**ImageNet Test Acc**	**GPU Time**
Handcrafted	ResNeXt-101 [27]	-	80.9 %	-
	ResNeXt-29 [27]	96.42 %	-	-
NAS	NAS-RL [1]	96.35 %	-	22 400 d
with Training	MetaQNN [7]	93.08 %	-	10 d
Partial	CNFs [28]	92.57 %	-	-
Training	AmoebaNet-A [12]	96.66 %	83.9 %	3 150 d
	NASH [29]	95.30 %	-	2 d
Without	NASWOTv1 [14]	91.61 %	36.37 %	17 s
Training	NASWOTv3 [15]	93.10 %	45.05 %	248 s
	EPE-NAS [16]	91.31 %	41.84 %	206 s

2.3.2 Neural Architecture Search with Partial Training

The computational cost of the above mentioned NAS approaches gives rise to the research discipline *NAS with partial training*. The objective of this discipline is to develop NAS methods that maintain a high performance while being less computationally expensive than methods that rely on full training of all candidate architectures. One can divide methods with partial training into four categories [30]: *low fidelity estimates*, *learning curve extrapolation*, *weight inheritance and network morphisms*, and *one-shot models*. This section briefly describes each of the categories and provides an example of an algorithm from each category. Table 2.2 shows the performance and training times of these algorithms compared to the conventional NAS algorithms from section 2.3.1. Table 2.2 also provides the performance of two *ResNeXt*[27] architectures as representatives of state-of-the-art handcrafted architectures.

A *one-shot NAS algorithm* is an algorithm that creates candidate architectures as sub-graphs of a huge super-graph. The super graph is trained once (one-shot). The sub-graphs then use parts of the trained weights from the super-graph. The candidate architectures can be fine-tuned but are not extensively trained again. One such approach is *Convolutional Neural Fabric* (CNF) [28]. A CNF is a densely connected network of trained layers. The network of layers is trained as a whole. Different architectures are created by pursuing different paths through the network. Overlapping paths share the weights in the layers. CNFs achieve a test accuracy of 92.57 %

on CIFAR10. The training time of the CNF is not reported.

Another way of reducing the training time for NAS is to use *low fidelity estimates* for the performance of the architectures instead of the final validation accuracy. These estimates can for example be the accuracy after fewer training epochs or the accuracy of down-scaled models. *AmoebaNet-A*[12] is one successful representative from the category of algorithms that reduce the training time of NAS by utilizing low-fidelity performance estimates. *AmoebaNet-A* uses an evolutionary algorithm that trains each architecture for only 25 epochs in each generation. Only the 20 architectures that perform best after the evolution are trained for the full 600 epochs. For example, *AmoebaNet-A* reduces the required computational cost from 22 400 GPU days for NAS-RL to 3 150 GPU days while yielding a slightly higher final test accuracy of 96.66 % in CIFAR10.

A representative for a NAS algorithm that uses *learning curve extrapolation* is [31]. [31] uses different regression models to predict the final validation accuracy based on features as network architectures, hyperparameters, and time-series validation performance data. [31] do not report final test accuracy on CIFAR10 or ImageNet as the focus of the article lies rather on the accurcy of the learning curve prediction. They state, however, that the algorithm achieves a 6x speed-up while maintaining the performance of MetaQNN.

An approach that uses *network morphisms* to reduce the computational cost of NAS is NASH [29]. The algorithm iterates over the following steps: (1) a set of candidate architectures is trained for a small number of epochs. (2) The architecture with the highest validation performance is picked as a parent architecture and (3) a new set of candidate architectures is generated by applying simple morphisims to the parent architecture. An example of a simple network morphisim is to double an existing layer. Most of the network remains unchanged. Therefore most of the weights can be inherited from the parent architecture. NASH yields 95.30 % test accuracy on CIFAR10 after only two GPU days.

To conclude, it can be stated that partial training methods achieve results that are on par with conventional NAS methods at a lower training cost. However, the computational budget for a full NAS run comprises still at least multiple GPU days. If one wants to outperform handcrafted architectures it requires even several thousands of GPU days. The still high resource demand of NAS with partial training motivates research into *NAS without training*, which will be discussed in section 2.3.3.

2.3.3 Neural Architecture Search without Training

Research on how to reduce the computational cost of NAS even further focuses on eliminating the training of the architectures by proxy scores for the validation accuracy that can be computed without training the architecture (*NAS score* for short). The objective when developing a NAS score is to achieve a high correlation between

the score and the validation accuracy after training. A means to assess the goodness of a NAS score is thus the correlation coefficient between the NAS score and the validation accuracy over a set of candidate architectures.

NAS scores can be grouped into two categories: Firstly, *pruning scores* that are repurposed as a NAS score and secondly scores that are directly developed for the purpose of NAS. This section briefly elucidates the concept of pruning scores for NAS and then describes the higher performing dedicated NAS scores in detail.

A *pruning score* is a scoring metric for parameters in a neural network which is supposed to indicate the importance of the parameter for the model performance. When pruning parameters in a neural network, parameters with a lower score are pruned before parameters with a higher score. An aggregation, e.g. the mean, of the pruning scores over all parameters in a network can be used to score an entire architecture for the purpose of NAS. The underlying hypothesis is that an architecture with high pruning scores performs better than an architecture with low pruning scores. [32] uses gradient based pruning scores, like *snip*[33] and *synflow*[13] to score the architectures for NAS.

[32] does not report the performance of architectures found by the *snip* and the *synflow* score on any image classification benchmark. Instead, the correlation of the scores with the validation accuracy for different architectures is reported. The correlation of these pruning scores with the validation accuracy is lower than for scores that are specifically developed for NAS [14]. Therefore, the following sections focus on the higher performing NAS scores, specifically developed for NAS. Sections 2.3.3.1, 2.3.3.2 and 2.3.3.3 describe the, to the best of our knowledge, best performing NAS scores to date. Section 2.3.3.4 concludes this chapter with a summary on how the score proposed in this thesis aims to improve over the presented, existing approaches.

2.3.3.1 Mellor et. al.'s NAS without Training Version 1

To the best of our knowledge, the highest performance of a training-free NAS algorithm to date is achieved by an algorithm called NASWOT (short for *NAS Without Training*) [15]. Two different versions of the NASWOT score have been published under the same name. During the following they are referred to as NASWOTv1[14] and NASWOTv3[15][6]. A full NAS with NASWOTv1 and NASWOTv3 takes only 17 and 248 seconds, respectively. This is several orders of magnitude less than NAS-with-training methods (see table 2.2). However, with test accuracies of 91.61 % and 93.10 %, respectively, on CIFAR10, there is also a significant performance deficit to conventional, state of the art NAS methods (see table 2.2). This motivates further research in the domain *NAS without training* with the objective to increase the correlation of the NAS score of the untrained network with the validation accuracy of

[6] [15] published another version of the article in between NASWOTv1 and NASWOTv3. The algorithm in the second version NASWOTv2, however, is identical to the algorithm in the third version NASWOTv3. Therefore it is referred to the newest version of the article, NASWOTv3, here.

the trained network. The approach that is proposed in this thesis is based on the ideas of NASWOTv1 and NASWOTv3, which are introduced in this section and the subsequent section, respectively.

The score from NASWOTv1 is computed based on the LLOs $\lambda^{(1)} \dots \lambda^{(256)}$ corresponding to a batch X of 256 data points $x^{(1)} \dots x^{(256)}$. The 256 data points are randomly sampled from the training data. The score s is computed as follows: the LLOs are arranged in a matrix:

$$J = \left(\lambda^{(1)} \dots \lambda^{(256)}\right)^T \in \mathbb{R}^{256 \times N}. \tag{2.14}$$

The LLOs are assumed to be vectors with N elements equal to the number of inputs of the networks. If the input shape of the networks is a matrix or a tensor, as is usually the case in image classification, the LLOs can be transformed into vector shape by a flatten operation. The matrix J can be considered the Jacobian of the network with respect to a batch of input data points since the LLOs are the local gradient of the network with respect to the input data points (see section 2.1). [14] then compute the covariance matrix

$$C_J = (J - M_J)(J - M_J)^T \tag{2.15}$$

of the LLOs with the elements $m_{i,j}$ of M_J holding the mean of the ith LLO:

$$m_{i,j} = \frac{1}{N} \sum_{n=1}^{N} J_{i,n} \tag{2.16}$$

The elements in C_J describe how the LLOs co-vary. Intuitively, for the semantics of the elements in C_J it can be stated that the element $c_{i,j}$ describes how similar the network maps the ith and the jth data point from batch X. To make this quantity scale invariant over the different inputs [14] transform the covariance matrix C_J into the correlation matrix Σ_J by computing the elements of Σ_J to

$$\sigma_{i,j} = \frac{c_{i,j}}{\sqrt{c_{i,i} \cdot c_{j,j}}} \tag{2.17}$$

based on the elements $c_{i,j}$ of C_J. The score s can then be computed as the negative Kullback-Leibler (KL) divergence between a uncorrelated multivariate Gaussian distribution and a Gaussian distribution with the kernel Σ_J:

$$s = -\sum_{i=1}^{N} \left(log(\nu_{J,i} + k) + (\nu_{J,i} + k)^{-1}\right) \tag{2.18}$$

$\nu_{J,i}$ are the N eigenvalues of Σ_J and k is a constant for numeric stability.

An intuition for the mechanics of the NASWOTv1 score can be formulated as follows: The lower the correlation between the LLOs of different data points is, the lower the off-diagonal elements of Σ_J. Low off-diagonal elements result in a low KL divergence with an uncorrelated multivariate Gaussian, since the uncorrelated Gaussian kernel is the identity matrix. The score is the negative KL divergence.

The negation of the low KL divergence results in a high score. Consequently, the score is high when the LLOs are uncorrelated and low when they are correlated. The objective that the NASWOTv1 score reflects is thus that all data points are mapped as differently as possible by the network.

The above stated objective reflected by the NASWOTv1 is remarkable; while it is intuitive that the mapping of two data points from different classes should be different, one should expect that data points from the same class should be mapped similarly. NASVOTv1, however, implies that mapping two data points from the same class differently also increases the score of a network. This thesis therefore hypothesizes that the architectures that score the highest with NASWOTv1 are *overly complex*, which means that the architectures have a higher discrimination potential than what is needed for the task. One disadvantage of an overly complex architecture is that it overfits the training data easily, which leads to a lower test performance.

2.3.3.2 Mellor et. al.'s NAS without Training Version 3

Similar to NASWOTv1, NASWOTv3 is based on a single batch X of data that is randomly sampled from the training data set. [15] finds that a batch size of $N = 128$ is sufficient to achieve a good correlation of the score with the validation accuracy. The underlying hypothesis of [15] is that two data points with a similar activation pattern are difficult to distinguish for the network while data points with different activation patterns are easier to distinguish. Therefore, NASWOTv3 bases the score, on the activation patterns that the data points of the batch cause in the network. [15] represent an activation pattern as a binary code c that encodes a specific activation pattern consists of one bit per neuron in the network. A bit c_i is one when the corresponding ith[7] neuron h_i in the network is active and zero when the neuron is dead. The score is computed by arranging the pairwise Hamming distances $d_H(c_i, c_j)$ between the binary codes of the data points of the batch in a matrix and subtracting it from the total number of neurons in the network N_A:

$$K_H = \begin{bmatrix} N_A - d_H(c_1, c_1) & \dots & N_A - d_H(c_1, c_N) \\ \vdots & \ddots & \vdots \\ N_A - d_H(c_N, c_1) & \dots & N_A - d_H(c_N, c_N) \end{bmatrix} \tag{2.19}$$

The final score s is the logarithm of the determinant of the matrix K_H:

$$s = log(det|K_H|) \tag{2.20}$$

An intuition for the mechanics of the NASWOTv3 score can be formulated as follows: The hamming distance $d_H(c_i, c_j)$ is the number of bits by which the two activation patterns c_i and c_j differ. Since N_A is equal to the number of bits in the activation patterns c_i and c_j, $N_A - d_H(c_i, c_j)$ corresponds to the number of bits the two activation patterns c_i and c_j have in common. The less similar the activation

[7] It is assumed that each neuron in the network gets assigned an index. The neurons of the network can be arbitrarily ordered as long as the order is the same for all images in the batch.

patterns of the data points in the batch are, the lower the off-diagonal elements in K_H will be compared to the elements on the main diagonal. Note that all elements on the main diagonal of K_H are N_A since the hemming distances $d_H(c_i, c_i) = 0 \ \forall i$. With the off-diagonal elements becoming lower relative to the elements on the main diagonal, the volume and thus the determinant of the matrix increases. The score is thus high if the activation patterns of all data points in the batch are as different as possible from all other activation patterns in the batch. The logarithm is a strictly monotonic function and does thus not influence the described causality.

[15] is the only of the three training-free NAS approaches that are presented in this work which reports the correlation of the score with the validation accuracy of candidate architectures directly: On 1000 randomly sampled architectures from the NAS-Bench-201 the Kendall's Tau correlation coefficient is $\tau = 0.574$. Even if the other approaches do not report the correlation directly it can, based on the performance of the found architectures, be stated that NASWOTv3 is the best performing NAS score among the presented approaches. See table 2.2 for a performance comparison between the architectures found using the different approaches.

The higher performance of NASWOTv3 compared to NASWOTv1 indicates that the activation patterns used by NASWOTv3 reflect the capabilities of the network better then the LLOs used by NASWOTv1. This thesis hypothesizes that this is because the activation patterns are more robust against weight changes during training: Changing a parameter of the network during the training does immediately change the LLO, while one needs to change it by a certain margin before the activation pattern changes. It is therefore possible that the activation patterns are representing the capabilities of the network more robustly through the training then the LLOs do.

Similar to NASWOTv1, NASWOTv3 encodes in the objective of the NAS score that all data points shall be mapped as differently as possible by the network, regardless of the class of the data point. This objective is considered to be sub-optimal for the same reasoning as previously discussed for NASWOTv1. The next section describes a training-free NAS approach that is based on NASWOTv1 which incorporates the class of the data point in the score.

2.3.3.3 Efficient Performance Estimation Without Training for Neural Architecture Search

Efficient Performance Estimation Without Training for Neural Architecture Search (EPE-NAS) is a training-free NAS approach that is based on NASWOTv1. The main difference to NASWOTv1 is that EPE-NAS incorporates the classes of the data points in the score computation. This section briefly describes the computation of the EPE-NAS score as well as which objective the score encodes.

The computation of the EPE-NAS score is very similar to the one from the NASWOTv1 score with the following differences: EPE-NAS constructs one Jacobian per class instead of just a single Jacobian for all data points from the batch: Assuming

Q classes, the Jacobian of all LLOs from class $q \in \{1, \ldots, Q\}$ is given as

$$J_q = \left(\lambda^{(p1)}, \ldots, \lambda^{(pn)}\right) \tag{2.21}$$

if the corresponding data points

$$x^{(p1)}, \ldots, x^{(pn)} \in \textit{class } q. \tag{2.22}$$

The covariance matrices

$$C_{J,q} = (J_q - M_{J,q})(J_q - M_{J,q})^T \tag{2.23}$$

with the elements $m_{i,j}$ of $M_{J,q}$ holding the mean of the ith LLO in class q

$$m_{i,j} = \frac{1}{N} \sum_{n=1}^{N} J_{q,i,n} \tag{2.24}$$

and the correlation matrices $\Sigma_{J,q}$ with its elements

$$\sigma_{q,i,j} = \frac{c_{q,i,j}}{\sqrt{c_{q,i,i} \cdot c_{q,j,j}}} \tag{2.25}$$

are also computed for each class separately. The correlation matrices $\Sigma_{J,q}$ are aggregated towards a scalar score in two steps[8]: in the first step the absolute values of the elements in the correlation matrices are logarithmized and summed for each class q:

$$e_q = \sum_{i=1}^{N} \sum_{j=1}^{N} log(|\sigma_{q,i,j}| + k) \tag{2.26}$$

k is a small constant for numeric stability. In the second step all e_qs are summed, which results in the final score:

$$s = \sum_{i=1}^{Q} e_q \tag{2.27}$$

An intuition for the mechanics of the EPE-NAS score can be formulated as follows: as previously discussed the LLOs determine how the network maps the data points. The more similar the LLOs the more similar maps the network the data points. The higher the correlation between the LLOs of the data points within one class, the higher are the off-diagonal elements of the correlation matrices Σ_q. The sum e_q of the absolute elements $|\sigma_{q,i,j}|$ in Σ_q grows with the absolute values of the off-diagonal ellements of the matrix as the diagonal elements are constant and 1. The absolute operation achieves that positive and negative correlations between LLOs from the same class equally contribute to a higher score. The objective that EPE-NAS score represents is therefore that for a good architecture the mappings of data points from the same class are similar.

This thesis hypothesizes that the objective of the EPE-NAS score is sub-optimal

[8]If 100 or more classes are present in the batch [16] computes the aggregation of the matrices differently. This path of the score computation is omitted here for simplicity.

in that it does not require a high inter-class difference. If a network would, in its output space, collapse all data points in the same cluster it could theoretically still score high, while the network would probably have difficulties to distinguish the classes during the training. Consequently, it can be stated that while EPE-NAS in contrast to NASWOTv1 and NASWOTv3 incorporates the class information in the score, it does so only sub-optimally.

EPE-NAS performs approximately equally well on CIFAR10 as NASWOTv1 but outperforms NASWOTv1 by a large margin on ImageNet (see table 2.2). [16] hypothesizes that this is due to the incorporation of the data point classes in the score computation. NASWOTv3, however, performs better than EPE-NAS. This thesis hypothesizes that the idea from EPE-NAS of differentiating the score objective depending on the class would also be beneficial to NASWOTv3.

2.3.3.4 Improvements over Existing Methods in NAS Without Training in this Thesis

In conclusion, two potential aspects of improvement are identified in NASWOTv1 and NASVOTv3:

1. The objective which NASWOTv1 and NASWOTv3 reflect is for a network to map all data points, regardless of from which class, as differently as possible from all other data points. This is likely favors overly complex networks.
2. NASWOTv1 is based on the LLOs of the network. The LLOs are composed of the parameters of the network. The parameters of the network are subject to changes during the training. This leads to changes of the LLOs and consequently the score of the network during the training. Computing an LLO-based score prior to training is thus likely to reflect the properties of the trained network sub-optimally.

This thesis proposes a novel method how to analyse LLOs and suggests a score that aims to improve in the two mentioned aspects as follows:

1. The objective of the score is changed so that a network that maps points from the same class similarly and points from different classes differently scores the highest.
2. Instead of basing the score directly on the LLOs, the score is based on the pairwise dependence of the LLOs. This aims to reflect how the network can adjust the LLOs relative to each other during the training instead of assessing how the LLOs are placed relative to each other before the training.

EPE-NAS is based on NASWOTv1 and improves upon NASWOTv1 by *partially incorporating the class* of the data points in the score. To partially incorporate the class in the score means in the case of EPE-NAS that architectures that map data points from the same class similarly score high. However, the objective of the EPE-NAS approach does not reflect whether the network is capable to discriminate between different classes. The score that is proposed in this thesis aims to improve

over EPE-NAS by incorporating both a high within-class similarity and a high inter-class difference of the LLOs in the score objective.

The following chapter, chapter 3, formulates the objective of the proposed NAS score in detail, firstly in plain text and then mathematically.

3
Methods

The objective of this work is to develop a scalar score for NAS that is correlated with the final validation accuracy of architectures before the architectures are exhaustively trained. The score can then be used to compare architectures without training them.

An architecture is assumed to be optimal for a particular task on a particular data set, in contrast to being optimal for all tasks and data sets. Therefore the score has to take the instances from the data set in question as well as their annotations into account. The idea for a score design presented in this chapter fulfills this requirement. Similar to [14] and [16] the score is based on the LLOs for a batch of data set instances. In contrast to [14] and [16] however, the objective for how similarly two data points should be mapped through the network is based on the parameter dependence of the local linear operators. This chapter describes the design of the score and validation methodology of the score. Section 3.1 explains the concept *parameter dependence of local linear operators*. Section 3.3 formulates objectives of NAS in terms of local linear operators and their parameter dependence and section 3.4 describes how these objectives can be captured by a proxy score. The setup of the experiments that are conducted to validate the resulting proxy score is described in 3.5.

3.1 Parameter Dependence of Local Linear Operators

One of the contributions of this work is to base the score on the mutual dependency of the LLOs instead of the values of the LLOs only. This is motivated by the fact that the score is computed prior to training. The values of the LLOs are different before and after training since the parameters of the architecture which the LLOs are composed of change during the training. Including the mutual dependency of the LLOs in the score allows for assessment of how dependently or independently the LLOs can be changed during training. This section proposes one way of quantifying the mutual dependency of LLOs.

LLOs of the same network are not independent from each other. Their elements are composed of intersecting subsets of network parameters. For instance, the ele-

ments $\lambda_1^{(C1)}$ and $\lambda_2^{(C1)}$ of the LLO

$$\lambda^{(C1)} = \left[\lambda_1^{(C1)}, \lambda_2^{(C1)}\right] = \left[w_{11} \cdot w_{21} + w_{12} \cdot w_{22}, w_{13} \cdot w_{21} + w_{14} \cdot w_{22}\right]. \tag{3.1}$$

from the example in section 2.1 are composed by the parameter sets

$$\omega_1^{(C1)} = \{w_{11}, w_{12}, w_{21}, w_{22}\} \tag{3.2}$$

$$\omega_2^{(C1)} = \{w_{13}, w_{14}, w_{21}, w_{22}\}. \tag{3.3}$$

These parameter sets provide a more fine-grained view on the LLOs that is for instance conclusive regarding the dependence of two LLO elements. During the following the parameter sets of the p-element LLO

$$\lambda^{(k)} = \left[\lambda_1^{(k)}, \ldots, \lambda_p^{(k)}\right] \tag{3.4}$$

will be denoted, arranged in a vector

$$\omega^{(k)} = [\omega_1^{(k)}, \ldots, \omega_p^{(k)}] \tag{3.5}$$

of the same dimensions as the referring LLO $\lambda^{(k)}$. $\omega_n^{(k)}$ is the subset of parameters of the network that influence the nth element of the LLO $\lambda^{(k)}$.

These parameter sets can be utilized to quantify how dependent two elements $\lambda_1^{(k)}$ and $\lambda_2^{(k)}$ of an LLO are. If a parameter w_{ij} is element of the parameter sets $\omega_1^{(k)}$ and $\omega_2^{(k)}$, changing this parameter during training will change both, $\lambda_1^{(k)}$ and $\lambda_2^{(k)}$ simultaneously. If the parameter w_{ij} is element of only one of the two parameter sets, $\omega_1^{(k)}$ and $\omega_2^{(k)}$, changing this parameter will change only one of the LLO elements. The larger the intersection

$$\omega_1^{(k)} \cap \omega_2^{(k)} \tag{3.6}$$

the higher the mutual dependence of the referring LLO elements $\lambda_1^{(k)}$ and $\lambda_2^{(k)}$. The same principal of quantifying mutual dependence can be applied across two LLOs.

3.2 Local Linear Operator Dependence - An Example

For the LLO dependence introduced in section 3.1 to be a promising indicator for a data specific NAS score, it is important that different inputs lead to different LLO dependencies. If this is not the case the data will not influence the LLO dependencies since the dependencies would be the same for all inputs. An LLO dependency based NAS score would consequently have no discrimination potential to find the optimal architecture for a specific data set. In this section, an example is used to illustrate the computation of the parameter sets and how their dependency changes over different inputs. This is a crucial concept for the score design in section 3.4.

Different inputs typically lead to different activation patterns in a network. In this example the parameter sets resulting from two different activation patterns of a

small neural network shall be compared. A fully connected architecture is considered first, followed by a locally connected architecture. The fully connected architecture is a network with three inputs, two hidden layers with each three ReLUs, and a single output neuron with identity activation. Figure 3.1 shows the network. Compared are the parameter sets in $\omega^{(1)}$ and $\omega^{(2)}$ for the LLOs of two randomly selected activation patterns, $\alpha^{(1)}$ and $\alpha^{(2)}$. Each activation pattern is represented by the set of ReLUs that are dead given the activation pattern:

$$\alpha^{(1)} = \{h_{11}, h_{12}, h_{22}\} \tag{3.7}$$

$$\alpha^{(2)} = \{h_{12}, h_{21}, h_{23}\} \tag{3.8}$$

The superscript indicates which activation pattern α leads to which parameter sets ω. The resulting parameter sets[1] are shown in equation 3.9 and 3.10.

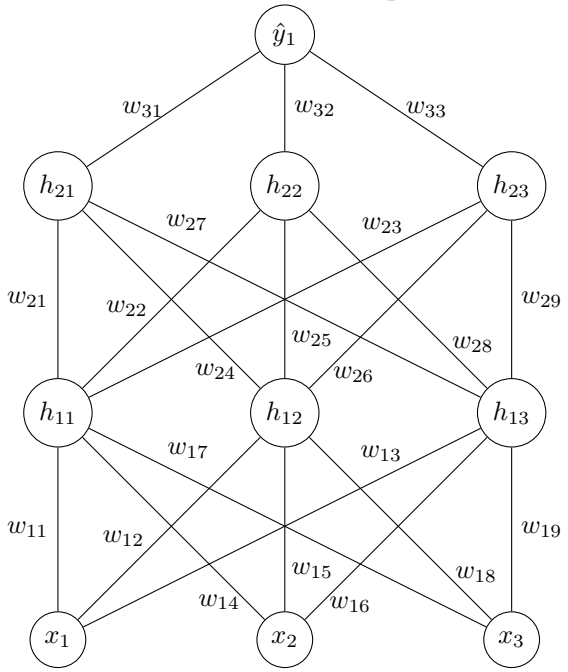

Figure 3.1: Fully connected network with three inputs x, two hidden layers with each three units, and a single output neuron $\hat{y}_1$.

$$\omega^{(1)} = \begin{bmatrix} \omega_1^{(1)} \\ \omega_2^{(1)} \\ \omega_3^{(1)} \end{bmatrix}^T = \begin{bmatrix} \{w_{13}, w_{27}, w_{29}, w_{31}, w_{33}\} \\ \{w_{16}, w_{27}, w_{29}, w_{31}, w_{33}\} \\ \{w_{19}, w_{27}, w_{29}, w_{31}, w_{33}\} \end{bmatrix}^T \tag{3.9}$$

$$\omega^{(2)} = \begin{bmatrix} \omega_1^{(2)} \\ \omega_2^{(2)} \\ \omega_3^{(2)} \end{bmatrix}^T = \begin{bmatrix} \{w_{11}, w_{13}, w_{22}, w_{28}, w_{32}\} \\ \{w_{14}, w_{16}, w_{22}, w_{28}, w_{32}\} \\ \{w_{17}, w_{19}, w_{22}, w_{28}, w_{32}\} \end{bmatrix}^T \tag{3.10}$$

[1]Note that the parameter set $\omega_n^{(m)}$ can be obtained by aggregating the parameters along all feasible paths through the network with start at input x_n and end at output $\hat{y}_1$. A path is feasible if it does not contain any of the dead ReLUs in $\alpha^{(m)}$.

One can observe that for the fully connected network, the parameter sets in $\omega^{(1)}$ and $\omega^{(2)}$ vary depending on the activation pattern. Across a single activation pattern, however, the parameter sets differ only by parameters of the first layer. For example the first two elements of $\omega^{(1)}$:

$$\omega_1^{(1)} = \{w_{13}, w_{27}, w_{29}, w_{31}, w_{33}\} \tag{3.11}$$

and

$$\omega_2^{(1)} = \{w_{16}, w_{27}, w_{29}, w_{31}, w_{33}\} \tag{3.12}$$

differ only by the parameter w_{13} and w_{16}, respectively, while the parameters of the second and third layer are the same in both sets. Similar observations can be made for all pairs of parameter sets of the same LLO. Consequently, changing a parameter from the LLO always changes all LLO elements.

A CNN is considered in the next step. To keep the example concise a CNN with three inputs, two one-dimensional convolution layers and a single output neuron is chosen. The convolution layers have two and one channel, respectively. The hidden neurons have ReLU activation. The output layer is fully connected and has identity activation. Figure 3.2 shows the network. As for the fully connected network the parameter sets for two different, randomly selected activation patterns

$$\alpha^{(3)} = \{h_{11}, h_{14}, h_{12}, h_{22}\} \tag{3.13}$$

$$\alpha^{(4)} = \{h_{14}, h_{15}, h_{16}, h_{23}\} \tag{3.14}$$

shall be compared. The parameter sets resulting from the two activation patterns $\alpha^{(3)}$ and $\alpha^{(4)}$, $\omega^{(3)}$ and $\omega^{(4)}$ are shown in equation 3.15 and 3.16.

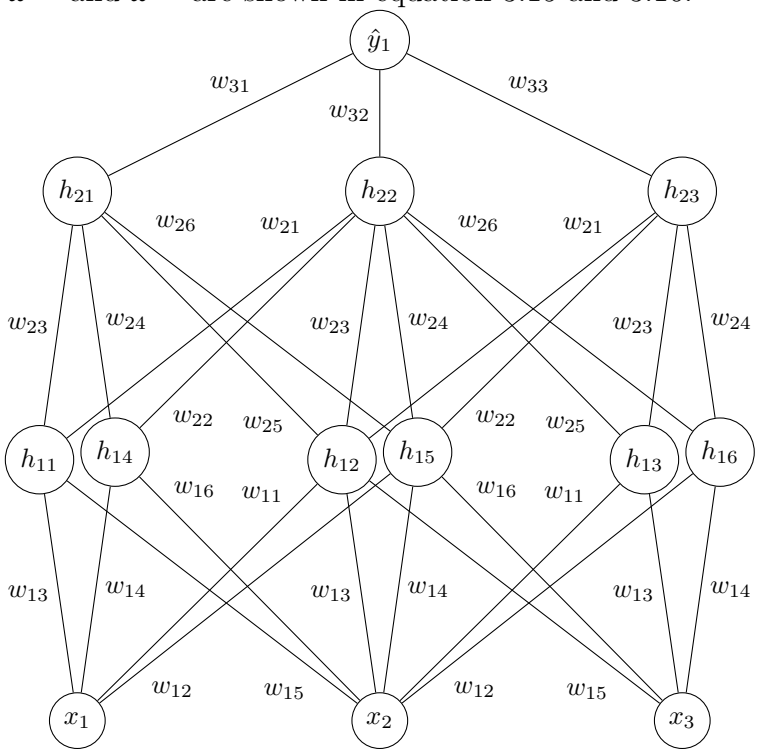

Figure 3.2: CNN with three inputs x, two hidden 1D-convolution layers with two and one channel, respectively, and a single output neuron o.

$$\omega^{(3)} = \begin{bmatrix} \omega_1^{(3)} \\ \omega_2^{(3)} \\ \omega_3^{(3)} \end{bmatrix}^T = \begin{bmatrix} \{w_{12}, w_{22}, w_{26}, w_{31}, w_{33}\} \\ \{w_{11}, w_{12}, w_{14}, w_{22}, w_{23}, w_{24}, w_{26}, w_{31}, w_{33}\} \\ \{w_{13}, w_{14}, w_{16}, w_{22}, w_{23}, w_{24}, w_{26}, w_{31}, w_{33}\} \end{bmatrix}^T \tag{3.15}$$

$$\omega^{(4)} = \begin{bmatrix} \omega_1^{(4)} \\ \omega_2^{(4)} \\ \omega_3^{(4)} \end{bmatrix}^T = \begin{bmatrix} \{w_{11}, w_{13}, w_{21}, w_{23}, w_{25}, w_{31}, w_{32}\} \\ \{w_{11}, w_{13}, w_{15}, w_{21}, w_{23}, w_{25}, w_{31}, w_{32}\} \\ \{w_{13}, w_{15}, w_{23}, w_{25}, w_{31}, w_{32}\} \end{bmatrix}^T \tag{3.16}$$

For the CNN the parameter dependencies vary between the two activation patterns as well. In contrast to the fully connected network the parameter dependencies between the different elements of the LLOs vary by parameters of both hidden layers. That means that changing parameters locally only influences some of the LLO elements.

The examples are insufficient to support the hypothesis that LLOs and their parameter dependencies could be a good indicator for a NAS score. For the small exemplary networks considered here however, the parameter dependencies of the LLOs vary significantly with the activation pattern. The activation pattern evidently changes with the architecture. Thus the experiments do not reject the hypothesis either.

For the locally connected network, i.e. the CNN, the example indicates that there are significantly different parameter dependencies even within a single LLO. This has important implications on the score design in section 3.4; the results differ depending on whether one first computes some quantification of element-wise mutual dependencies and aggregates afterwards or first aggregates the parameter sets over the LLOs and then computes the mutual dependencies. In more generic terms; the mutual dependence of the LLOs has to be examined on the element level, not on the LLO level.

The minimal examples above indicate that the parameter sets in different ω differ depending on the input. This is an important premise for the proposed data specific NAS approach as an input-invariant metric would not be able to include the influence of the data. Whether the variations of the parameter dependencies are in any way correlated with the goodness of the architecture, however, is difficult to show with this kind of theoretical consideration. The practical experiments described in section 3.5 are conducted for this reason.

3.3 Objective of the Proxy Score

The NAS score objective is to be high for an architecture with a high validation accuracy and low for an architecture with a low validation accuracy. This section formulates the objective of a NAS score in terms of the LLOs and their mutual parameter dependence.

In a classification task an architecture will have a high validation accuracy when it produces similar outputs for instances of the same class and distinct outputs for instances of different classes. How a network maps a particular input can be represented by the LLO of the LLR in which the respective input lies. One approach to score an architecture would be to evaluate the pair-wise similarity of the LLOs for the training data. For pairs from the same class, a similar mapping and therefore similar LLOs are desired and for pairs from different classes distinct mappings and therefore distinct LLOs are desired. If the score is computed before the network is trained however, the LLOs will change, as the network's parameter that they are composed of change during training. A score based solely on the LLOs will thus merely have limited correlation with the validation accuracy after the training. Therefore, instead of evaluating the similarity of the LLOs directly, this work proposes to evaluate how capable the network is to bring the LLOs into the desired setup during training. This can be achieved utilizing the concept of parameter dependence. If, for instance, the LLOs of two data points from the same class are distinct, they can only then be annealed during training if they depend on different parameters of the network. Otherwise, if the LLOs depend on the same parameters in the network, changing the parameters would only change the two LLOs simultaneously and never bring them closer together. If two LLOs of the same class in turn are already similar, it is a desired property of the LLOs to depend on approximately the same sets of parameters. The reason is that LLOs that depend on the same parameters cannot diverge during training. Similar objectives for the LLO dependence can be stated for data points from different classes: If the LLOs of two data points from different classes are similar, they can only be separated during training if they depend on different parameters of the network. Otherwise, if the LLOs depend on the same parameters in the network, changing the parameters would only change the two LLOs simultaneously and never move them apart. If two LLOs of the same class in turn are already different, it is a desired property of the LLOs to depend on approximately the same sets of parameters. This is since LLOs that depend on the same parameters cannot converge during the training.

Equation 3.21 provides an enumeration of the above described cases of parameter dependence together with the desired score behavior in a more formal notation. The cases are identified in terms of two LLOs

$$\lambda^{(i)} = \left[\lambda_1^{(i)}, \ldots, \lambda_k^{(i)}\right] \tag{3.17}$$

$$\lambda^{(j)} = \left[\lambda_1^{(j)}, \ldots, \lambda_k^{(j)}\right], \tag{3.18}$$

the two referring parameter set vectors

$$\omega^{(i)} = \left[\omega_1^{(i)}, \ldots, \omega_k^{(i)}\right] \tag{3.19}$$

$$\omega^{(j)} = \left[\omega_1^{(j)}, \ldots, \omega_k^{(j)}\right], \tag{3.20}$$

and the labels y_i and y_j of two instances, x_i and x_j, of the training data set. The notations $a \approx b$ and $a \not\approx b$ indicate that the two operands are similar and dissimilar,

respectively. How similarity can be defined in case of the LLO elements and in case the parameter sets is discussed in detail in section 3.4.

$$s_{i,j} = \begin{cases} (1)\ high & \text{if } y_i = y_j,\ \lambda_n^{(i)} \approx \lambda_n^{(j)},\ \omega_n^{(i)} \approx \omega_n^{(j)} \\ (2)\ low & \text{if } y_i = y_j,\ \lambda_n^{(i)} \approx \lambda_n^{(j)},\ \omega_n^{(i)} \not\approx \omega_n^{(j)} \\ (3)\ low & \text{if } y_i = y_j,\ \lambda_n^{(i)} \not\approx \lambda_n^{(j)},\ \omega_n^{(i)} \approx \omega_n^{(j)} \\ (4)\ high & \text{if } y_i = y_j,\ \lambda_n^{(i)} \not\approx \lambda_n^{(j)},\ \omega_n^{(i)} \not\approx \omega_n^{(j)} \\ (5)\ low & \text{if } y_i \neq y_j,\ \lambda_n^{(i)} \approx \lambda_n^{(j)},\ \omega_n^{(i)} \approx \omega_n^{(j)} \\ (6)\ high & \text{if } y_i \neq y_j,\ \lambda_n^{(i)} \approx \lambda_n^{(j)},\ \omega_n^{(i)} \not\approx \omega_n^{(j)} \\ (7)\ high & \text{if } y_i \neq y_j,\ \lambda_n^{(i)} \not\approx \lambda_n^{(j)},\ \omega_n^{(i)} \approx \omega_n^{(j)} \\ (8)\ low & \text{if } y_i \neq y_j,\ \lambda_n^{(i)} \not\approx \lambda_n^{(j)},\ \omega_n^{(i)} \not\approx \omega_n^{(j)} \end{cases} \quad \forall n, \forall i, \forall j \tag{3.21}$$

$s_{i,j}$ is the score for the training data set instances x_i and x_j. Case (1) from equation 3.21 can be read as follows: The score $s_{i,j}$ is high if the labels y_i and y_j are equal, the nth local linear operator elements of both LLOs, $\lambda^{(i)}$ and $\lambda^{(j)}$ are similar, and the nth parameter sets $\omega_n^{(i)}$ and $\omega_n^{(j)}$ of both LLOs are similar. The other cases can be read in a similar fashion.

3.4 Score Design

This section discusses how a score that has the desired behavior described by the cases in equation 3.21 can be designed. Firstly, a high level concept for a NAS score is proposed. Then, components of the score are defined and related design choices are discussed systematically for each component. However, equation 3.21 only defines corner cases. Theoretical reasoning for the behavior of the score between the corner cases is difficult to provide. Therefore, some of the design choices in this section have to be made empirically.

The score will be computed in a pair-wise fashion for each two data points from a randomly sampled batch of training data and then aggregated over the batch. The reason to compute the score over a batch of training data is, that computing the score for the entire data set is expensive and similar NAS scores such as [14][15] or [16] achieved good results utilizing only a single batch of data. The influence of the batch size on the score will be examined during the experiments.

The four cases in equation 3.21 where the data points belong to the same class, $y_i = y_j$, have the negated desired score effect compared to the four cases where the data points belong to different classes, $y_i \neq y_j$:

- If $\lambda_n^{(i)} \approx \lambda_n^{(j)}$ and $\omega_n^{(i)} \approx \omega_n^{(j)}$;
  a **high** score is desired for $y_i = y_j$ and **low** score is desired for $y_i \neq y_j$.
- If $\lambda_n^{(i)} \approx \lambda_n^{(j)}$ and $\omega_n^{(i)} \not\approx \omega_n^{(j)}$;
  a **low** score is desired for $y_i = y_j$ and **high** score is desired for $y_i \neq y_j$.

- If $\lambda_n^{(i)} \not\approx \lambda_n^{(j)}$ and $\omega_n^{(i)} \approx \omega_n^{(j)}$;
  a **low** score is desired for $y_i = y_j$ and **high** score is desired for $y_i \neq y_j$.
- If $\lambda_n^{(i)} \not\approx \lambda_n^{(j)}$ and $\omega_n^{(i)} \not\approx \omega_n^{(j)}$;
  a **high** score is desired for $y_i = y_j$ and **low** score is desired for $y_i \neq y_j$.

Based on this observation, a score can be developed that covers the first four cases where the data points are from the same class, $y_i = y_j$, and then simply be negated for the remaining four cases where $y_i \neq y_j$.

The score depends on how similar or how distant the LLO elements and the parameter sets are. To quantify these inputs to the score a distance function

$$\delta_\lambda(\lambda_a, \lambda_b) \tag{3.22}$$

for the two LLO elements and another distance function

$$\delta_\omega(\omega_a, \omega_b) \tag{3.23}$$

for the parameter sets is introduced. Note that δ_λ operates on two scalars and δ_ω operates on two sets. The functions are defined in detail later.

For the first four cases the score is desired to be high if either (1) both LLO elements, $\lambda_n^{(i)}$ and $\lambda_n^{(j)}$, and both parameter sets, $\omega_n^{(i)}$ and $\omega_n^{(j)}$ are similar or (2) both are dissimilar and low otherwise. In terms of the above introduced distance functions one can express the same logic as follows: For the first four cases the score is desired to be high if the distance between the LLO elements, $\delta_\lambda(\lambda_n^{(i)}, \lambda_n^{(j)})$, and the distance between the parameter sets, $\delta_\omega(\omega_n^{(i)}, \omega_n^{(j)})$ are either both large or both small and low otherwise. In other words, the distances $\delta_\lambda(\lambda_n^{(i)}, \lambda_n^{(j)})$ and $\delta_\omega(\omega_n^{(i)}, \omega_n^{(j)})$ are desired to be similar. To quantify this distance a similarity function

$$\varphi(\delta_\lambda, \delta_\omega) \tag{3.24}$$

is introduced which returns a high value when its two arguments δ_λ and δ_ω are similar. The exact definition of the similarity function φ is discussed later.

Based on the two distance functions δ_λ and δ_ω, and the similarity function φ one can now compute the score $s_{i,j,n}$ for the nth element of the instances $x^{(i)}$ and $x^{(j)}$ from the batch of training data as

$$s_{i,j,n} = \varphi(\delta_\lambda(\lambda_n^{(i)}, \lambda_n^{(j)}), \delta_\omega(\omega_n^{(i)}, \omega_n^{(j)})), \tag{3.25}$$

where $\lambda^{(i)}$ and $\omega^{(i)}$ are the LLO and the vector of parameter sets, respectively, for the ith training data instance $x^{(i)}$.

Until here the score merely implements the first four cases of equation 3.21 where the two data points in questions belong to the same class, $y_i = y_j$. As mentioned before, to fulfill the remaining four cases the score for the first four cases has to be negated. A negation of the score can be achieved by multiplying the score with the coefficient

$$2I(y_i, y_j) - 1 \tag{3.26}$$

where I is the indicator function

$$I(y_i, y_j) = \begin{cases} 1 & \text{if } y_i = y_j \\ 0 & \text{Otherwise} \end{cases}. \tag{3.27}$$

The coefficient in equation 3.26 is 1 if the two data points $x^{(1)}$ and $x^{(2)}$ with the labels y_1 and y_2 are from the same class and -1 otherwise. The score $s_{i,j,n}$ for all eight cases in equation 3.21 can then be expressed as

$$s_{i,j,n} = (2I(y_i, y_j) - 1) \cdot \varphi(\delta_\lambda(\lambda_n^{(i)}, \lambda_n^{(j)}), \delta_\omega(\omega_n^{(i)}, \omega_n^{(j)})). \tag{3.28}$$

Given an architecture with an m-dimensional input one can aggregate the score for the input instances $x^{(1)}$ and $x^{(2)}$ as the mean of the score over all m elements of the LLOs and parameter sets.

$$s_{i,j} = \frac{2I(y_i, y_j) - 1}{m} \sum_{n=1}^{m} \varphi(\delta_\lambda(\lambda_n^{(i)}, \lambda_n^{(j)}), \delta_\omega(\omega_n^{(i)}, \omega_n^{(j)})). \tag{3.29}$$

This score can now be computed for all data points in the batch. One can arrange the score in a $m \times m$ matrix S where the element $s_{i,j}$ is the score for input instance $x^{(i)}$ and $x^{(j)}$ computed according to equation 3.29:

$$S = \begin{bmatrix} s_{1,1} & \dots & s_{1,m} \\ \vdots & \ddots & \vdots \\ s_{m,1} & \dots & s_{m,m} \end{bmatrix} \tag{3.30}$$

Note that in practice one does not need to compute the lower triangle of the matrix. The lower and the upper triangle of the matrix are identical since the score computation in equation 3.29 is commutative for two data points. Further, the main diagonal elements of this matrix are negligible as they compare a data point with itself.

The final step towards a score for the entire architecture is to aggregate the score over all pairs of data points in the batch. One way of aggregation is to take the mean of the upper triangle elements of S:

$$s = \frac{2}{(k-1) \cdot k \cdot m} \sum_{i=1}^{k} \sum_{j=i+1}^{k} (2I(y_i, y_j) - 1) \sum_{n=1}^{m} \varphi(\delta_\lambda(\lambda_n^{(i)}, \lambda_n^{(j)}), \delta_\omega(\omega_n^{(i)}, \omega_n^{(j)})) \tag{3.31}$$

Here the batch is assumed to have k instances. The normalization by the number of elements in the LLO, m, and the number of elements in the upper triangle of the matrix S, $(0.5 \cdot (k-1) \cdot k)$ is required to make the score robust against the input size and the batch size, respectively.

The entire score is built from the following components:

1. Distance measure δ_λ for the LLO elements
2. Distance measure δ_ω for the parameter sets

3. Similarity measure φ
4. Aggregation over the LLO
5. Aggregation over the Batch

Different choices for distance, similarity and aggregation functions for the five score components open up a space of different score designs. This work examines only a few instances from this space. A comprehensive analyses of further options for the above listed components is beyond the scope of this work. However, this thesis hypothesizes that the performance of the score is highly sensitive to these design choices. Therefore section 6.3 discusses how different options for the score components can be analysed in future work. The following paragraphs describe the design choices for the score components that are examined in this work:

For the distance measure δ_λ for the LLO elements four different metrics are examined:

1. The absolute difference between two LLO elements $\lambda_n^{(i)}$ and $\lambda_n^{(j)}$:

$$\delta_\lambda(\lambda_n^{(i)}, \lambda_n^{(j)}) = |\lambda_n^{(i)} - \lambda_n^{(j)}| \tag{3.32}$$

2. The absolute difference between two LLO elements $\lambda_n^{(i)}$ and $\lambda_n^{(j)}$ normalized by the square root of the product of the two LLO elements:

$$\delta_\lambda(\lambda_n^{(i)}, \lambda_n^{(j)}) = \frac{|\lambda_n^{(i)} - \lambda_n^{(j)}|}{\sqrt{|\lambda_n^{(i)} \cdot \lambda_n^{(j)}|}} \tag{3.33}$$

3. The squared difference between two LLO elements $\lambda_n^{(i)}$ and $\lambda_n^{(j)}$:

$$\delta_\lambda(\lambda_n^{(i)}, \lambda_n^{(j)}) = (\lambda_n^{(i)} - \lambda_n^{(j)})^2 \tag{3.34}$$

4. The squared difference between two LLO elements $\lambda_n^{(i)}$ and $\lambda_n^{(j)}$ normalized by the absolute value of the product of the two LLO elements:

$$\delta_\lambda(\lambda_n^{(i)}, \lambda_n^{(j)}) = \frac{(\lambda_n^{(i)} - \lambda_n^{(j)})^2}{|\lambda_n^{(i)} \cdot \lambda_n^{(j)}|} \tag{3.35}$$

For the distance function δ_ω between the two parameter sets $\omega_n^{(i)}$ and $\omega_n^{(j)}$ the *Jaccard distance* is examined in this work. The Jaccard distance is one minus the cardinality of the intersection of the sets normalized by the cardinality of the union of the sets, or just the cardinality of the symmetric difference $\triangle$ of the sets normalized by the cardinality of the union:

$$\delta_\omega(\omega_n^{(i)}, \omega_n^{(j)}) = 1 - \frac{|\omega_n^{(i)} \cap \omega_n^{(j)}|}{|\omega_n^{(i)} \cup \omega_n^{(j)}|} = \frac{|\omega_n^{(i)} \triangle\ \omega_n^{(j)}|}{|\omega_n^{(i)} \cup \omega_n^{(j)}|} \tag{3.36}$$

For the similarity function φ between the two distances $\delta_\lambda(\lambda_n^{(i)}, \lambda_n^{(j)})$ and $\delta_\omega(\omega_n^{(i)}, \omega_n^{(j)})$ two options are examined in this work:

1. The harmonic mean of the two distances $\delta_\lambda(\lambda_n^{(i)}, \lambda_n^{(j)})$ and $\delta_\omega(\omega_n^{(i)}, \omega_n^{(j)})$:

$$\varphi(\delta_\lambda(\lambda_n^{(i)}, \lambda_n^{(j)}), \delta_\omega(\omega_n^{(i)}, \omega_n^{(j)})) = \frac{2 \cdot \delta_\lambda(\lambda_n^{(i)}, \lambda_n^{(j)}) \cdot \delta_\omega(\omega_n^{(i)}, \omega_n^{(j)})}{\delta_\lambda(\lambda_n^{(i)}, \lambda_n^{(j)}) + \delta_\omega(\omega_n^{(i)}, \omega_n^{(j)})} \tag{3.37}$$

2. The correlation between the vector of LLO element distances $\delta_\lambda(\lambda^{(i)}, \lambda^{(j)})$ and the vector of parameter set distances $\delta_\omega(\omega^{(i)}, \omega^{(j)})$:

$$\varphi(\delta_\lambda(\lambda^{(i)}, \lambda^{(j)}), \delta_\omega(\omega^{(i)}, \omega^{(j)})) = corr(\delta_\lambda(\lambda^{(i)}, \lambda^{(j)}), \delta_\omega(\omega^{(i)}, \omega^{(j)})) \tag{3.38}$$

   where $corr(a, b)$ is the sample correlation coefficient of the two k-element vectors a and b:

$$corr(a, b) = \frac{\sum_{n=1}^{k} (a_n - \bar{a}) \cdot \left(b_n - \bar{b}\right)}{\sqrt{\sum_{n=1}^{k} (a_n - \bar{a})^2 \cdot \sum_{n=1}^{k} \left(b_n - \bar{b}\right)^2}} \tag{3.39}$$

   $\bar{a}$ and $\bar{b}$ denotes the sample mean of the vectors a and b, respectively. Note that the correlation is not a similarity function. It does however still represent the score objective described in section 3.3. Note also that the correlation combines the third and fourth score component, namely the similarity function and the aggregation over the LLO, as it yields a scalar for the entire LLO.

The aggregation over the LLO and over the batch is in this work computed as the mean over all elements in the LLO and all data point pairs in the batch and normalized by the number of LLO elements times the number of pairs in the batch as shown in equation 3.31.

The next section describes the set-up for the experiments that are conducted in this work with the above described score.

3.5 Experiment Set-Up

The implementation of the proposed score that is used for the experiments is not optimized for speed or memory consumption. Such optimizations are beyond the scope of this work. It is therefore infeasible to compute the score for a large number of big architectures in order to evaluate the correlation of the score with the validation accuracy of the architectures[2]. Instead the experiments in this work aim to provide insights into the discrimination potential of the score with respect to the goodness of the candidate architectures. This section describes the set-up for these experiments.

Due to the large memory consumption of the parameter sets only small architectures can be scored with the current, experimental implementation of the score. For the experiments conducted in this work the score is evaluated on a number of

[2] 1000 is a common order of magnitude in the domain for the number of architectures to validate the score on [14][15][16].

small architectures from the size search space of the NATS bench. Since difference between the different architectures in the size search space of the NATS bench is the number of channels, the architectures are in the following identified by a tuple with five integer elements referring to the number of channels in the five cells of the architecture, including the two reduction cells of the architectures. E.g. the tuple $(8, 8, 8, 8, 16)$ identifies the architecture that has eight channels in all cells except for the last cell where it has 16 channels. Table 3.1 lists the architectures that are examined in this work together with their validation accuracy on CIFAR10.

Table 3.1: Architectures from the SSS of the NATS Bench identified by number of channels in their five cells, together with their validation accuracy on CIFAR10.

Channels	**CIFAR10 Val Acc**
$(8, 16, 8, 16, 8)$	82.35 %
$(8, 8, 16, 8, 16)$	84.61 %
$(16, 16, 8, 16, 16)$	85.83 %
$(16, 8, 16, 16, 16)$	87.23 %

The architectures where picked so that each the highest and the lowest performing architecture from a certain size is represented. Architecture $(8, 16, 8, 16, 8)$ and architecture $(8, 8, 16, 8, 16)$ are the lowest and highest performing architectures with three times eight and two times 16 channels. Architecture $(16, 16, 8, 16, 16)$ and architecture $(16, 8, 16, 16, 16)$ are the lowest and highest performing architectures with one times eight and four times 16 channels.

The experiments are conducted as follows: the architectures are queried from the NATS bench. A batch of 32 images[3] from the training split of the CIFAR10 data set is sampled randomly. The batch of images is the same for all architectures. The parameter sets and LLOs for the architectures for all images in the batch are created. The above discussed score is computed based on the parameter sets and LLOs for all combinations of distance functions and similarity measures.

The following analyses are conducted on the results and intermediate results of the above described experiments:

1. The parameter set distances of
   (a) a large architecture with high performance,
   (b) a large architecture with low performance,
   (c) a small architecture with high performance and,
   (d) a small architecture with low performance

   are compared, to identify characteristics of the parameter set differences for architectures of different size and performance.

2. The parameter set distances of

[3]The small batch size compared to other scores like [14], [15], and [16] is also motivated by the high memory consumption of the parameter sets.

   (a) an architecture with high performance between images of the same class,
   (b) an architecture with high performance between images of different classes,
   (c) an architecture with low performance between images of the same class and,
   (d) an architecture with low performance between images of different classes

   are compared, to analyse the impact of the data point class on the parameter set differences for different architecture performances.

3. The LLO differences and parameter set differences are compared for
   (a) an architecture with high performance, and
   (b) an architecture with low performance.

4. The correlation between the validation accuracy and the score over the architectures is computed.

The chapter 5 presents the results of the above described experiments.

4

Implementation

The inputs to the score proposed in this thesis are the LLOs $\lambda^{(i)}$ and sets of the network's parameters $\omega^{(i)}$ that the elements of the LLOs are composed of. The LLOs can be obtained as the gradient of the network's output with respect to its input. Most deep learning libraries implement functionalities to compute such gradient through the models [22][21]. The LLOs can therefore be computed in a straight forward way when implementing the score in practice. The second input to the score, the parameter sets ω, are however not trivial to compute. This chapter is dedicated to the implementation of the parameter set computation. The implementation of the equations to compute the score given $\lambda^{(i)}$ and $\omega^{(i)}$ discussed in section 3.4 is trivial and not documented here. The chapter is structured in two sections: section 4.1 shows how the parameter sets can be computed in a back-propagation fashion and section 4.2 describes a network representation that is found to be convenient when computing the parameter sets.

4.1 Parameter Back-Propagation

A parameter set $\omega_n^{(i)}$ contains all parameters that determine the influence of the LLO element $\lambda_n^{(i)}$ on the networks output. Intuitively, one can think about the parameter set as the set of the parameters that lie on any path through the network that connects the input with the output. Figure 4.1 visualizes this concept using the one-dimensional convolution network from section 3.2 as an example.

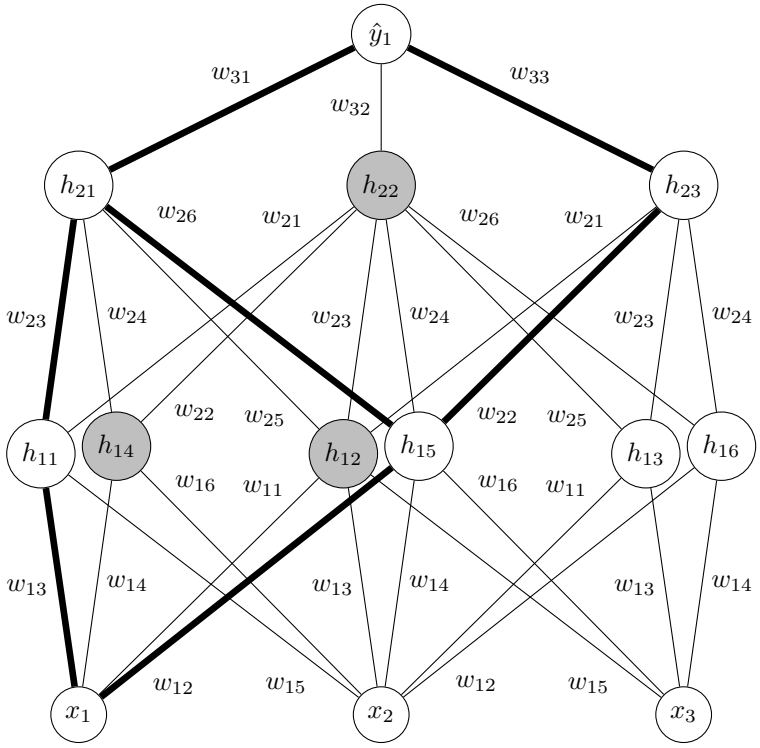

Figure 4.1: The one-dimensional convolution network from section 3.2. Assuming that the ith data point $x^{(i)}$ from a batch gives rise to an activation pattern where the neurons h_{12}, h_{14} and h_{22} are dead (grey) and all other neurons are alive (white) the first element of the input $x_1^{(i)}$ influences the output $\hat{y}_1$ via the paths highlighted in bold. The parameter set $\omega_1^{(i)}$ that corresponds to input element $x_1^{(i)}$ contains the parameters from these paths: $\omega_1^{(i)} = \{w_{12}, w_{13}, w_{22}, w_{23}, w_{26}, w_{31}, w_{33}, \}$. Note that a path does not contribute to the parameter set if it contains one or more dead neurons.

Evaluating all paths from an input to the output for a large network becomes a combinatorial explosion since the number of paths grows exponentially with the number of neurons in the network. It is more efficient to modularize the computation *layer-wise*: Starting from the last layer of the network the parameter sets are computed backwards layer-by-layer for each neuron in the network until the input is reached. A parameter set has the same semantics for hidden units as it has for inputs of the network; it contains all parameters by which the hidden unit influences the output. For each layer the parameter sets of the layer's neurons are redistributed to the previous[1] layer's neurons. The parameter sets of the neurons in the previous layer are composed by the parameter sets of all deeper neurons that are by an edge connected to the neuron and supplemented by the weights of all connecting edges. The parameter set of a neuron is empty if a neuron is dead, since it does not influence the output. If an active neuron is connected to a dead neuron the weight of the connecting edge does not contribute to the parameter set of the active neuron. The next section provides an example for how the parameter sets for the inputs of a network can be obtained by computing the parameter sets for the hidden neurons

[1] *Previous* refers to the order of the layers in the forward pass.

layer-by-layer. The example uses the same network as shown in figure 4.1.

4.1.1 Parameter Back-Propagation - An Example

Figure 4.2 shows the parameter sets of the last hidden layer of the one-dimensional convolution network. The parameter set ω_{y1} of the output neuron $\hat{y}_1$ is empty since it is the output of the network itself and there are consequently no parameters in-between the neuron and the output. The parameter sets ω_{h21} and ω_{h22} are the ω sets of the hidden units h_{21} and h_{22}, respectively. They only contain the weight of the connecting edge since the set ω_{y1} is empty. The parameter set ω_{h22} of neuron h_{22} is empty since the neuron is dead. Note that the biases of the neurons are omitted here for simplicity.

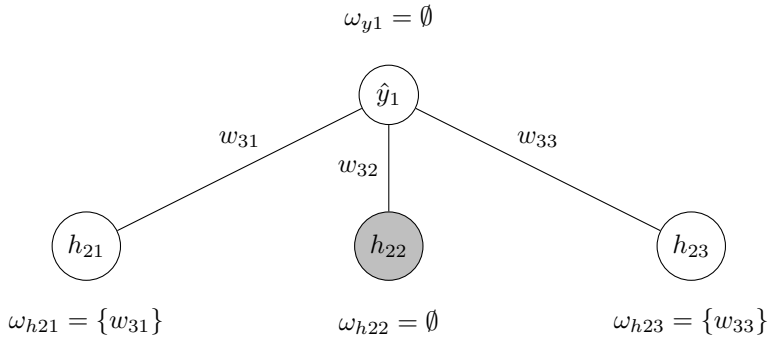

Figure 4.2: The output layer of the one-dimensional convolution network from section 3.2 together with the parameter sets of the neurons. A grey neuron is dead and a white neuron is active.

After all parameter sets for the output layer have been computed, the parameter sets for the previous layer can be computed. Figure 4.3 shows how the parameter sets that the output layer mapped to the second hidden layer in figure 4.2 are in a similar fashion mapped by the second hidden layer to neurons of the first hidden layer.

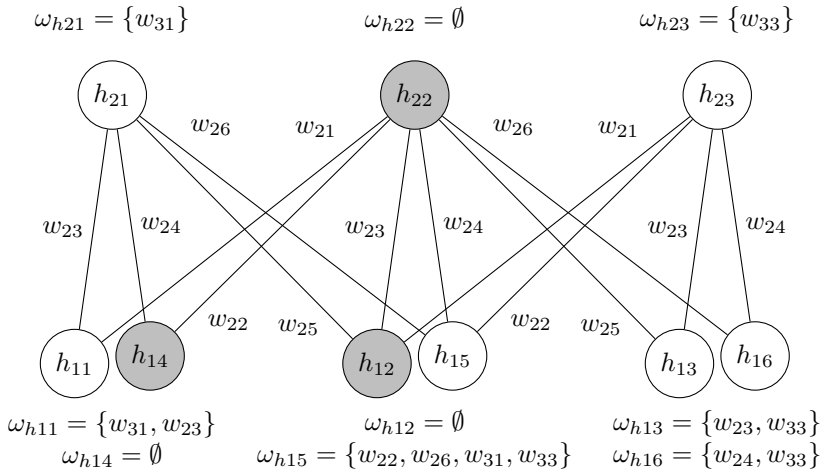

Figure 4.3: The second hidden layer of the one-dimensional convolution network from section 3.2 together with the parameter sets of the neurons. A grey neuron is dead and a white neuron is active.

The parameter sets ω_{1x} from the neurons in the first hidden layer each consist of the union of the parameter sets of the neurons in the second hidden layer ω_{2x} to which they are connected and the weights of the connecting edges. Figure 4.4 shows how the parameter sets that the second hidden layer mapped to the first hidden layer in figure 4.3 are in a similar fashion mapped by the first hidden layer to input neurons.

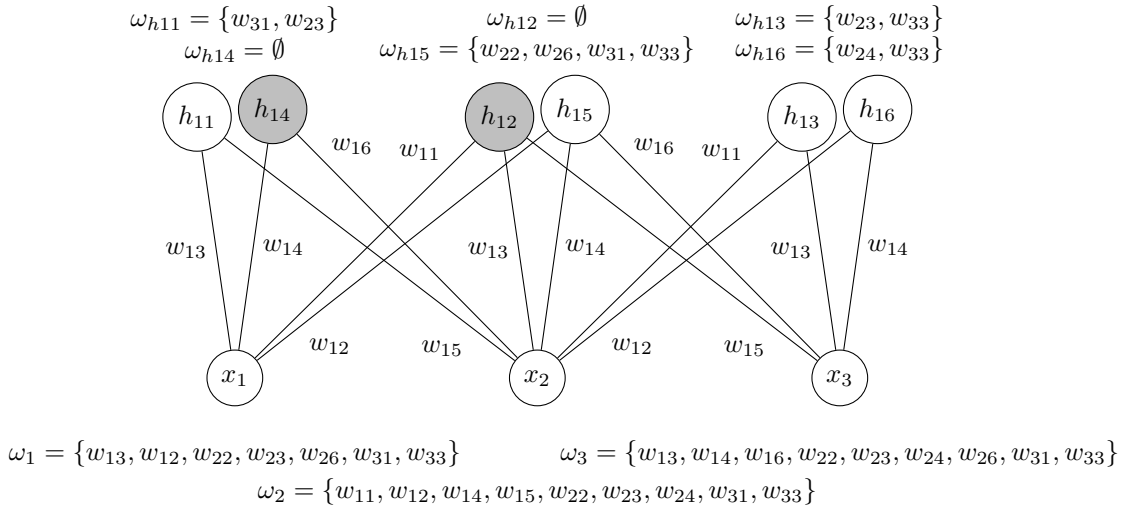

Figure 4.4: The first hidden layer of the one-dimensional convolution network from section 3.2 together with the parameter sets of the neurons. A grey neuron is dead and a white neuron is active.

The resulting parameter sets ω_1, ω_2, and ω_3 are the final parameter sets of the network. Note that the parameter set ω_1 from the layer-by-layer back-propagation of the parameters is identical with the parameter set ω_1 that was obtained by an exhaustive evaluation of all paths through the network in figure 4.1.

The example shows how the parameter sets can be back-propagated efficiently through the network[2] in a layer-by-layer fashion. The method that this thesis proposes to compute the parameter sets utilizes this principle. The main functionality that needs to be implemented to realize the back-propagation of the parameter sets is the backwards mapping of the parameter sets for all different layer types in the network. The layer type determines how the neurons of the layer are connected to the ones of the previous layer and consequently how the parameter sets are mapped. Therefore the logic of the backwards mapping of the parameter set is layer type specific. The next section lists the different layer types relevant for the experiments in this thesis and explains the backwards mapping of the parameter sets for each layer type.

4.1.2 Parameter Set Mapping of Different Layer Types

This section describes the parameter set mapping for different layer types but also for operations that are not commonly referred to as a *layer*, as e.g. the *addition*

[2]The network is assumed to be an acyclic graph of neurons and connecting, weighted edges.

operation. During the following the term *operation* refers to all layers but also to all other operations in a neural network. The experiments in this thesis are conducted on architectures from the NATS bench. Even though, the implementation of the score proposed in this thesis, scales to other architectures, it is in this work limited to operations that are used in the architectures from the NATS bench. These operations are the following:

- Linear
- 2D Convolution
- 2D Average Pooling
- Zeroize
- Identity
- ReLU
- Batch Norm
- Addition

This work proposes one procedure for the mapping of the parameter set for each operation type. The following paragraphs explain the parameter set mapping for the above listed operations:

The *linear* operation, refers to a *fully-connected* layer in a neural network. When mapping the parameter sets from the neurons of a linear layer backwards to the neurons of the previous layer each neuron h_n in the previous layer gets mapped three components:

1. The union of all parameter sets from the neurons of the linear layer
2. The set of the weights from all edges that connect the neuron h_n with an active neuron of the linear layer
3. The set of the biases of the active neurons of the linear layer

The mapping for the *convolution* operation is the most elaborate among the operations considered here. The parameter sets of a convolution operation are arranged in a three dimensional tensor of the same shape as the output feature map[3] of the operation. The tensor is mapped to a tensor with the shape of the input feature map of the operation. Each element h_n of the resulting tensor with the shape of the input feature map is the union of the following three components:

1. The union of all sets in the parameter set tensor of the layer within the receptive field of one filter around the position of the element h_n.
2. The set of weights of all filters, at the same channel as the element h_n. A weight is not included if the corresponding set in the parameter set tensor of the layer is empty.

[3]The notions *Input* and *output* feature map refer to the input and the output of an operation in the forward pass. In the parameter set back-propagation process the parameter sets are propagated backwards through the network. Therefore the mapping is conducted from the output to the input feature maps of the operations.

3. The set of all biases except for the ones where all elements within the receptive field of one filter around the position of the element h_n are empty at the referring channel in the parameter set tensor of the layer.

The receptive field of a filter of a convolution layer that is aligned at a certain position in a feature map describes all elements of the feature map that contribute to the result of the current alignment of the filter. Figure 4.5 provides an example of two different receptive fields for different hyperparameter settings of the convolution layer. For the above described parameter mapping the concept of the receptive field is applied to the parameter set tensor of the layer instead of to feature map of the layer.

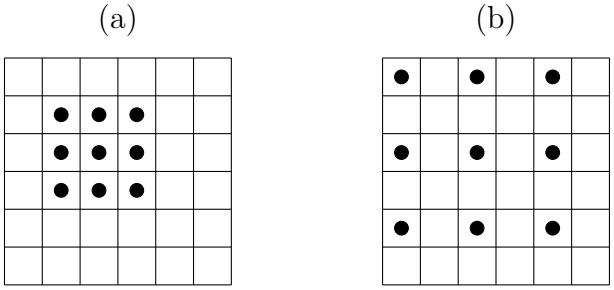

Figure 4.5: The receptive field of (a) a 3×3 filter with a dilation of 1 and (b) a 3×3 filter with a dilation of 3. The filters in (a) and (b) have both been aligned at the third row and the third column of a 6×6 feature map with a single channel. Each square is one element of the feature map. A black dot indicates that an element belongs to the receptive field.

The *average pooling* operation does not have own parameters. Therefore the mapping of the parameter sets can be described by a single component: each element h_n in the yielded tensor is the union of all sets in the parameter set tensor of the layer within the receptive field of one filter around the position of the element h_n at the same channel as the element h_n.

For the *ReLU* operation the parameter sets are mapped backwards as follows: all sets where the corresponding element in the input feature map of the ReLU operation is positive are passed on as they are. All other sets are replaced by an empty set in the resulting tensor.

The parameter set mapping for the *zeroize* operation is conducted by replacing all parameter sets of the operation each by an empty set.

The *identity* operation and the *batch norm* operation do not apply any modifications to the parameter sets. The parameter sets are passed to the previous layer as they are. Even though the batch norm operation strictly speaking has some trainable parameters, they are not included in the parameter sets as they would contribute to all parameter sets in the same way and thus would not increase the discrimination potential of the parameter sets.

The *addition* operation is the only operation among the considered ones that takes

two tensors as the input in the forward pass. Therefore the operation has two predecessor operations. The mapping of the parameter sets for the addition operation is conducted by mapping one copy of the unmodified parameter sets of the operation to each of the two predecessor operations.

4.2 Graph Representation

This section introduces a format to represent a neural network in software that is found convenient when computing the parameter sets. Transforming the networks into this format is a prerequisite for the method of the parameter set computation that is presented below.

Each operation in the network is represented as a *node* object. Each node object has four attributes:

1. A type identifier
2. A collection of hyperparameters
3. A collection of predecessor nodes
4. A tensor of parameter sets

In case of a ReLU operation the node has the feature map of the ReLU as a fifth attribute. All attributes are needed during the parameter back-propagation: The type identifier determines which of the above explained mapping mechanisms is applied to the parameter sets of the node. The collection of hyperparameters is required by the mapping mechanism. For e.g. a convolution operation, the collection of hyperparameters to determine the receptive field of a filter alignment would comprise the filter size and the dilation. The collection of predecessor nodes contains all nodes of which the output is an input to the node in question. The mapped parameter sets of a node are written to all nodes in the collection of predecessor nodes. The tensor of parameter sets of a node is empty until another node writes its mapped parameter sets into it.

To construct the above described node objects one needs to know the graph structure of the Network. The NATS bench which is utilized for the experiments in this thesis provides the architectures as *Pytorch*[21] models. Pytorch uses *dynamic computational graphs* to represent neural networks. *Dynamic computational graphs* are only constructed when they are executed. Their properties and structure can depend on the input. The graph structure that is required to construct the above described nodes for the operations in the network is therefore not accessible from the plain pytorch representation of a network. This thesis utilizes the *Pytorch* module *FX* to obtain the information of how the different nodes, corresponding to the different operations in the network, are connected. FX implements a *symbolic tracer*. The tracer executes the network on a symbolic input and records the graph structure while it is executed. For this thesis the networks of the NATS bench are traced with FX. The resulting graph information is then embedded in the above described node objects. A whole network is then represented as a ordered collection of node

objects. The node objects are sorted in a way that a node can only be appended to the collection if all its predecessor nodes are already members of the collection. Table 4.1 shows how the described graph representation would look for the example cell from a NATS bench from figure 2.6.

Table 4.1: Graph representation used in this thesis for the cell from a NATS bench shown in figure 2.6. The table shows the node index together with the node attributes. The attribute *parameter sets* is omitted here since it is empty for all nodes until the parameter sets are back-propagated.

Node	**Type**	**Hyperparameters**	**Predecessors**
1	conv2D	[kernel size: 3x3,...]	[Input]
2	zeroize	[ ]	[Input]
3	identity	[ ]	[Input]
4	conv2D	[kernel size: 1x1,...]	[1]
5	conv2D	[kernel size: 1x1,...]	[1]
6	add	[ ]	[2,4]
7	avgPool	[kernel size: 3x3,...]	[6]
8	add	[ ]	[5,3]
9	add	[ ]	[7,8]

Based on a network, represented in the presented graph format the parameters sets can be mapped through the network in a layer by layer fashion by iteration over the following steps for each node in the node ordered collection of nodes:

1. Map the parameter sets of the node backwards based on the hyperparameters utilizing the mapping mechanism for the type of the node.
2. Write the mapped parameter sets to the nodes in the collection of predecessor nodes of the node.

Note that the ordered collection of nodes has to be traversed in reversed order to propagate the parameter sets backwards from the output layer.

The next chapter shows the results based on the implementation presented in this chapter.

5

Results

This chapter presents the results of the experiments described in section 3.5. Firstly, intermediate results from the score computation are shown. Then the final score is presented for all assessed architectures and all examined score component combinations. All results are presented together with observations that can be made when viewing the results. Section 5.1 compares the parameter set distances for small and big as well as for high and low performing architectures. Section 5.2 shows the parameter set differences for image pairs from the same and from different classes. Section 5.3 compares local linear operator differences to the parameter set differences on the same image. The score of the different architectures for different score components is presented in section 5.4.

The local linear operator distances and the parameter set distances are visualized as heat maps in the subsequent sections. The following holds for all heat maps: each heat map shows examples of differences for the first channel of the input image, referring to the color red in the images. A single channel is picked to simplify the visualisation. The architecture that the visualized differences stems from is indicated in the title of each heat map by the channel-tuple of the architecture. The title of a heat map further contains the validation accuracy of the architecture denoted by *acc* as well as the mean and standard deviation of the plotted distances denoted by μ and σ, respectively. As is the case for the images of the CIFAR10 data set, the heat maps have spatial dimensions of 32×32. The heat maps are visualizing the differences for one example image each. Further examples are included in the appendix in order to make the findings more representative. The distance metric that was utilized to compute the visualized parameter set differences is the Jaccard distance. The distance metric that was utilized to compute the visualized LLO element differences is the absolute difference.

The distances in the subsequent sections are computed for different input images. Some images are referred to as *hard* and is referred to as *easy*. An image is referred to as *hard* if it is difficult to classify and as *easy* if it is easy to classify for a neural network[1].

[1]The methodology applied in this work to determine whether it is hard or easy to classify an image is as follows: A VGG-13 architecture is trained on the CIFAR10 training data set. The images in the batch that is utilized for the score computation in this work are classified by the VGG-13 architecture. The images with the lowest and highest confidence for the correct class are the *hard* and *easy* images, respectively

5.1 Parameter Set Differences of Big vs. Small Architectures

Figure 5.1 and figure 5.2 show the heat maps of the parameter set differences for large and small architectures with high and low performance. Figure 5.1 shows the heat maps resulting from a hard images and figure 5.2 shows the heat map resulting from easy images.

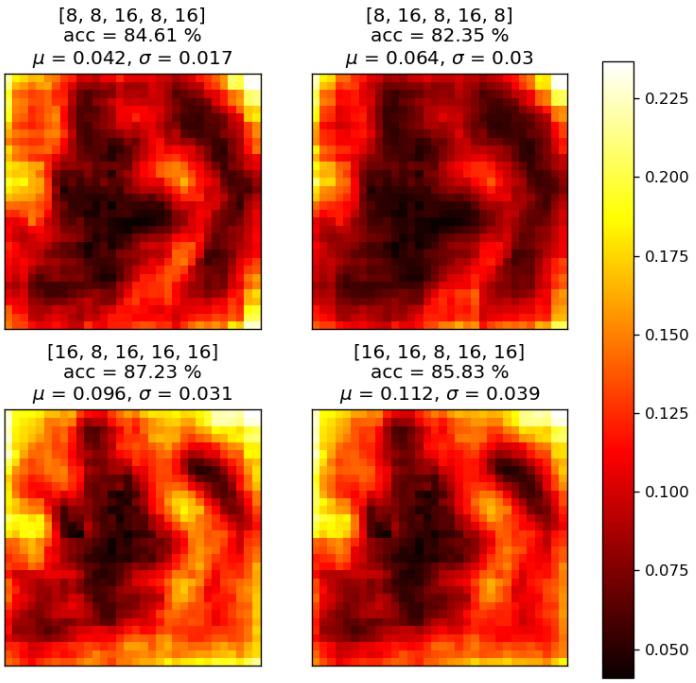

Figure 5.1: A heat map of the parameter set differences for the two hard images x_{10} and x_{18} for the small architecture (8,8,16,8,16) with high performance, the small architecture (8,16,8,16) with low performance, the large architecture (16,8,16,16,16) with high performance, and the large architecture (16,16,8,16,16) with low performance.

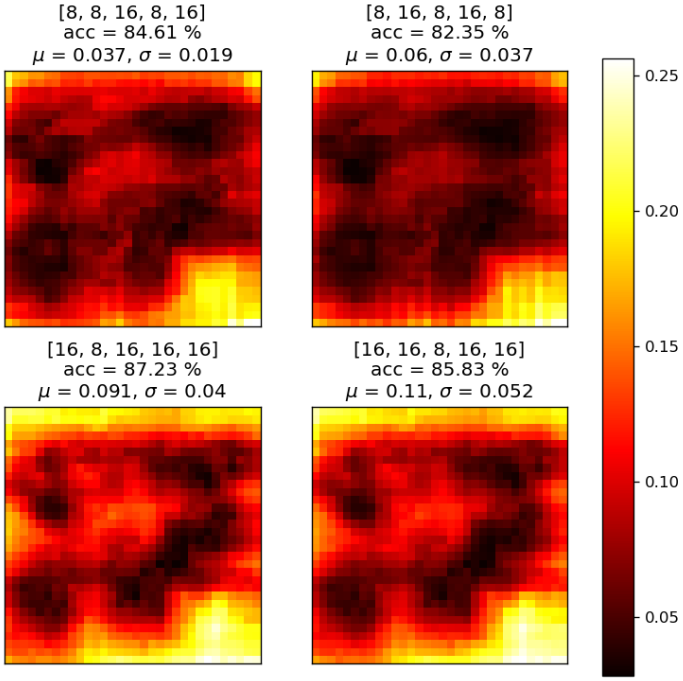

Figure 5.2: A heat map of the parameter set differences for the two easy images x_4 and x_{11} for the small architecture (8,8,16,8,16) with high performance, the small architecture (8,16,8,16) with low performance, the large architecture (16,8,16,16,16) with high performance, and the large architecture (16,16,8,16,16) with low performance.

Further examples of the parameter set differences for small and large architectures with high and low performance are presented in appendix A.2.

The following observations can be stated based on figure 5.1 and figure 5.2: The larger architecture has on average the higher Jaccard distances. The distances in the middle of the image are generally lower than at the edges of the image. The heat maps for architectures with lower performance seem to show more *blurry* patterns than the ones from architectures with higher performance. The latter motivates to asses the distribution of the parameter set distances over the magnitude of the distances. Figure 5.3 shows the histogram of parameter set distances for all four evaluated architectures.

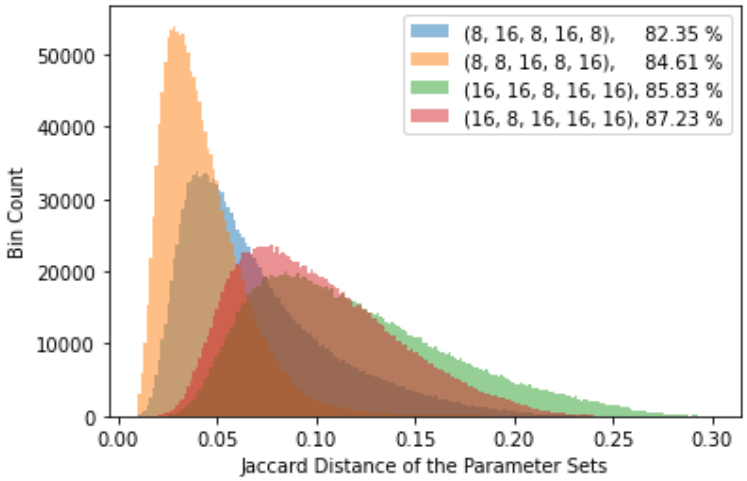

Figure 5.3: Histograms of the Jaccard distances of the parameter sets of all image pairs in the utilized batch for the four examined architectures.

The distributions in figure 5.3 are distinct for the four architectures. Pairing the architectures so that each the two architectures with the same size form a pair, it holds for both pairs that the higher performing architecture in a pair has the more concentrated distribution.

5.2 Inter-Class Vs. Within-Class Parameter Set Differences

Figure 5.4 shows the heat maps of the parameter set differences for images from the same as well as for images from different classes each for a high and a low performing architecture. Both architectures are of the same size.

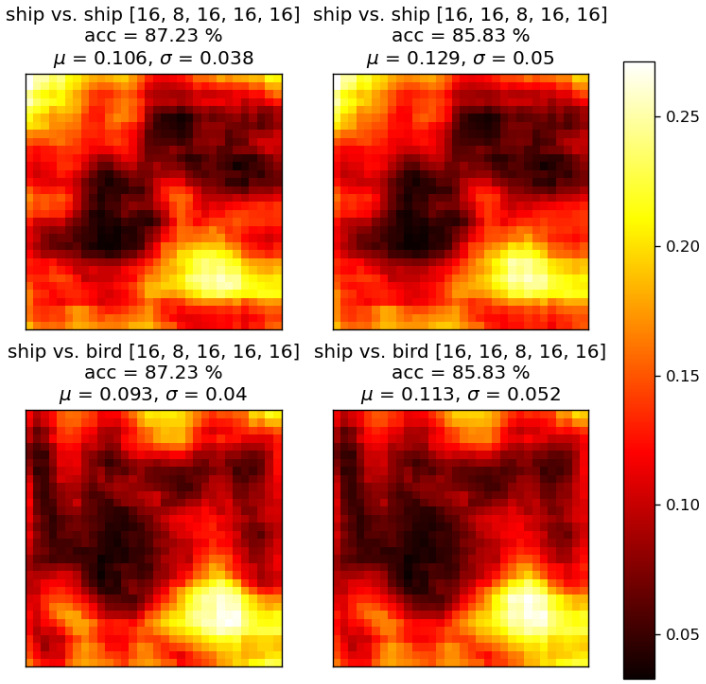

Figure 5.4: Heat maps of the parameter set differences for the architecture (16,8,16,16,16) with high performance, and the architecture (16,16,8,16,16) with low performance, each once on the two images x_0 and x_2 from the same class, *ship*, and on the two images x_0 and x_5 from different classes; *ship* and *bird*, respectively. Further examples of inter-class and within-class parameter set differences are presented in appendix A.3.

In the heat maps of the parameter set differences alone no characteristics can be identified that would indicate whether the two underlying images belong to the same or to different classes.

5.3 Parameter Set Differences Vs. Local Linear Operator Differences

Figure 5.5 and figure 5.6 show the parameter set distances and the LLO distances of the same image pair next to each other. The images in the pair plotted in figure 5.5 stems from the same class while the images in the pair from figure 5.6 stem from different classes.

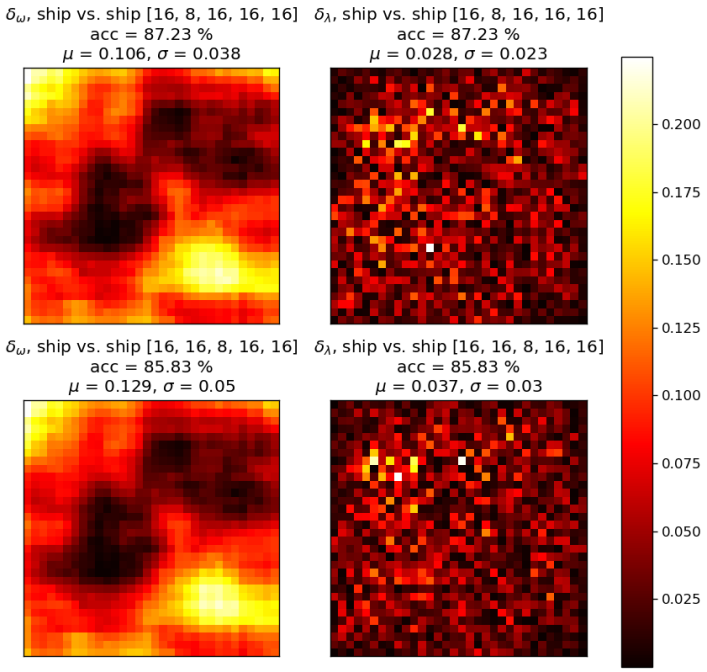

Figure 5.5: Heat maps of the parameter set differences δ_ω and the LLO differences δ_λ, each for the architecture (16,8,16,16,16) with high performance, and the architecture (16,16,8,16,16) with low performance. The architectures are executed on the two images x_0 and x_2 from the same class, *ship*

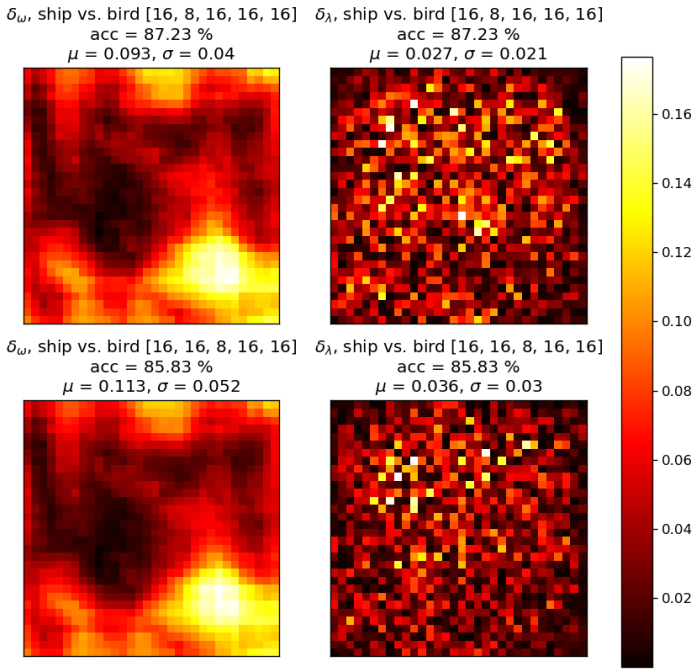

Figure 5.6: Heat maps of the parameter set differences δ_ω and the LLO differences δ_λ, each for the architecture (16,8,16,16,16) with high performance, and the architecture (16,16,8,16,16) with low performance. The architectures are executed on the two images x_0 and x_5 from the classes *ship* and *bird*, respectively.

Further examples of parameter set differences compared to the referring LLO differences are presented in appendix A.4.

It can be observed in the above shown plots that, regardless of whether the two underlying images stem from the same class or from different classes, the parameter set distances and the LLO distances show a slight negative correlation; the brightest areas in the heat map of the LLO distances are the darkest areas in the heat map of the parameter distances.

5.4 Correlation of Score and Validation Accuracy

Table 5.1: The score of four architectures for all examined combinations of score components. The score is computed over the same randomly sampled batch of input images from the training split of the CIFAR10 data set for all architectures. The column *channels* identifies the architectures by their channel-tuple. The column *acc* states the validation accuracy of the architectures on the CIFAR10 data set, reported in the NAS bench. *abs* and *sq* indicate that the score was computed based on the absolute and squared LLO distances, respectively. *norm* indicates that the LLO distances are normalized. For details regarding the score components see section 3.4. The row R states the Pearson correlation coefficient of the score and the validation accuracy of the architectures.

		harmonic mean				**correlation**			
architecture				**norm**				**norm**	
channels	**acc**	**abs**	**sq**	**abs**	**sq**	**abs**	**sq**	**abs**	**sq**
$(8, 16, 8, 16, 8)$	82.35 %	-0.0251	-0.0020	-0.1067	-0.1037	0.1472	0.1154	-0.0040	-0.0044
$(8, 8, 16, 8, 16)$	84.61 %	-0.0276	-0.0042	-0.0681	-0.0659	0.1136	0.8709	0.0001	0.0028
$(16, 16, 8, 16, 16)$	85.83 %	-0.0419	-0.0046	-0.1758	-0.1715	0.1275	0.0988	-0.0046	-0.0035
$(16, 8, 16, 16, 16)$	87.23 %	-0.03232	-0.0026	-0.1497	-0.1455	0.1164	0.0896	0.0003	0.0010
R	-	-0.6302	-0.3445	-0.5701	-0.5685	-0.7549	-0.7536	0.4624	0.4583

The following observations can be stated based on table 5.1: All scores that utilize the harmonic mean as a similarity function have negative values with a magnitude below 0.2. The scores that utilize the correlation as the similarity function are positive and lie between approximately 0.1 and 0.9 for the not-normalized LLO distances and around zero for the normalized distances. The correlation R of the harmonic-mean-based score and the validation accuracy is negative in all cases and lies between approximately -0.7 and -0.3. The correlation R of the correlation-based score and the validation accuracy is approximately -0.75 for both not-normalized LLO distances and approximately 0.45 for both normalized LLO distances. The highest correlation R of 0.4624 achieves the score which utilizes the correlation as a similarity function and the normed absolute difference as the distance function for the LLO elements.

5.5 Discussion

This section discusses the results which are presented above. The number of architectures and the number of images examined in this work is very limited. The discussion is based on these limited results. The thesis does not claim that the findings scale to other architectures, however. The main contribution of this work is the method to analyse LLOs in terms of their parameter dependencies. The results presented above show examples of how this method can be utilized for analyses and in a NAS score.

It is found that the distribution of the parameter set distances given the architecture size seems to depend on the performance of the architecture. This is a key finding since it indicates that the parameter sets are an appropriate means to assess the goodness of an architecture before training. Further, the dependency that was

found between the distribution of the parameter set differences and the architecture performance motivates a future research direction: further research could be directed towards finding an appropriate dispersion measure to describe the difference in the parameter set distributions as a basis for a NAS score. The experiments show that this would likely require a normalization mechanism that eliminates the influence of the architecture size on the score.

To guide the choices of the examined score components the following can be stated: The objective for the score is to achieve a high[2] correlation R with the validation accuracy of an architecture. Six of eight score variants in table 5.1 yield a negative R. Only the two scores that utilize the correlation as the similarity function and the normalized differences for the distances of the LLOs yield a positive R. This supports the hypothesis that the score is highly sensitive to the involved design choices.

Based on table 5.1 the correlation is, compared to the harmonic mean, the better choice for the similarity function of the score, as it achieves on average the higher correlation with the validation accuracy of the architectures. The correlation-based score seems to be sensitive to the normalization of the LLO distances; the correlation-based score achieves the worst R among all scores if the LLO distances are not normalized and the best R if they are normalized.

The choice between the absolute and the squared distances seems not to have a significant influence on the correlation R with the validation accuracy of the architectures. The best performance based on the correlation R between score and validation accuracy of the architectures is achieved by the score which utilizes the correlation as a similarity function and the normalized absolute difference as the distance function for the LLO elements.

Beyond directly guiding the score design the results show that analyses of the parameter set differences can reveal remarkable insights which are potentially useful for neural architecture design in general: all assessed heat maps indicate larger parameter set distances at the edges of the input image compared to the center of the input image. That means it is more difficult for a network to distinguish between images of which the decisive features are located in the image center compared to images with the decisive features being located at the edges of the image. That is remarkable since the objects on the images in many data sets as e.g. CIFAR10 are located approximately in the center of the image.

[2]Note that a score with a large negative correlation with the validation accuracy also serves to identify high performing architectures. This would however contradict the theory which the score proposed in this work is based on.

6
Conclusion

Motivated by the computational cost of conventional NAS methods this thesis proposes a novel proxy score for the validation accuracy of neural architectures which can be computed before the architecture is trained. Previous work shows that the best performing training-free NAS scores to date are based on the local linear operators of an architecture. The proposed score is also based on the local linear operators of the architectures and aims to improve over previous approaches in two aspects: firstly, it assesses the pairwise dependency of the local linear operators instead of only the local linear operators directly. The pairwise dependency of the local linear operators indicates how dependently two local linear operators can be adjusted during the training. The dependency of the local linear operators is hypothesized to preserve the properties of the network better through the training then the local linear operators themselves. Secondly, the score reflects in its objective that a good architecture should be able to obtain both, a high within-class similarity and a high inter-class distance in the mappings of the data points. Previous approaches reflect at most one of these two aspects in the score objective.

The thesis proposes to quantify the pairwise dependency of the local linear operators as the set distances between the *parameter sets* of the local linear operator elements. The *parameter set* of a local linear operator element contains all parameters of the network that the local linear operator element is composed of.

The thesis proposes a method to obtain the parameter sets of the local linear operator elements in practice. This method is considered to be valuable beyond the application in a NAS score since it allows an in-depth analyses of the parameter contributions of different layers in a network. The contributions of this work are stated below.

6.1 Contributions

Local linear operators are an important direction in training-free neural architecture search. The work proposes a novel method to quantify the pairwise dependency of local linear operators of neural networks with ReLU activation.

The work describes a procedure that implements the above mentioned method to obtain the dependency of LLOs. By this means the work contributes a powerful tool for the analysis of local linear operators to the domain.

The thesis provides examples of how to utilize the analyses of pairwise dependencies of local linear operators to gain remarkable insights into the mechanics of neural networks. These insights are considered to be valuable for NAS and beyond. E.g for the domain of *Explainable AI*.

The work proposes to base a NAS score for training-free NAS on the pairwise dependency of the local linear operators and introduces a class of score designs that implement this idea.

The work proposes to reflect both, a high within-class similarity and a high inter-class difference for the mapping of data points in the score objective and incorporates this idea in the objective of the proposed class of NAS scores.

6.2 Limitations

This section describes the limitations of the results in this thesis. The thesis proposes a score design for a NAS score that can be computed for architectures before the training. The thesis cannot provide strong evidence for or against the hypothesis that the score computed for an architecture prior to training is correlated with the validation accuracy of the architecture after the training. It cannot be stated if the score is on par with or even outperforms existing approaches.

The conducted experiments are based on a small selection of architectures. These architectures cover only a fraction of the size search space of the NATS bench and the NATS bench covers only a fraction of the search space of architectures that would be feasible with today's computational resources. The score is computed for architectures on the data set CIFAR10. CIFAR10 poses a, by today's standards, simple image classification task. The score is not evaluated in other image recognition disciplines than image classification.

Due to the above limitations the thesis cannot claim that the presented findings scale to other architectures, other data sets, or other NAS search spaces.

A fundamental part of the contributions of this work is a method to quantify the pairwise dependency of local linear operators of neural networks. The current implementation of the method is in an experimental state and not yet optimized for speed and memory efficiency. This limits the number and extent of the experiments that can be conducted with the method within a feasible computation budget.

6.3 Further Work

This section states suggestions on how to develop the ideas presented in this thesis further. The section starts with improvement suggestions of the current score design on a detailed level and moves on to formulating high-level visions for the domain of

NAS without training.

Developing a theory to motivate design choices for the score design is difficult. It is easier to motivate these design choices empirically. Therefore more extensive experiments need to be conducted. The score needs to be developed and evaluated based on more architectures, more underlying image classification data sets, more images per batch and on different NAS benches. Experiments at a larger scale require a more efficient implementation of the score. The following measures are suggested to be taken to improve the speed and memory efficiency of the score computation:

1. Implement a custom *set* class which does not maintain a hash table as the python set class, utilized in this work, does.
2. Identify redundancies in the parameter set mapping functions for the different layer types.
3. Implement vectorized set operations and replace loops in the parameter set mapping functions by these operations.
4. Parallelize the score computation for the images of a batch.

When speed and memory consumption of the implementation are optimized to a sufficient degree it is suggested to analyse different choices for the score components:

1. Distance measure δ_λ for the LLO elements
2. Distance measure δ_ω for the parameter sets
3. Similarity measure φ
4. Aggregation over the LLO
5. Aggregation over the Batch

This includes research into the normalization factors for each of the components. For the choice of the similarity function, a concrete method to find a suitable function is proposed: one can map a set of architectures in a three dimensional space spanned by the distances of the parameter sets, the distance of the LLOs, and the validation performance of the architectures. One can then pick the function of the distances of the parameter sets and the distance of the LLOs that best fits the validation performance.

Once a high performing score is found, an ablation study needs to be conducted to verify that the score does not overfit the architecture search space it was developed on. It should further be a subject of the ablation study to show empirically that the score

1. is not correlated with the number of parameters of the network,
2. is independent from the batch size, and
3. is independent from the initialization of the network.

It is possible that the correlation of the score with the validation accuracy of a networks increases significantly if the networks are trained for a few epochs. It could

be subject of further research to examine if a potentially higher correlation is worth the increase in computational cost caused by partial training of the architectures.

This work discusses the score based on image classification. Further work could examine how the presented approach scales to regression and object detection tasks.

A more high-level suggestion to improve over the presented score approach is the following: instead of quantifying the pairwise dependency of the LLOs by the parameter sets one could quantify them by the gradient of the LLOs with respect to the parameters of the network. The intuition behind the gradient is that it reflects how much and in which direction each parameter influences the LLO elements. The suggestion is motivated by two disadvantages of the set-based quantification of the LLO dependence: The computation of the parameter sets is complex and it does not reflect the direction and the extent to which a parameter influences an LLO element.

This thesis is concluded with a vision for the global development of the domain NAS without training: The current methodology in NAS is to compare different candidate architectures and pick the candidate with the highest validation accuracy or, in the case of NAS without training, with the highest score. This methodology requires to examine a large number of architectures and introduces a bound to the performance in that the highest performance that can be achieved is the performance of the best candidate architecture in the search space. The vision for the future of NAS is to design a network directly for a given data set instead of comparing candidates for their performance on the data set.

The first fundamental prerequisite to design architectures directly for a given data set is to be able to formulate which capabilities a data set requires from a network to solve the task. The second prerequisite to design architectures directly for a given data set is to understand how the operations in the network influence the capabilities of the network. This knowledge is needed to know which operations to combine in which way to match the capabilities required by the data set. The parameter sets computed in this work enable analysis of how the different operations in a network influence the mapping of the data points. Therefore, this work contributes one tool to aid gaining the required understanding in the influence of the network's operations on the network's capabilities.

Bibliography

[1] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *CoRR*, vol. abs/1611.01578, 2016. [Online]. Available: http://arxiv.org/abs/1611.01578

[2] C. Ying, A. Klein, E. Real, E. Christiansen, K. Murphy, and F. Hutter, "Nas-bench-101: Towards reproducible neural architecture search," *CoRR*, vol. abs/1902.09635, 2019. [Online]. Available: http://arxiv.org/abs/1902.09635

[3] X. Dong, L. Liu, K. Musial, and B. Gabrys, "Nats-bench: Benchmarking NAS algorithms for architecture topology and size," *CoRR*, vol. abs/2009.00437, 2020. [Online]. Available: https://arxiv.org/abs/2009.00437

[4] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[6] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[7] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," *CoRR*, vol. abs/1611.02167, 2016. [Online]. Available: http://arxiv.org/abs/1611.02167

[8] X. Dong and Y. Yang, "Nas-bench-201: Extending the scope of reproducible neural architecture search," *CoRR*, vol. abs/2001.00326, 2020. [Online]. Available: http://arxiv.org/abs/2001.00326

[9] J. Siems, L. Zimmer, A. Zela, J. Lukasik, M. Keuper, and F. Hutter, "Nas-bench-301 and the case for surrogate benchmarks for neural architecture search," *CoRR*, vol. abs/2008.09777, 2020. [Online]. Available: https://arxiv.org/abs/2008.09777

[10] NVIDIA, "NVIDIA TITAN RTX:the ultimate pc gpu," 2019, 12th April 2022. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/titan/documents/titan-rtx-for-creators-us-nvidia-1011126-r6-web.pdf

[11] E. E. Agency, "Greenhouse gas emission intensity of electricity generation by country," 2022, 12th April 2022. [Online]. Available: https://www.eea.europa.eu/data-and-maps/daviz/co2-emission-intensity-9#tab-googlechartid_googlechartid_googlechartid_googlechartid_chart_11111

[12] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," *CoRR*, vol. abs/1802.01548, 2018. [Online]. Available: http://arxiv.org/abs/1802.01548

[13] H. Tanaka, D. Kunin, D. L. K. Yamins, and S. Ganguli, "Pruning neural networks without any data by iteratively conserving synaptic flow," *CoRR*, vol. abs/2006.05467, 2020. [Online]. Available: https://arxiv.org/abs/2006.05467

[14] J. C. Mellor, J. Turner, A. J. Storkey, and E. J. Crowley, "Neural architecture search without training (version i)," *CoRR*, vol. abs/2006.04647, 2020. [Online]. Available: https://arxiv.org/abs/2006.04647

[15] J. Mellor, J. Turner, A. J. Storkey, and E. J. Crowley, "Neural architecture search without training (version ii)," *CoRR*, vol. abs/2006.04647, 2020. [Online]. Available: https://arxiv.org/abs/2006.04647

[16] V. Lopes, S. Alirezazadeh, and L. A. Alexandre, "EPE-NAS: efficient performance estimation without training for neural architecture search," *CoRR*, vol. abs/2102.08099, 2021. [Online]. Available: https://arxiv.org/abs/2102.08099

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[18] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[19] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014. [Online]. Available: http://arxiv.org/abs/1409.4842

[20] B. Hanin and D. Rolnick, "Deep relu networks have surprisingly few activation patterns," 2019.

[21] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[22] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[23] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[24] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "A downsampled variant of imagenet as an alternative to the cifar datasets," 07 2017.

[25] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *CoRR*, vol. abs/1905.11946, 2019. [Online]. Available: http://arxiv.org/abs/1905.11946

[26] P. Verbancsics and J. Harguess, "Generative neuroevolution for deep learning," *CoRR*, vol. abs/1312.5355, 2013. [Online]. Available: http://arxiv.org/abs/1312.5355

[27] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," *CoRR*, vol. abs/1611.05431, 2016. [Online]. Available: http://arxiv.org/abs/1611.05431

[28] S. Saxena and J. Verbeek, "Convolutional neural fabrics," *CoRR*, vol. abs/1606.02492, 2016. [Online]. Available: http://arxiv.org/abs/1606.02492

[29] T. Elsken, J.-H. Metzen, and F. Hutter, "Simple and efficient architecture search for convolutional neural networks," 2017. [Online]. Available: https://arxiv.org/abs/1711.04528

[30] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," 2018. [Online]. Available: https://arxiv.org/abs/1808.05377

[31] B. Baker, O. Gupta, R. Raskar, and N. Naik, "Practical neural network performance prediction for early stopping," *CoRR*, vol. abs/1705.10823, 2017. [Online]. Available: http://arxiv.org/abs/1705.10823

[32] M. S. Abdelfattah, A. Mehrotra, L. Dudziak, and N. D. Lane, "Zero-cost proxies for lightweight NAS," *CoRR*, vol. abs/2101.08134, 2021. [Online]. Available: https://arxiv.org/abs/2101.08134

[33] N. Lee, T. Ajanthan, and P. H. S. Torr, "SNIP: single-shot network pruning based on connection sensitivity," *CoRR*, vol. abs/1810.02340, 2018. [Online]. Available: http://arxiv.org/abs/1810.02340

[34] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *CoRR*, vol. abs/1603.02754, 2016. [Online]. Available: http://arxiv.org/abs/1603.02754

[35] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *CoRR*, vol. abs/1810.00826, 2018. [Online]. Available: http://arxiv.org/abs/1810.00826

[36] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf

[37] H. Liu, K. Simonyan, and Y. Yang, "DARTS: differentiable architecture search," *CoRR*, vol. abs/1806.09055, 2018. [Online]. Available: http://arxiv.org/abs/1806.09055

A

Appendix

A.1 NAS-Bench-301 and DARTS

NAS-Bench-301 is the first publicly available surrogate NAS benchmark. The metrics in NAS-Bench-301 are obtained only for CIFAR10. The motivation to make a surrogate benchmark is the limited architecture search space of previous tabular approaches like NAS-Bench-101 or NATS. With a tabular bench in which every instance needs to be trained, these limitations are hard to overcome due to the vast computational expense. In a surrogate benchmark, however, the results for most instances are obtained by interpolating between a few actually trained instances. NAS-Bench-301 comprises 10^{18} architectures based on approximately 60 000 actually trained and evaluated architectures. This is many orders of magnitude higher than previous tabular benchmarks. The authors further claim that the surrogate benchmark is better than previous tabular approaches, not only due to the larger search space but also for a second reason: The performance of a single training run of an architecture can be seen as a single sample from a random variable due to the stochasticity of mini-batch-SGD. Tabular NAS benches can only hold a limited number of samples from this random variable. A surrogate NAS bench can provide a better proxy for the dispersion of the results by utilizing ensembles of surrogate regression models. This hypothesis is validated comparing NAS-Bench-101 and a surrogate model on the architectures of NAS-Bench-101.

As surrogate regression models for the performance metrics both XGBoost[34] and GIN[35] models are used. An ensemble of ten models of the same kind is used to make the predictions to provide a proxy for the dispersion of the results. The variation in the ensemble is achieved by training on different cross-validation folds and different parameter initializations. An LGBoost[36] surrogate regression model is used to predict the training times of the architectures. Other architecture metrics, such as the number of parameters and multiply-adds, do not require a surrogate model but can be queried exactly.

The 10^{18} architectures in NAS-Bench-301 are from the so called DARTS search space. In the first place DARTS[37] proposes a differentiable architecture representation that is utilized to solve the problem of NAS with a gradient descent optimizer. The paper however also introduces a architecture space that is often utilized for NAS benchmarking independently of the proposed NAS algorithm. As in previous NAS benches, the architectures are cell based. In NAS-Bench-301 the architectures are

built with stacks of each N=8 cells. Each cell is a DAG with V=7 nodes. A node is a feature map and an edge is a transformation of this representation, e.g. a convolution. A cell has two input nodes and a single output node. The two inputs are the outputs of the previous two cells. The output of a cell is the result of a depth-wise concatenation of all intermediate results in the cell. Each intermediate node is the sum of all its predecessors processed each by the operation of the connecting edge. The set of operations for a cell are 3×3 and 5×5 separable convolutions, 3×3 and 5×5 dilated separable convolutions, 3×3 max pooling, 3×3 average pooling, identity, and zeroize. The stride for all operations is 1. ReLU activation is applied before and batch norm is applied after each convolution. The cells at one third and two thirds of the depth of the network are reduction cells. All operations adjacent to the input node are of stride 2.

A.2 Further Parameter Differences Of Large and Small Architectures

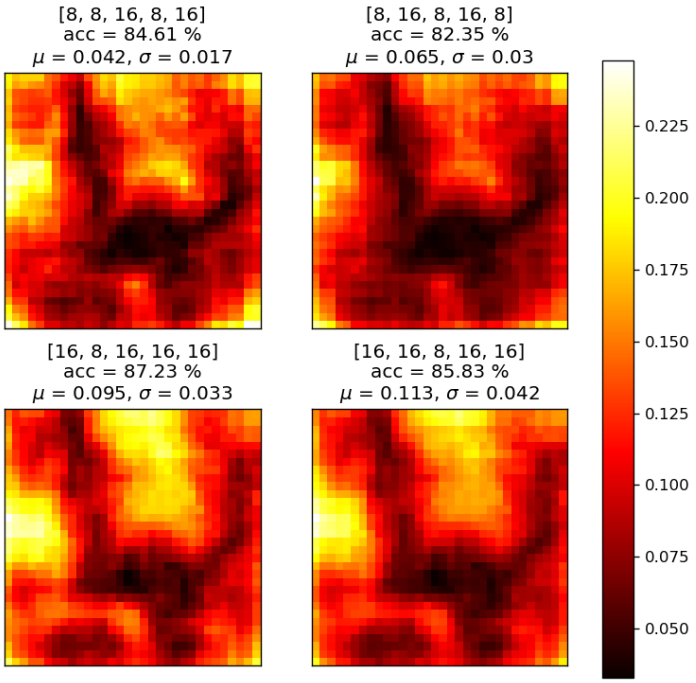

Figure A.1: A heat map of the parameter set differences for the two hard images x_{10} and x_{24} for the small architecture (8,8,16,8,16) with high performance, the small architecture (8,16,8,16) with low performance, the large architecture (16,8,16,16,16) with high performance, and the large architecture (16,16,8,16,16) with low performance.

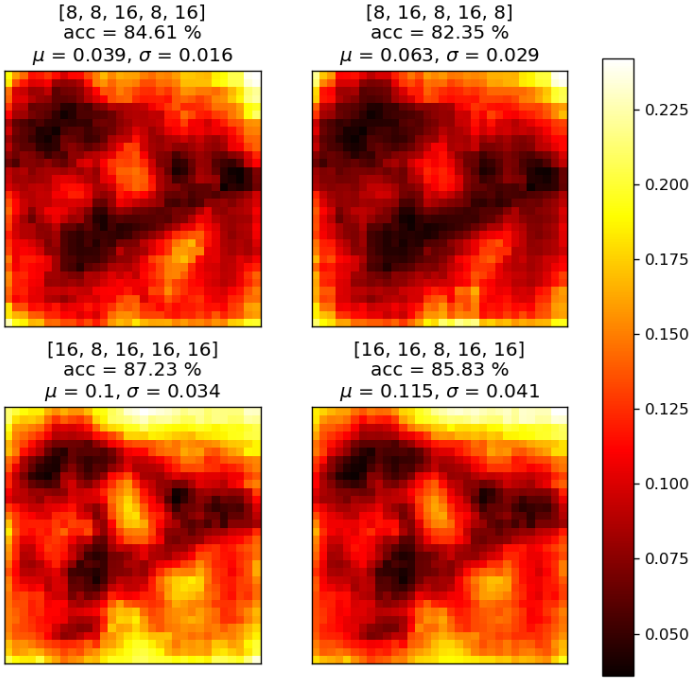

Figure A.2: A heat map of the parameter set differences for the two hard images x_{18} and x_{24} for the small architecture (8,8,16,8,16) with high performance, the small architecture (8,16,8,16) with low performance, the large architecture (16,8,16,16,16) with high performance, and the large architecture (16,16,8,16,16) with low performance.

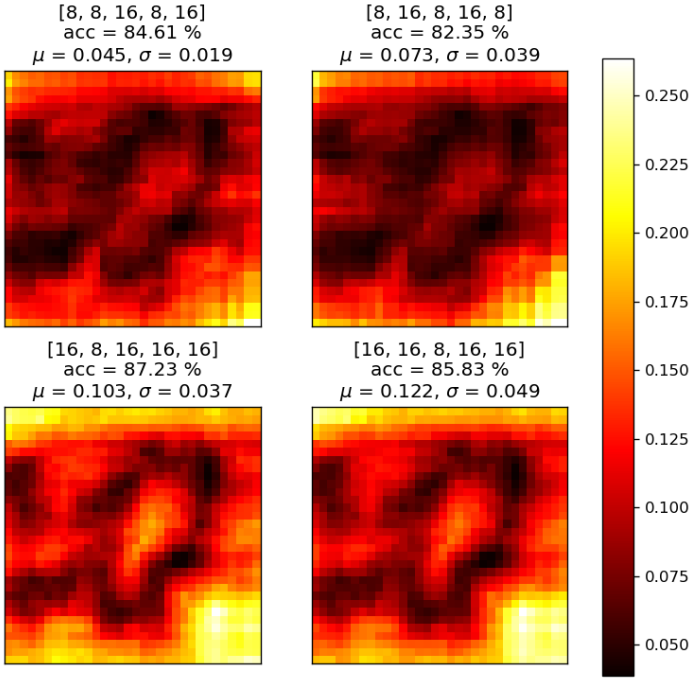

Figure A.3: A heat map of the parameter set differences for the two easy images x_4 and x_{13} for the small architecture (8,8,16,8,16) with high performance, the small architecture (8,16,8,16) with low performance, the large architecture (16,8,16,16,16) with high performance, and the large architecture (16,16,8,16,16) with low performance.

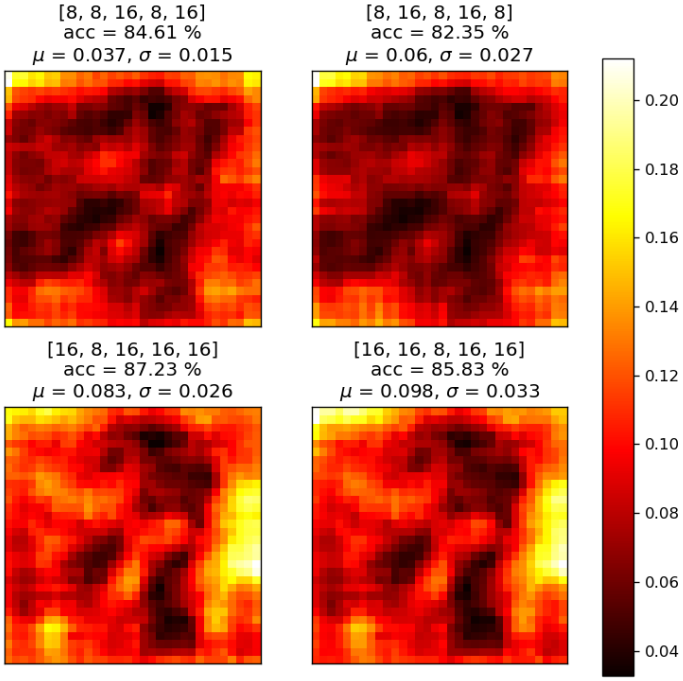

Figure A.4: A heat map of the parameter set differences for the two easy images x_{11} and x_{13} for the small architecture (8,8,16,8,16) with high performance, the small architecture (8,16,8,16) with low performance, the large architecture (16,8,16,16,16) with high performance, and the large architecture (16,16,8,16,16) with low performance.

A.3 Further Inter-Class Vs. Within-Class Parameter Set Differences

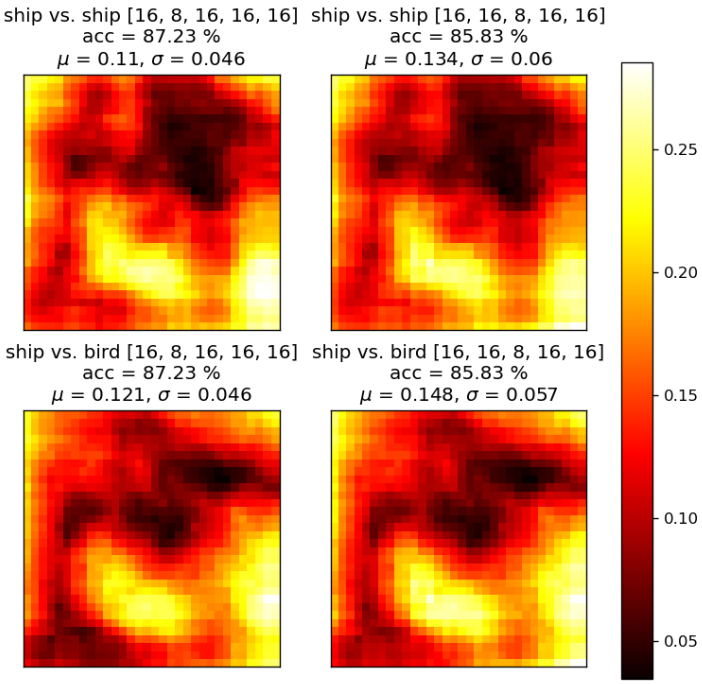

Figure A.5: Heat maps of the parameter set differences for the architecture (16,8,16,16,16) with high performance, and the architecture (16,16,8,16,16) with low performance, each once on the two images x_1 and x_2 from the same class, *ship*, and on the two images x_1 and x_3 from different classes; *ship* and *bird*, respectively.

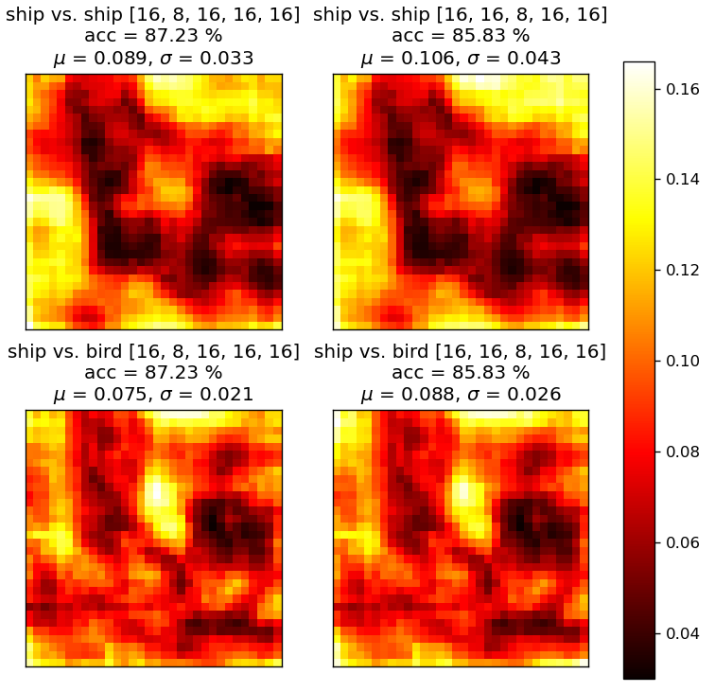

Figure A.6: Heat maps of the parameter set differences for the architecture (16,8,16,16,16) with high performance, and the architecture (16,16,8,16,16) with low performance, each once on the two images x_2 and x_{24} from the same class, *ship*, and on the two images x_2 and x_{23} from different classes; *ship* and *bird*, respectively.

A.4 Further Parameter Set Differences Vs. Local Linear Operator Differences

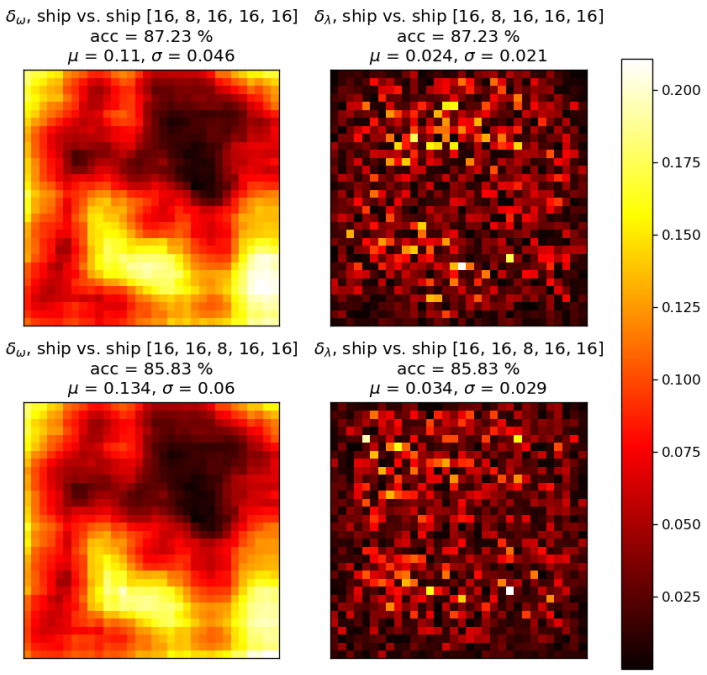

Figure A.7: Heat maps of the parameter set differences δ_ω and the LLO differences δ_λ, each for the architecture (16,8,16,16,16) with high performance, and the architecture (16,16,8,16,16) with low performance. The architectures are executed on the two images x_1 and x_2 from the same class, *ship*

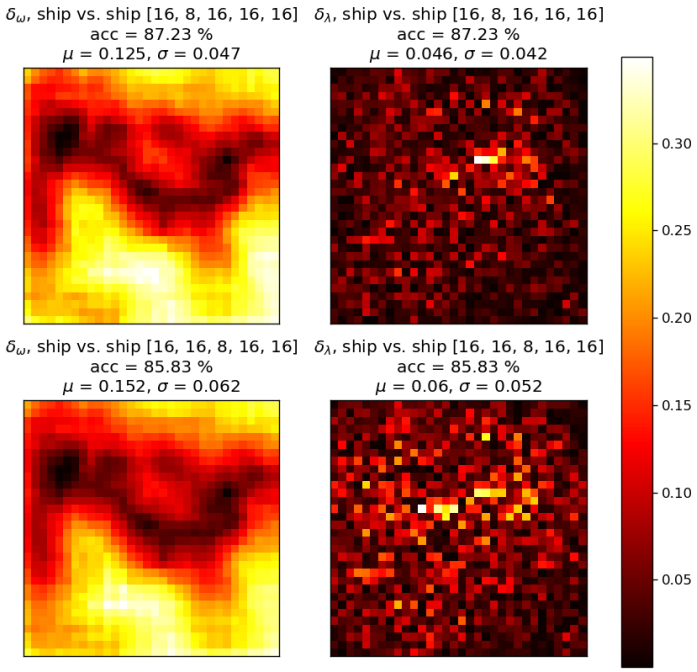

Figure A.8: Heat maps of the parameter set differences δ_ω and the LLO differences δ_λ, each for the architecture (16,8,16,16,16) with high performance, and the architecture (16,16,8,16,16) with low performance. The architectures are executed on the two images x_1 and x_{24} from the same class, *ship*

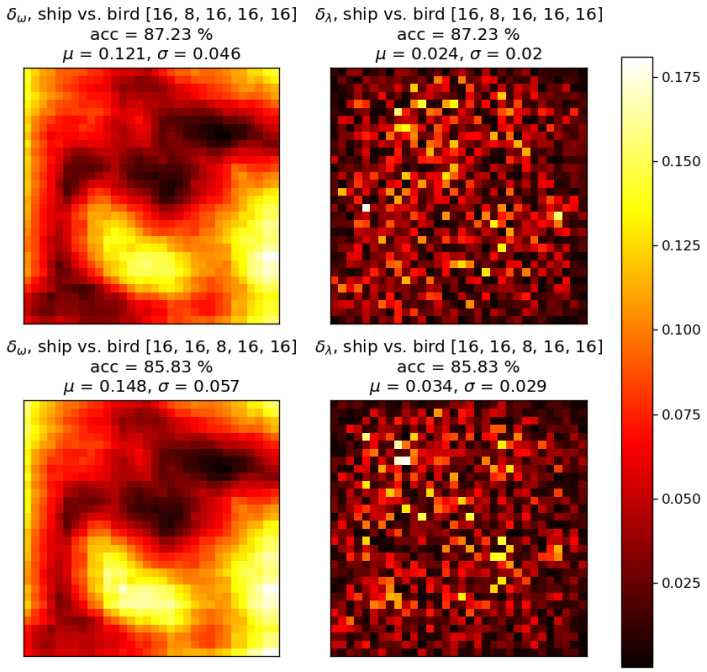

Figure A.9: Heat maps of the parameter set differences δ_ω and the LLO differences δ_λ, each for the architecture (16,8,16,16,16) with high performance, and the architecture (16,16,8,16,16) with low performance. The architectures are executed on the two images x_1 and x_3 from the classes *ship* and *bird*, respectively.

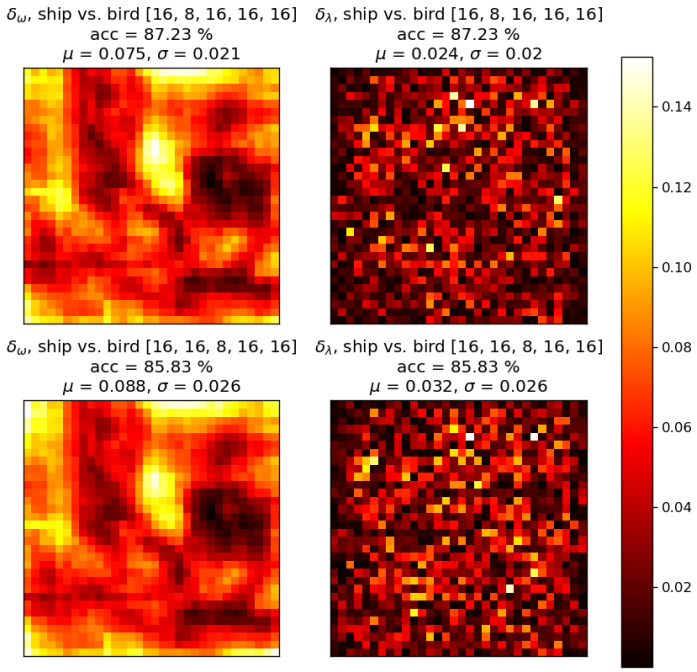

Figure A.10: Heat maps of the parameter set differences δ_ω and the LLO differences δ_λ, each for the architecture (16,8,16,16,16) with high performance, and the architecture (16,16,8,16,16) with low performance. The architectures are executed on the two images x_2 and x_{23} from the classes *ship* and *bird*, respectively.