Exploring Neural Network Architectures with Automated Machine Learning Approaches

Dissertation

University of Wuppertal
Dept Mathematics and IZMD

submitted by
Julian Burghoff, M. Sc.
to obtain a doctoral degree

Supervised by Prof. Dr. Hanno Gottschalk and Dr. Matthias Rottmann

Wuppertal, 23.09.2023

Acknowledgments

First of all, I would like to thank Hanno Gottschalk and Matthias Rottmann for their constant and intensive support and for giving me the opportunity to work in the field of machine learning. I have benefited greatly from your knowledge in this area, which will certainly help me enormously in my future career and for which I am very grateful.

I would also like to thank all my current and former colleagues in the Stochastics group, who have always listened to me and given me motivational support.

Moreover, I would like to express my gratitude to the ERDF and the "German Federal Ministry for Economic Affairs and Climate Action" who made this work possible through their funding.

Furthermore, I would also like to thank the companies ControlExpert and NeraCare, through whose cooperation I was able to carry out application-related projects together and create joint publications.

Last but not least, I would like to thank my parents, my wife and my friends who have provided me with support mentally and through proofreading.

Foreword

Parts of this work have been or will be published in the following papers:

- J. Burghoff, M. Rottmann, J. von Conta, S. Schoenen, A. Witte, and H. Gottschalk, *Resbuilder: Automated machine learning with residual structures*. Code: `https://github.com/JuliBu/ResBuilder`, 2023
  - Implementing the presented method
  - Generating and visualizing the results (except for benchmarks in industrial use case of ControlExpert)
  - Main contributions to writing the paper (except for section of industrial use case of ControlExpert)

- J. Burghoff, M. H. Monells, and H. Gottschalk, *Who breaks early, looses: goal oriented training of deep neural networks based on port hamiltonian dynamics*, 2023
  - Setting up the technical implementation
  - Supervising types of experimental setups
  - Supervising visualization
  - Writing section 4 of the paper

- J. Burghoff, L. Ackermann, Y. Salahdine, V. Bram, K. Wunderlich, J. Balkenhol, T. Dirschka, and H. Gottschalk, *Risk stratification of malignant melanoma using neural networks*, 2023
  - Setting up method
  - Implementing method on NeraCare's data
  - Writing sections 2, 3, 4 and 5 (except for Table 1 and clinical description)

Contents

Chapter 1

Introduction

Machine learning and so-called artificial intelligence (AI) are becoming more and more well-known and widespread these days, as can be seen as it triggers hot debates ([118], [88]).

But the idea of artificial intelligence is not new: One of the most important developments in the field of machine learning are neural networks, which date back to 1958 [103]. Inspired by the functioning of the brain, connections of artificial neurons were created that jointly form a multilayer perceptron - the first type of an artificial neural network. In the following, this type of networks was constantly developed further and new types of network architectures such as convolutional neural networks [68], residual structures in neural network architectures [50] or recurrent neural networks for the prediction of time-dependent correlations [78] were invented. Another development that has increased the utility of neural networks has been the introduction of deep neural networks [5]. Here, layers of neurons are stacked so that the number of weight parameters within the network increases significantly, which also positively affects in the prediction quality of the model.

The use of such technology not only offers new possibilities in the private sector, but companies also benefit from these developments. However, since not all companies have the know-how about machine learning to immediately integrate it into their processes, expert knowledge is often needed to handle the flood of adjustable hyperparameters [23]. This is often difficult to obtain and, in particular, expensive as the underlying data and problems are always of a different nature.

One way to simplify and accelerate such access to the technology of machine learning is "automated machine learning" (AutoML) [51]. In this process, hyperparameters of these technologies are automatically adapted to the user's problems and data sets and a good model is designed. Although there are many ways in

which AutoML can help in the development of a good predictive method, one important part of AutoML is the network architecture search (NAS) [33], which searches for suitable neural network architectures that provide high accuracy for a predefined computational cost.

Similarly, it is important to be aware of the choice of optimization algorithms available, which are behind the training of the machine learning models and how to look at how these algorithms work in order to obtain a functioning data science method. This helps in assessing problems that can arise in machine learning and out of this, techniques can be developed on how to eliminate these problems.

Furthermore, it is important to see that not only tasks like image recognition [107], natural language processing [83], object detection [98], image segmentation [91] and many more topics are suitable for machine learning applications, but there are also medical approaches that help in giving more certain medical prognosis suggestions [30].

In this thesis, the problem of the availability of machine learning technology is addressed by developing the well performing NAS-approach ResBuilder [18], which is able to search for architectures in depth as well as in width. To achieve this, it constructs ResNet-style [50] architectures from scratch or modifies a given architecture in order to achieve high accuracy on classification problems while achieving nearly state-of-the-art performance on many academic datasets for a set of default hyperparameter settings and also holds for an industrial use case application. For this purpose, different regularization terms are combined during the training of the neural network so that computing capacities can be optimally distributed within the entire architecture. This means that even companies with only a limited amount of computing capacity can build and use methods with residual architecture structures that are easy to use, since the number of computing operations can be set in advance.

Since the neural network optimization problem also strongly depends on the underlying optimization method, the port-Hamiltonian approach of an optimizer [17] provides a new instructive view on the stochastic gradient descent method. Here, the progression for the position of the optimization method in the loss landscape is considered as a heavy ball with friction, which rolls over local minima and can be seen as a physical application of the often used momentum strategy. It therefore can find better minima during training which results in a better overall accuracy. Furthermore, the method has been further developed in such a way that it can interrupt the training process in a goal-oriented manner, whereby minima are exploited in the best possible way, which has also proven to be a countermeasure to the overfitting problem.

Another problem besides overfitting that often arises in the evaluation of methods with neural networks is the challenge of the domain gap. This means that if the

data comes from only one source (such as images from a particular scanner in this example), it can happen that images from another source (another scanner) are predicted significantly worse or are not comparable with the predictions from the data from the first source. We address this problem in an use case where a method was developed together with NeraCare to determine a survival score for scans of malignant melanoma slides [16]. This is done by combining Cox's proportional hazards model with neural networks.

This thesis is therefore structured as follows:

First, there is a general introduction to the topic of machine learning and the basics of neural networks in Chapter 2, where the use of neural networks is mathematically substantiated. Then, Chapter 3 focuses on the explanation of the port-Hamiltonian approach to optimization problems, where our approach to the development of a physically motivated optimization algorithm is described and evaluated. Chapter 4 then provides an overview of meta-learning algorithms, laying the foundations for our NAS approach. Our ResBuilder method is also motivated and defined there. Since the ResBuilder is based on MorphNet, an AutoML method developed by Google, numerical results of our experiments with this method are provided in Chapter 5, before the results of the entire ResBuilder method are presented in Chapter 6. Subsequently, Chapter 7 will show the use case of neural networks in the field of survival analysis, where the problem of the domain gap will also be highlighted and addressed by suitable regularization techniques. Finally, Chapter 8 will provide a summarizing overview of the results discussed in this thesis.

Chapter 2

Basics of machine learning and neural networks

This chapter provides a short overview of the basics in machine learning because these insights are essential to understand further topics like Googles MorphNet ([42]) oder meta learning algorithms in general. Therefore we start giving some information on the theory of machine learning in Sec. 2.1 where we explain its concept. Then we take a look at the basic definitions of neural networks and their components in Sec. 2.2. Sec. 2.3 concludes with an overview of the datasets used in this work and their characteristics.

2.1 Machine Learning in general

Machine learning describes the computational way to generate knowledge from experience data. Widely known is the more abstract definition of Mitchell: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E." [80].

Like in [41], we will have a look on each of these mentioned parts of Mitchells quote:

2.1.1 The Task T

The usecases of machine learning technology are widely spread:
Besides tasks like computer vision tasks [61] or language processing [24], machine

learning is also used for autonomous driving [36], predicting medical information [34] and much more. Although there are much more possible abstract categories for use cases of machine learning technologies (see chapter 5.1.1 in [41]), we focus on **classification** tasks in this work but we will also have a look on **survival analysis** in Chapter 7:

Classification Classification is the task to assign a category $y \in \mathcal{Y}$ to an input vector $x \in \mathbb{R}^n$. Therefore the program to solve this task can be seen as function $f : \mathbb{R}^n \to \mathcal{Y}$. For simplyfication reasons we assign each category to a numerical index, such that we can set $\mathcal{Y} = \{1, \ldots, k\}$ for k as the number of categories.

An example for such a classification task gives the Fashion-MNIST dataset[126] which consists of 70,000 pictures of fashion products. There are 10 different categories of these clothing pieces like sandals, dresses or T-shirts (see 2.3.2 for further information). Each picture consists of 28×28 gray scaled pixels and so x is a vector of size 784 and the outcome categories are the 10 different types of fashion product. So the classification task is to find a function $f : \mathbb{R}^{784} \to Y$ with $Y = \{1, \ldots, 10\}$.

Survival Analysis Survival analysis covers predicting the probability of absence of an event (e.g. the death of a patient) until time t using the parameters β of a suitable model [47]. One of the most widely used methods is Cox's Proportional Hazards Model which we will focus on in Chapter 7 [16].

2.1.2 The Experience E

In general, our experience E can be understood as a dataset, on which the machine learning algorithm is trained and later evaluated. While machine learning algorithms can be separated in **supervised** and **unsupervised** learning methods, in this work we focus on supervised learning which means that all of our datasets have labels. In supervised learning, it is important to divide the available database, consisting of input data x that has to be learned and associated labels y, into at least two disjoint partial data sets (training and test data). Often, an additional validation set is also used, especially if, for example, the test data is only provided without a label for the time of developing and evaluating the method.

2.1.3 The Performance Measure P

In order to evaluate the performance of a machine learning model we are having a look on the rate of correct predictions in a validation dataset, we introduce the well known confusion matrix as shown in Table 2.1.

	Actual Positive	Actual Negative
Predicted Positive	true positives (TP)	false positives (FP)
Predicted Negative	false negatives (FN)	true negatives (TN)

Table 2.1: Confusion Matrix.

The Confusion Matrix provides various suitable measures that can be used to evaluate the quality of machine learning methods. The most common measures are:

- Accuracy: $\frac{TP+TN}{TP+TN+FP+FN}$
- Precision: $\frac{TP}{TP+FP}$
- Recall: $\frac{TP}{TP+FN}$
- Specificity: $\frac{TN}{TN+FP}$

In order to be able to realise a performance measure for a machine learning method, a machine understandable solution has to be found, which is called the *loss function*, where we will have a closer look on in Sec. 2.2.3.

However, it is often not only performance measures derived from the number of correctly classified data points that are of interest. Especially in the field of automated machine learning (AutoML), a measure such as the computing capacity used by the model at hand is also interesting, which is why it also has an influence on the loss function described in Sec. 2.2.3.

More information about the performance measure can also be read in chapter 5.1.2 of [41].

2.2 Feed-Forward Neural Nets

Neural networks are playing an increasingly important role in machine learning. Inspired by the layout of the human brain, these structures are constructed in layers and can solve complex non-linear problems like image classification, natural language processing or providing medical diagnoses.

The notation of the following definitions is mostly based on [41] or [110].

A general, simple neural network consists of one *input layer* l_1, one *output layer* l_k and an arbitrary number of so-called *hidden layers* $l_2, \ldots, l_{k-1}$ in between. Each layer l_i consists of a number of neurons which are connected to the neurons the following layer by a weighted edge $\omega_{a,b}^{(l_i)}$, with a the index of a neuron in l_i and b the index of a neuron in l_{i+1}. Let $|F|$ denote the depth of the network which is defined as the total number of layers without taking the input layer l_1 into consideration. The width $|l_i|$ of a fully connected layer l_i is defined as the number of neurons within this layer including the *bias* neuron, which are neurons that do not have an incoming edge from the previous layer but have an outgoing edge to every neuron in the following layer.

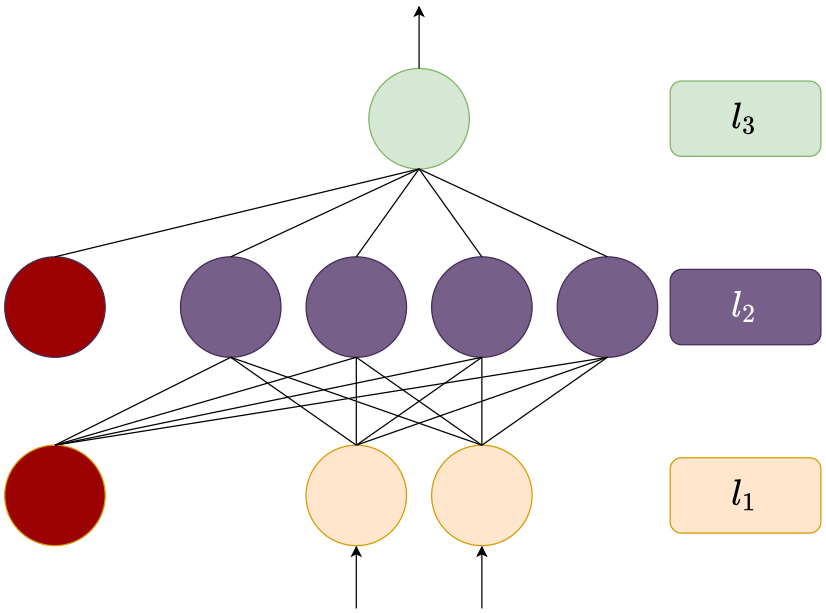


Figure 2.1: An example for a simple neural net.

In Fig. 2.1 an example of such a simple net with a depth of $|F| = 2$ and a maximum width of $|l_2| = 5$ is given. Between the orange input-layer and the green output layer there is one violet hidden layer. The red neurons represent bias neurons.

To infer data through this net, an input vector $x_0 \in \mathbb{R}^{|l_1|-1}$ is given to the first layer and the value of the output layer can be calculated by a concatenation of the different layers:

$$F(x_0, \omega) = (f^{(l_{|F|})} \circ f^{(l_{|F|-1})} \circ \ldots \circ f^{(l_1)})(x_0, \omega) \tag{2.1}$$

The function $f : \mathbb{R}^{|l_i|} \to \mathbb{R}^{|l_{i+1}|}$ between the layers l_i and l_{i+1} is defined as:

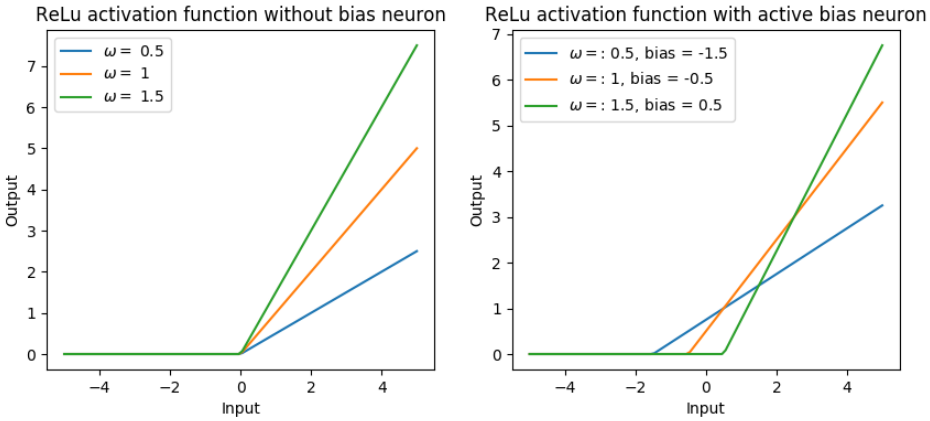


Figure 2.2: Influence of a bias neuron on the rectified linear unit activation function.

$$x_{i+1} = f^{(l_{i+1})}(x_i) = \Phi^{(i+1)}(W^{(i+1)}x_i + b^{(i+1)}) \forall i = 1, \dots |F| - 1 \tag{2.2}$$

where x_i is the output of layer l_i and so the input of layer l_{i+1}, Φ the activation function which will be described further in Sec. 2.2.1, $W^{(i)} \in \mathbb{R}^{|l_i| \times |l_{i-1}|}$ the matrix of all weights $\omega^{(i)}$ between the layers l_{i-1} and l_i and $b^{(i)}$ a bias neuron which is added to the layer l_i and ensures that the activation function is applied in the right place. This allows the activation function (Sec. 2.2.1) to be used much more flexibly, as can be seen in Fig. 2.2.

2.2.1 Activation functions

In order to fulfill the universal approximation theorem (Subsec. 2.2.2) it is necessary that at least one layer contains a non-polynomial function. This is usually realized with the implementation of an activation function for every neuron. Different types of non-linear functions are possible, where it is generally reasonable to choose a function which is differentiable in order to train the network properly.[1]

- The **sigmoid** function is defined by

$$\text{sig}(x) = \frac{1}{1 + e^{-x}} \text{ and so its derivative is } \text{sig}'(x) = \frac{e^{-x}}{(1 + e^{-x})^2}. \tag{2.3}$$

- As it can be seen for example in Fig. 2.3 the **hyperbolic tangent** is closely related to the sigmoid function.

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \text{ with } \tanh'(x) = 1 - \tanh(x)^2 \tag{2.4}$$

[1]For example for using SGD. See Sec. 2.2.4 for more information.

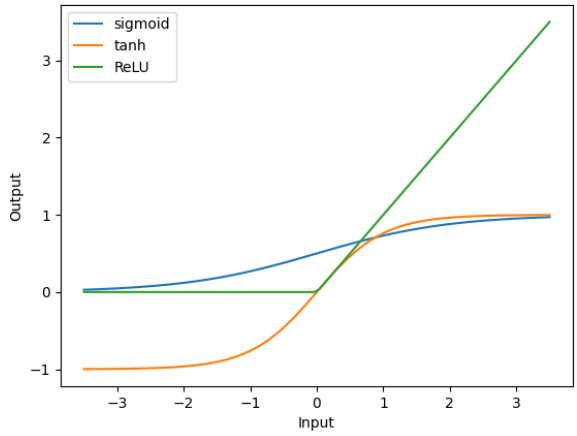


Figure 2.3: Overview of different activation functions.

- One solution which is often used in image classification is the **Rectified Linear Unit (ReLU)**:

$$\mathrm{ReLU}(x) = \max\{0, x\} \tag{2.5}$$

  Although it is not differentiable in zero, this problem is not relevant in practical use.

- The **Softmax** function $\sigma : \mathbb{R}^{|\mathcal{Y}|} \to (0,1)^{|\mathcal{Y}|}$ is a special type of activation function as it is usually used for the last layer of a neural network to determine an interpretable probability from the features of the neural network output:

$$\sigma(x)_i = \frac{e^{z_i}}{\sum_{j=1}^{|\mathcal{Y}|} e^{z_j}} \forall i = 1, \ldots, |\mathcal{Y}| \tag{2.6}$$

  with z_i the activation of the neurons in the last layer $l_{|F|}$. In further notation we denote $\hat{y}_i = \sigma(F(x_i))$, which is the predicted label of x_i.

2.2.2 Universal Approximation Theorem

Whether machine learning algorithms in the form of a neural network can solve a given problem at all (and if so, how accurately) is answered by the universal approximation theorem: Let $F(x, \omega)$ be the function obtained from the neural network before softmax, which is continuous and bounded and should approximate $F^* : \mathbb{R}^n \to \mathbb{R}$. With a given input data set $\mathcal{X}$ and the associated labels $\mathcal{Y}$, we assume that there is a labelling function F^* that exactly satisfies $\sigma(F^*(x)) = y, x \in \mathcal{X}, y \in \mathcal{Y}$ with σ the softmax activation. The universal approximation theorem now states that no matter what the optimal function F^*

looks like, a sufficiently large "multilayer perceptron" (i.e., a neural network with simple, dense layers) can represent it arbitrarily well as long as it has at least one hidden layer with a "squashing" activation function (e.g., the logistic sigmoid activation function) [41]. This was originally proven in [56] and in a more recent work [72] it was proven that this theorem also holds for other activation functions like the nowadays often used ReLU activation function.

2.2.3 Lossfunction

As already mentioned in Subsec. 2.1.3 choosing a suitable lossfunction is very important to guarantee the learning success of the neural network.

For this purpose, we first define the concept of empirical risk minimization in analogy to [110]. Let $S \subset (\mathcal{X}, \mathcal{Y})$ be a randomly drawn training set from the population of a data set that follows a distribution $\mathcal{D}$ and has been labelled by an objective function F^*. The algorithm for minimizing empirical risk (in the following referred to as ERM learner) should now find an optimal assignment $F_S : \mathcal{X} \to \mathcal{Y}$ that minimizes the error with respect to the unknowns $\mathcal{D}$ and F^*. For this purpose, the training error is defined as follows:

$$L_S(F) = \frac{|\{i \in \{1, \ldots, |\mathcal{X}|\} : \hat{y}_i \neq y_i\}|}{|\mathcal{X}|} \tag{2.7}$$

with $\hat{y}_i$ is the by F predicted class label of x_i.

Cross-Entropy loss A slightly more advanced idea for the loss function is the so-called **cross-entropy**. It was motivated in [105] in 1997 and adapted for the discrete case in 1999 [104]. For two probability distributions P, Q the cross-entropy $H(P, Q)$ is defined as

$$H(P, Q) = -\mathbb{E}_{X \sim P} \log Q(x) \tag{2.8}$$

for what it is similar to the well known Kullback-Leibler-Divergence $D_{KL}(P||Q)$ [66]: $H(P, Q) = H(P) + D_{KL}(P||Q)$.

In the use case of a loss function in order to train a neural network we then receive the following term:

$$L_{CE} = -\sum_{i=1}^{|S|} y_i \log F(x_i) \tag{2.9}$$

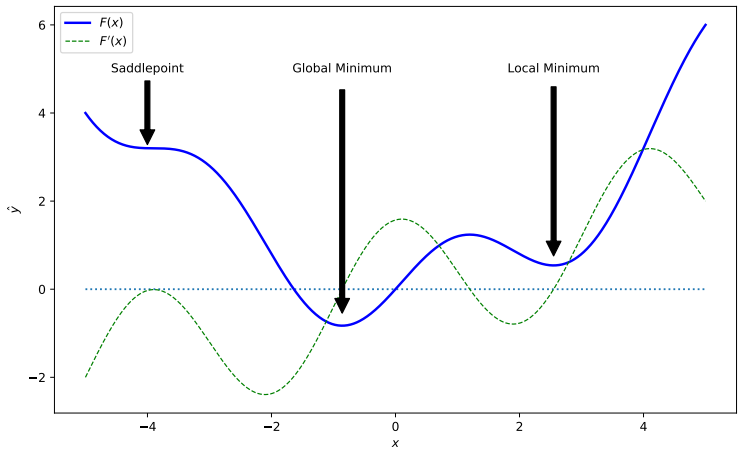


Figure 2.4: Schematically shown loss landscape with a local and global minimum and a saddlepoint.

for a given training set S, y_i the true label of the input data x_i, $(x_i, y_i) \in S$ and $F(x_i)$ the output (before softmax) of the considered neural net.

2.2.4 Gradient based Optimization

In order to minimize the loss function L, various methods have been developed that adjust the weights ω of a neural network $F(\omega, x)$ in such a way that the value of the loss function successively decreases. The task now is to find an as good as possible local minimum of L, at best even the global minimum.

As it can be seen in [41], the idea for a gradient based method is that the value of a function F becomes smaller in the direction of its negative derivative ∇F, at least for an ϵ-sized step: $F(x - \epsilon \operatorname{sign}(\nabla F(x))) < F(x), \nabla F(x) \neq 0$ for a small enough ϵ as it can be seen in Fig. 2.4. In the case of $F'(x) = 0$, this equation does not provide any information on how to proceed to minimize the loss, but it does tell us that we are in a stationary point. This can either be a local or global minimum or maximum, or a saddle point. This idea goes back to 1847[20].

Fig. 2.4 also shows at which point we would like the algorithm to converge: While for the example shown, the worst case would be for the optimization algorithm to converge at the saddle point on the left, convergence at the local minimum on the right would be better. Optimally, the algorithm finds the global minimum in the middle. For this, however, it would have to overcome the maximum between the two minima in case of a randomly selected starting point on the right side (e.g. $x > 4$), which could be problematic for naive algorithms.

2.2.4.1 Stochastic Gradient Descent (SGD)

A widely used approach to make adjustments to weights goes back to 1951, where Stochastic Gradient Descent finds its roots [101].

In the application, ϵ is often referred to as the learning rate α, as this indicates the step size with which the weight updates are completed. For every training step t each weight of ω experiences an update due to the stochastic gradient descent:

$$\omega_{t+1} = \omega_t - \alpha \nabla_{\omega_t} \mathcal{L} \tag{2.10}$$

with $\nabla_\omega \mathcal{L}$ the gradient of the lossfunction as defined in (2.11).

Optimally, one would perform the gradient descent on all data from the complete data set. However, this is impractical in practice, as

- With large amounts of data and high-dimensional data, there are problems with the size of the random access memory of the underlying system.
- (2.11) converges with the law of large numbers [7, 29]. However, if $\mathcal{L}$ is already sufficiently converged, the investment of additional computational resources in (2.11) is not profitable.
- In strongly non-convex optimization problems, stochasticity can help in the selection of mini batches to overcome local minima.

This is why stochastic gradient descent performs batchwise weight updates resulting in the following term for the gradient of the loss function:

$$\nabla_\omega \mathcal{L} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\omega L(\hat{y}_i, y_i) \tag{2.11}$$

SGD with momentum An extension of the original SGD algorithm is to apply a momentum to the changes due to the gradients, which makes it possible to "roll over" bad local minima in the optimization process. For this purpose, equation (2.10) is updated accordingly:

$$\mathbf{v}_t = \lambda_{\text{mom}} \mathbf{v}_{t-1} + (1 - \lambda_{\text{mom}}) \nabla_{\omega_t} \mathcal{L} \tag{2.12}$$

$$\omega_{t+1} = \omega_t - \mathbf{v}_t \tag{2.13}$$

for a given momentum strength λ_{mom}.

PHS-based Optimization We provide another interpretation of the SGD with momentum in [17], where we imagine the optimization model in the hilly loss landscape as a heavy ball with friction. The updates of the weights (as described for SGD in (2.10)) then work as follows:

$$\omega_{i+1} = \omega_i + \alpha \frac{1}{\mathbf{m}} p_i \text{ with} \tag{2.14}$$

$$p_i = p_{i-1} - \alpha \frac{\rho}{\mathbf{m}} p_{i-1} - \alpha \nabla_\omega \mathcal{L} \tag{2.15}$$

with a friction coefficient ρ and a predetermined mass $\mathbf{m}$ which is why one could interpret p_i as impulse in step i.

2.2.4.2 Further Optimizers

Like [41], we also have a short look at other developments that have emerged from the SGD:

AdaGrad In addition to stochastic gradient descent, there are other gradient-based optimization methods such as AdaGrad [32], which is based on SGD but has an individual learning rate α_i for each weight $omega_i$ depending on the value of its partial derivative ∇_{ω_i} in relation to the average value of all partially derived weights.

RMSProp One weakness of the AdaGrad method is that if it takes too long to converge, the algorithm may settle into a local minimum along the way. By implementing an erasing memory, the RMSProp algorithm [119] eliminates this weakness, as it allows more big steps to happen a few iterations later, even if there is a big step at the beginning.

Adam The Adam optimiser[62], which is the short name of *adaptive moment estimation*, is based on the RMSProp method but applies a clever kind of momentum to it, making it the standard optimizer for most deep learning applications nowadays.

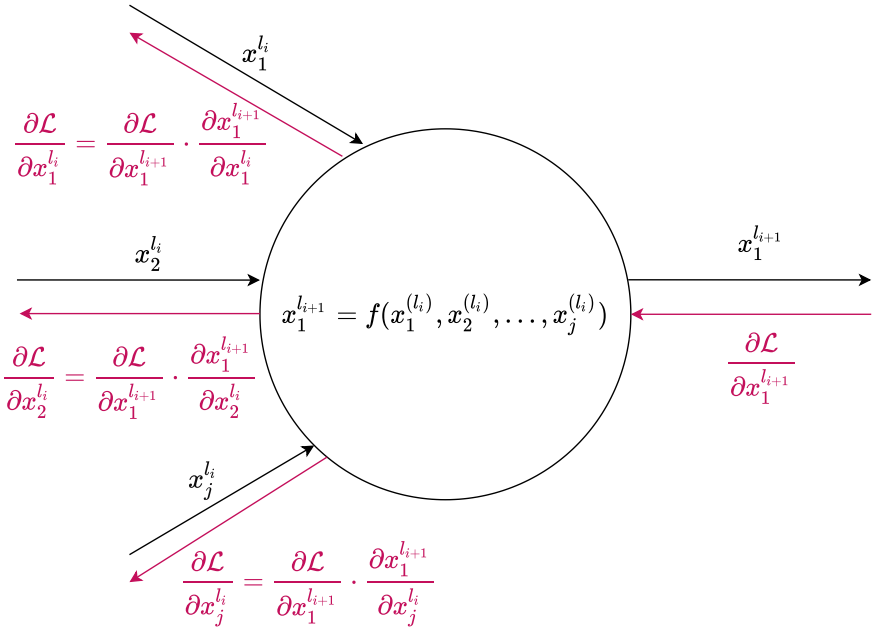


Figure 2.5: Backpropagation running through a neuron.

2.2.5 Backpropagation

This section on backpropagation [106] summarizes the whole process of weight adjustments and explains how the updates of the weights are done or how the partial derivatives for each of the several thousand weights can be determined.

To explain the process of backpropagation, we assume that we consider only one data point x at a time, even though in practical use mini-batches of data points are usually used. Furthermore, we assume that the neural net only consists of fully connected layers l_i and its weights $\omega^{(i)}$ are already initialized (with random values). The backpropagation process consists of 3 parts:

1. First, the data point x is propagated forward through the network.

2. Then, the loss function for the current state of the network regarding its weights ω and the currently considered data point x is calculated.

3. Finally, we look at the actual process of backpropagation, where we distribute the loss to the individual weights using the chain rule and partial derivatives.

Forward Propagation Since the neural network consists of several layers l_i, the input x is first propagated through the individual layers, whereby we denote the input of each layer with x^i in the following. The calculation of the output of each neuron is performed according to Fig. 2.5 while the calculation rule of f is stated in (2.2). From this we obtain the final output of the network $F(x_i)$.

Error calculation After we have received the output of the net, we can determine the value of our loss function $\mathcal{L}$ according to (2.9), since we also know the true value y of our training sample x.

Backpropagation In the last step, the gradient of the loss function $\nabla\mathcal{L}$ is calculated and passed on to the individual weights $\omega_j^{(l_i)}$ using the chain rule. What is meant here is the chain rule according to calculus and not that of the probability calculation:

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x} \tag{2.16}$$

This chain rule must now be applied to all intermediate calculations (such as an activation function or the summation part of a neuron) that lie between the output of the neural network and the weight under consideration. For ease of notation, let's combine the individual computations that happen inside a neuron into one, which we will call z. Further, assume that for each weight ω we have recorded the neurons that are passed during the forward propagation to the output value of the network in a list $z_\omega = (z_1, \ldots, z_m)$, which must now be processed in sequence in order to carry out the backpropagation from the loss function back to the weight.

$$\begin{aligned} \frac{\partial L}{\partial \omega_j^{(l_i)}} &= \frac{\partial L}{\partial z_1} \cdot \frac{\partial z_1}{\partial z_2} \cdot \frac{\partial z_2}{\partial z_3} \cdot \ldots \cdot \frac{\partial z_{m-1}}{\partial z_m} \cdot \frac{\partial z_m}{\partial \omega_j^{(l_i)}} \\ &= (\ldots((\frac{\partial L}{\partial z_1} \cdot \frac{\partial z_1}{\partial z_2}) \cdot \frac{\partial z_2}{\partial z_3}) \cdot \ldots \cdot \frac{\partial z_{m-1}}{\partial z_m}) \cdot \frac{\partial z_m}{\partial \omega_j^{(l_i)}} \end{aligned} \tag{2.17}$$

It is worth to notice that the partial derivatives can be calculated stepwise, as indicated in the second line of (2.17). This means that only one vector at a time has to be stored temporarily and not the entire Jacobian matrix. In each neuron, backpropagation is applied as shown in Fig. 2.5 in red.

It has to be considered that we must pay attention to the fact that the individual functions used must be differentiable within the structure of a neural network.

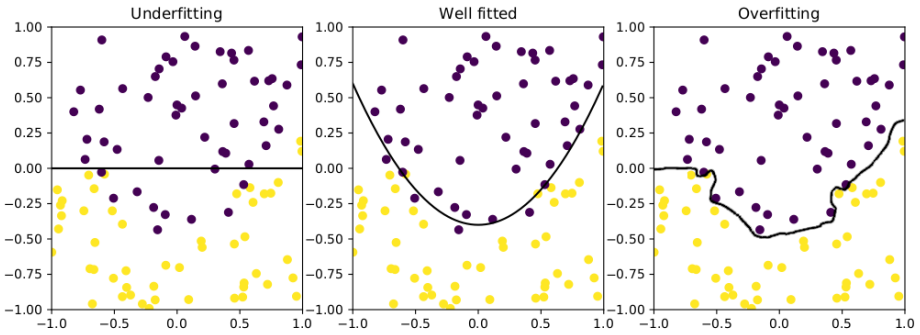


Figure 2.6: Under- and Overfitting a binary classification problem.

2.2.6 Overfitting

The idea of minimizing the empirical loss $\mathcal{L}$ on the training data set S is not a bad idea in the first place, but often leads to problems in application, as the algorithm can over adapt on S, causing it to perform poorly on data that was not part of the training. It only learns the labels $\mathcal{Y}_S$ of the training data by "heart" and does not generalize to unseen data. This problem is also called overfitting, which is the opposite of underfitting, which occurs, for example, when a method is trained for too short a time or the neural network lacks capacity.

The comparison of these two extremes is illustrated in Fig. 2.6 where a binary classification problem is schematically shown. The aim of the black line should be to separate the violet dots from the yellow dots. The very left plot shows the problem of underfitting as the line poorly separates the two classes. The plot in the middle the shows a good classification. Although some of the points near the boarder are classified to the wrong class, the pattern has been recognized what would lead to a good generalizability. The outer right plot shows the problem of overfitting: Even though every single point is categorized to the correct class label, this line would not perform good in the task of generalizability.

So just by observing the training progress in reducing the loss function, one cannot tell if it is affected by overfitting. Therefore, one should always pay attention to whether the accuracies of training and test (or validation) data differ too much.

Fig. 2.7 schematically shows a loss and accuracy curve as it could occur during the training of a neural network. At about epoch 150, the loss or accuracy on the validation data increases, which indicates that the method is now beginning to overfit. Precautions against such overfitting would be, for example, an early stopping of the training process or the regularization technique explained in Subsec. 2.2.6.2.

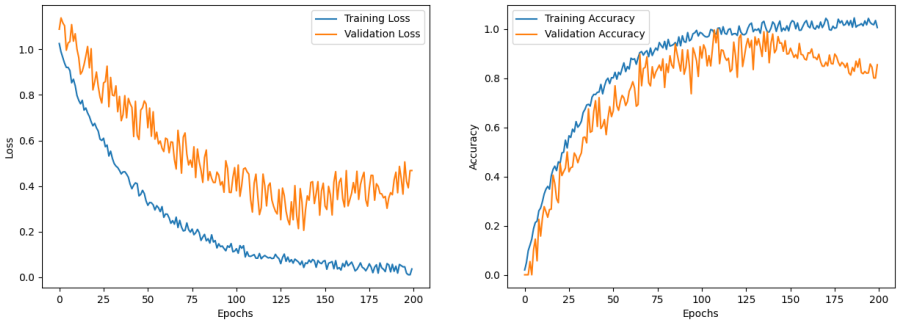


Figure 2.7: Indication of an overfitting problem while training.

2.2.6.1 Early stopping

One approach to reduce overfitting problems is the early stopping of the training process. If we consider Fig. 2.7 again, we could stop the training at about epoch 145 and have a better generalized model than if we would continue to train the model beyond that epoch. This had been shown in [82] where the problem of overparametrization is addressed.

2.2.6.2 Regularization

Another technique often used in machine learning methods is regularization. When regularizing, we penalize high values of the net's weights mainly in order to reduce overfitting. In this work, we also use regularization to artificially pull weights down, that the sum of weights in a specific filter/layer is below a given threshold to eliminate groups of neurons. See Sec. 5.1.1 for more information on this and Subsec. 6.1.1.2 for more information on the parameter settings that are used therefore.

To implement regularization we add a term of the form $\lambda_{str} \cdot \mathcal{G}(\omega)$ to the lossfunction $\mathcal{L}$ for every kind of used regularization. For the case of a typical cross-entropy lossfunction ((2.8)) we achieve:

$$\mathcal{L} = \mathcal{L}_{CE} + \lambda_{str} \cdot \mathcal{G}(\omega) \tag{2.18}$$

whereas λ_{str} is the choosable regularization strength and $\mathcal{G}$ is a function on the weights ω that corresponds with the values of each individual weight $\omega_{i,j}^{l_k}$. Most often used regularization techniques are L_1- or L_2-regularization:

L_1-regularization:

$$\mathcal{G}(\omega) = \sum_{k=1}^{|F|} \sum_{i=1}^{l_{k-1}} \sum_{j=1}^{l_k} |\omega_{i,j}^{(l_k)}| \tag{2.19}$$

L_2-regularization (also called *weight decay* [46]):

$$\mathcal{G}(\omega) = \frac{1}{2} \sum_{k=1}^{|F|} \sum_{i=1}^{l_{k-1}} \sum_{j=1}^{l_k} |\omega_{i,j}^{(l_k)}|^2 \tag{2.20}$$

for a network F consisting of $|F|$ layers whereas each layer l_k has $|l_k|$ neurons and l_0 are the input neurons.

2.2.6.3 Input augmentation

Besides regularization (see Subsec. 2.2.6.2) or early stopping the training process (2.2.6.1) there is also the input augmentation technique [112, 124] to reduce overfitting:

Therefore, the input data can be augmented randomly by several options: In case of image data as input one can do many things as zoom in the image, crop the image, shift it horizontally or vertically, flip or rotate it along its axis and many more. This also helps the training process to generalize.

2.2.6.4 Dropout

The authors of [117] have made another suggestion on how to avoid overfitting: Dropout. This means that one sets the output value of a certain percentage of the neurons in fully connected layers to 0. In this way, the neural network cannot rely on a few specific, important edges/weights in the network, but learns "more intensively". This has also been successfully tested for convolutional layers (which we will introduce in Subsec. 2.2.7.1).

2.2.7 Types of layers

A very common variant of layers in a neural network are the fully connected layers, as already described above. Here, each neuron of layer l_i has an incoming connection of each neuron from layer l_{i-1} and an outgoing connection to each neuron of layer l_{i+1}. To reduce the number of weighted edges and make the best use of regional information, several other types of layers have been developed, which are discussed below:

0	5	5	0	0	3	0
0	5	5	0	0	3	0
0	5	5	0	0	3	0
0	5	5	0	0	3	0
0	5	5	0	0	3	0
0	5	5	0	0	3	0
0	5	5	0	0	3	0

$*$

-1	0	1
-1	0	1
-1	0	1

$=$

15	-15	-15	9	0
15	-15	-15	9	0
15	-15	-15	9	0
15	-15	-15	9	0
15	-15	-15	9	0

Table 2.2: Example of a two-dimensional convolution.

2.2.7.1 Convolutional Layer

Especially for image recognition tasks the *Convolutional layers* [13] are of enormous importance. In order to be able to apply a convolution within a neural network to two-dimensional input data (such as an image) I, we define a filter (also called a kernel) K, which is pushed pixel by pixel over I. In the process, pixels can also be skipped periodically, which determines the stride of the convolution. In relation to the image size I_x (width of the input data) and I_y (height of the input data), the kernel dimensions K_x (width of the filter) and K_y (height of the filter) are usually significantly smaller: $K_x \ll I_x, K_y \ll I_y$. Since the entire input I is not considered at the same time, the part of the input regarded by the filter is referred to as the "receptive field" of a filter. Mathematically, the convolution operation $(I * K)$ for a given position (i, j) in the image can be seen as follows:

$$(I * K)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n) \tag{2.21}$$

However, most machine learning libraries tend to use the very similar cross-correlation, which is also called convolution in the machine learning field:

$$(I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n) \tag{2.22}$$

Table 2.2 shows a simple example of a convolution of a 7×7 input image with a 3×3 kernel. The term of (2.22) would look like this for $i = 0, j = 0$ (the blue colored cells):

$$(I * K)(1, 1) = \sum_{m=0}^{2} \sum_{n=0}^{2} I(1 + m, 1 + n)K(m, n)$$

$$(I * K)(1,1) = (0 \cdot (-1) + 5 \cdot 0 + 5 \cdot 1) \cdot 3 = 5 \cdot 3 = 15$$

The result of the convolution is called a feature map in the context of machine learning. As can be seen here, the convolution reduces the dimension by (filter size -1) in each image dimension. This could be avoided by using a so-called padding. This allows the filter to be pushed beyond the edge of the input as long as the center of the filter remains within the input data. With padding, non-existent pixels are simulated. The most common methods of assigning a value to this non-existent pixel is either to select the same value from the next existing pixel within the image, or to always use the value 0. While in the examples considered up to here only one color channel was considered at a time (as would be the case, for example, in a grey-scaled image), the procedure is of course also suitable for images with more channels (e.g. RGB images with 3 channels or ELA images with 6 channels). For this, the formula is adapted as follows:

$$(I * K)(i,j) = \sum_m \sum_n \sum_d I(i+m, j+n, d) K(m,n,d) \tag{2.23}$$

Even though only images as input data were mentioned here, the procedure for feature maps as input remains the same. Here, too, the dimension must be adjusted according to the depth of the previous feature map.

2.2.7.2 PoolingLayer

Another technique that is often used in convolutional neural networks is the pooling layer. These reduce the dimension of the feature maps in height and width and thus allow to find larger components of context within an image that would go beyond the receptive field of a single filter. Even though there are various other intuitions regarding pooling, such as (weighted) average pooling or using an L_2-norm on the neighborhood of the region under consideration, max pooling [129] has established itself.

To apply Max Pooling, we determine a region size K_x, K_y (analogous to the kernel size for convolutional layers) on which pooling should be applied. Often $K_x = K_y = 2$ is chosen, which halves the height and width of the feature map. Now the image is divided into subsections of size $K_x \times K_y$ and for each of these subsections the maximum is determined, which gives the value in the output feature map, as can be seen in Table 2.3.

0.25	0.51	0.52	0.11
0.34	0.24	0.73	0.02
0.31	0.65	0.37	0.76
0.69	0.42	0.17	0.23

→

0.51	0.73
0.69	0.76

Table 2.3: Example of a MaxPooling operation.

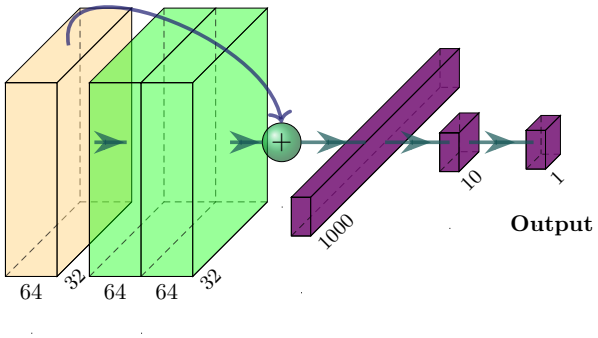


Figure 2.8: Residual block in a convolutional neural network.

2.2.7.3 Residual Layerblock

Even though it is not a layer species per se, we also consider residual structural elements here, as they are an important structural element of modern deep learning architectures. The problem with modern network architectures is often that they have many layers, which means that the gradient due to backpropagation hardly reaches the actual weights, which is called a vanishing gradient. As a result, the accuracy in deep networks decreases significantly. [49] To counteract this, residual structures were introduced that incorporate a skip connection across several layers, which means that the output of one (e.g. convolutional) layer is copied at an outgoing skip connection and added to the output of a succeeding (convolutional) layer [50]. This identity mapping makes the gradient in backpropagation vanish noticeably slower.

Fig. 2.8 shows such a residual block (green) in a small convolutional neural network.

When inserting such a structure, it is important to pay attention to the respective dimensions of the layers. E.g. the size of the feature map should be the same. If one wants to set up a cross-dimensional skip connection, one must again use some kind of padding. One idea in [50] is that these padded values could also be learned using 1x1 convolutions. With the ResNets, they also give examples in

different scales of how such residual blocks can be used in network architectures. These types of architectures have redefined many state-of-the-art benchmarks.

2.2.7.4 Other types of layers

In the context of convolutional nets, there are other types of layers, such as unpooling [128] or deconvolution [99] layers, which are often found in nets such as the U-Net [102]. However, these are not considered in more detail in the context of this work.

2.3 Academic Benchmark Datasets

In this work we evaluate our approaches to automated machine learning algorithms on different datasets. Most of them are academic datasets due to more comparable benchmark reasons. We used different sources for our dataset, which mainly have been keras [22] and the tensorflow-datasets package [3].

2.3.1 MNIST

MNIST, the modified version of the *NIST* dataset defined in [69], contains 60,000 training pictures of handwritten digits, as well as 10,000 pictures in a test dataset. All pictures have size 28×28 pixels and each pixel has a gray value. This dataset is commonly used for first experiences in machine learning as it provides good results quickly without much effort. We therefore also conducted our first experiments on this data set (see Chapter 6 for more information on the experiments of our *ResBuilder* method.) For this work we used the version available from the keras datasets [22].

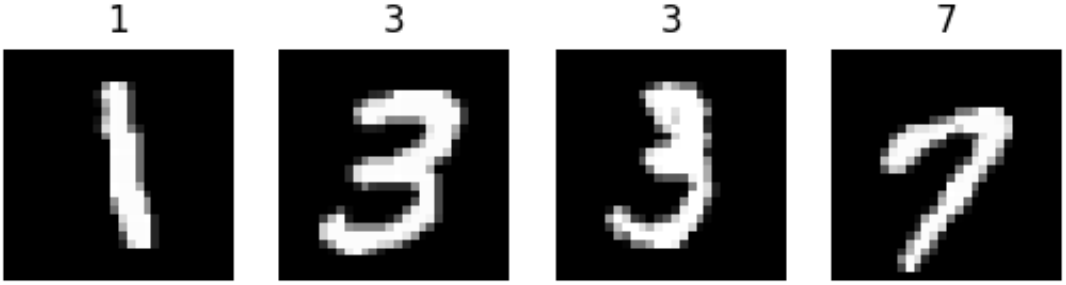


Figure 2.9: Four random images from the MNIST dataset.

2.3.2 FashionMNIST

As MNIST (2.3.1), **FashionMNIST** [126] is also one of the most often used benchmark datasets. In FashionMNIST we also have 70,000 gray-scaled pictures of the size 28×28, separated in 60,000 training and 10,000 test images. In contrast to MNIST the pictures of this dataset contain 10 different types of clothing which have to be classified. These types of clothing are: T-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag and ankle boot. Because this dataset is also easy to learn but a slightly more challenging task for neural nets than MNIST, it supplies a great opportunity to test new technologies implemented in the method. Some example images can be seen in Fig. 2.10. For this work we used the version available from the keras datasets [22].

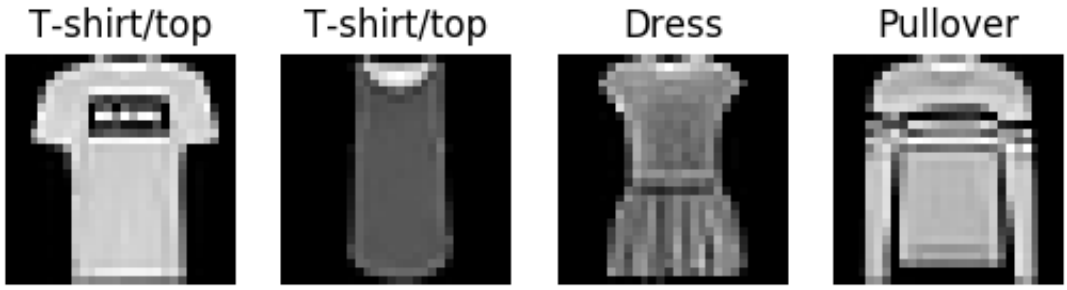


Figure 2.10: Four random images from the FashionMNIST dataset.

2.3.3 EMNIST

Another commonly used, MNIST-like dataset is **EMNIST** [25]. Here we also have grayscaled images of size 28×28 pixels, but in extension to MNIST (2.3.1) this dataset also contains handwritten letters, which also expands the number of classes from 10 to 62, where the first 10 classes represent the digits from 0 to 9, classes 11 to 36 the capital letters and the last classes from 37 to 62 stand for the lowercase letters. Like the original EMNIST data, images provided by the used tensorflow dataset [3] are inverted horizontally and rotated 90 degree anti-clockwise as we can see in Fig. 2.11. Overall there are 814,255 images in the dataset but in this case, the images are not equally distributed among the classes. Most images (44,704) are in class 1(digit "1") and least number of images is 2,213 for class 45 (which equals the lowercase character "i"). The dataset is split into 697,932 training images and 116,323 test images.

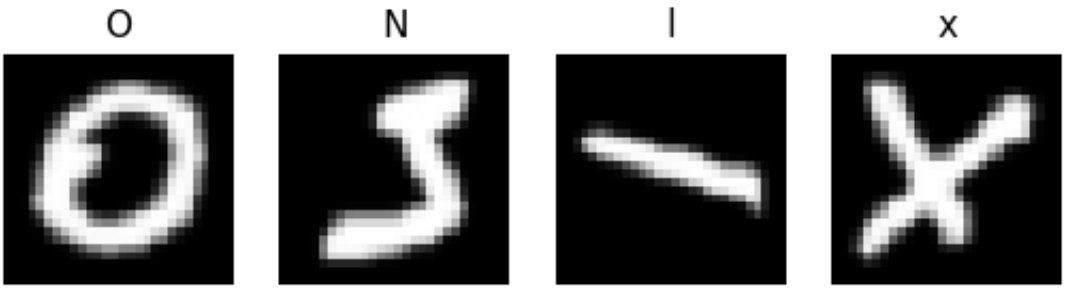


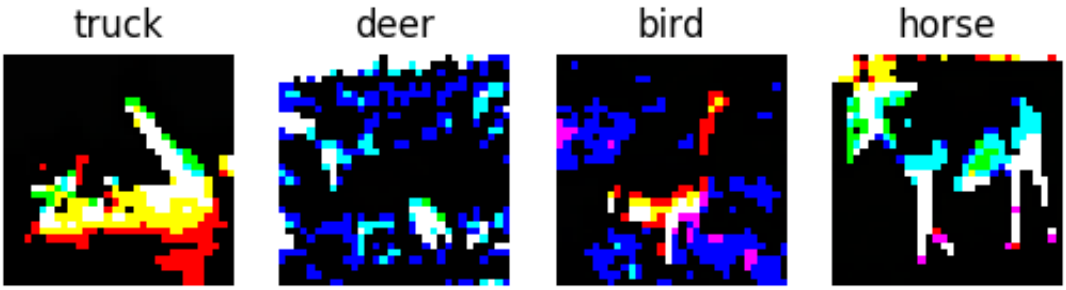
Figure 2.11: Four random images from the EMNIST dataset which are already flipped horizontally and rotated 90 degree anti-clockwise.

2.3.4 CIFAR10

The first colored image dataset in this work is the **CIFAR10** dataset [64] with which most of the experiments with the training pipeline have been performed. It contains 60,000 pictures (split in 50,000 training images and 10,000 test images) equally distributed in ten different categories, which are: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. All pictures have size 32×32 pixels where each pixel has a 3-channeled color codex.

A special feature concerning the CIFAR10 dataset is that it has been normalized before use, since this is often done [114], it not only increases the prediction quality of the networks, but at the same time generates a better comparability with other works. In Fig. 2.12 four random pictures of the CIFAR10 dataset are shown and Fig. 2.12(a) shows them before and Fig. 2.12(b) after the normalization process. For this work we use the version available from the keras datasets [22].

((a)) Raw versions of the images from the CIFAR10 dataset.



((b)) Images after the normalization process.

Figure 2.12: Four random images from the CIFAR10 dataset.

2.3.5 CIFAR100

The **CIFAR100** dataset [64] is similar to the CIFAR10 dataset (described in 2.3.4), it is just extended to 100 classes, but the number of pictures per category decreased from 6,000 images to 600 images, such that the total number of images in the dataset is still 60,000. For a full list of the classes have a look in [64]. The 100 classes are also grouped in 20 subclasses which are not used in this work. Like in CIFAR10 the pictures of this dataset have size 32×32 and a 3-channeled color codex. An example of pictures is displayed in Fig. 2.13.

Figure 2.13: Four random images from the CIFAR100 dataset.

2.3.6 Small NORB dataset

The Small NORB dataset [70] contains 97,200 96×96 pictures of five different categories of 50 different toys whereas the categories are airplanes, cars, four-legged animals, human figures and trucks. Each of the 50 toys is photographed under 6 different lightning conditions, 9 elevations (30 to 70 degrees, every 5 degrees) and 18 azimuths (0 to 340 every 20 degrees). In Fig. 2.14 we can see four random pictures from this dataset.

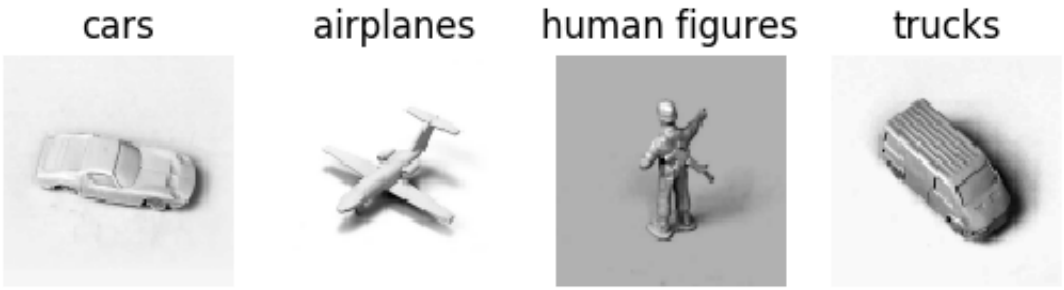


Figure 2.14: Four random images from the smallNORB dataset.

2.3.7 Animals10

The Animals10 [1] dataset contains 26,189 different images of animals acquired through Google Image Search and manually reviewed by a human. The images are divided into 10 different classes: dog, cat, horse, spider, butterfly, chicken, sheep, cow, squirrel, elephant. The images are not equally distributed among the classes, but each class contains between 2000 and 5000 images. In Fig. 2.15 there are shown four random example images. As we can see, the pictures do not all have the same size or ratio. Therefore, they are resized to the size 300×300 pixels for our purposes.

((a)) Cane/Dog ((b)) Cavello/Horse ((c)) Elefante/Elephant ((d)) Gatto/Cat

Figure 2.15: Four random images from the Animals10 dataset.

2.3.8 Overview of all datasets

Table 2.4 gives an overview of the datasets used in this work.:

Name	Content	Size (px)	Classes
Animals10 [1]	Animals	300	10
CIFAR10 [64]	Misc.	32	10
CIFAR100 [64]	Misc.	32	100
MNIST [69]	Digits	28	10
FashionMNIST [126]	Clothing	28	10
EMNIST [25]	Letters	28	62
SmallNORB [70]	Toys	96	50

Table 2.4: Overview of datasets used.

Chapter 3

Port-Hamiltonian Optimizer

As shortly mentioned in 2.2.4.1 it is worth to have a closer look on the optimizers which are used in neural networks. In this chapter, we will take a physical view of the optimisers by looking at the heavy ball with friction, whereas this chapter fully relies on [17]:

3.1 Motivation

The success of deep neural networks (DNN) significantly depends on the cheap computation of gradients using back-propagation enabling gradient based minimization of the loss functions. As the parameter count of DNN ranges between several tens of thousand in small classification networks to several billion in large scale generative models, there seems to be no alternative to the use of gradients. However, gradient based optimization is beset with the problem of local minima (as we have seen in Sec. 2.2.4), of which the energy landscape of DNN offers plenty. Exploitation of a local minimum with gradient descent comes with guarantees for progress relative to previous optimization steps, but does not guarantee a decent level of performance. In order to go more global, momentum methods have therefore been introduced to overcome local minima.

As compared to gradient descent, momentum based methods have more parameters to adjust. Besides the strength of the inertial forces controlled by the 'mass' parameter, a 'friction' parameter has to be determined, which is responsible for slowing down the search motion and bringing it to rest, ultimately. Finally, the learning rate needs to be controlled throughout the progress of the optimization process, like in gradient descent.

The complexity in setting and controlling the aforementioned hyperparameters can be alleviated by an interpretation of the optimization process in physical terms as already indicated by the physical connotations of 'mass' and 'friction'. It has been recently proposed to cast the optimization process in a port Hamiltonian framework, which makes the convergence of the optimization process to a stationary point transparent via energy based considerations, where loss is connected to potential and momentum to kinetic energy, whereas 'friction' accounts for energy dissipation and interdicts motion at high pace for unlimited time. It is clear that the friction / energy dissipation parameter is essential for the (non) locality of the optimization process: if high, friction essentially damps out all momentum and the procedure essentially 'just flows down the hill' as for gradient descent, resulting in low exploration and high exploitation. If low, the motion will go on essentially un-damped and not rest and thereby explore all of the accessible parameter space. Exploration is high, and exploitation is low in this setting.

Then, parameter settings can be modified over time or controlled adaptively as a part of the optimization algorithm is a familiar thought. The physics based intuition of port Hamiltonian systems can be helpful in the design of such adaptive strategies. Here we suggest a simple, event based adaptive parameter selection strategy that starts the optimization in an exploratory phase with low friction and turns over to exploitation by 'heavy breaking', once the potential energy (i.e. the loss function) is sufficiently reduced. Sufficiency is pre-defined as the minimum reduction goal of the optimization, which can be set, e.g., as the reduction of the loss obtained in previous trials.

In this paper, we show that the proposed strategy actually works for some classical examples in deep learning and improves the optimization loss and also the test accuracy for a standard, Le-Net-5 [68] based architecture on two well known academic classification tasks solved by deep learning, namely the CIFAR10 [64] and the FashionMNIST [126] data-sets.

In order to focus on the optimization only, we do not employ data augmentation or pre-training and thereby do not achieve SOTA performance in our experiments. We however consistently achieve an advantage over the widely used stochastic gradient descent as a benchmark. We also observe consistent gains in performance after 'heavy breaking' is finally triggered.

3.2 Related Work

The fact that neural networks with parameter counts ranging from some tenth of thousands to several hundreds of billions can actually be trained, largely depends on the cheap computation of gradients, see [69, 123] for original work and [41] for

a recent reference. Gradient based optimization itself has been studied since the days of Newton, see e.g. [9, 125]. In the context of deep learning, the formation of randomly sub-sampled mini-batches is necessary as big data often exceeds the working memory available [74]. One has therefore to pass over to the stochastic gradient descent method (SGD) [108, 110].

One of the problems in neural network training is the complex, non-convex structure of the energy landscapes [10]. This makes it necessary to avoid local minima, which is mostly done by the momentum method [39, 84, 96]. From a theoretical side, momentum can be understood as a discretized version of a second order ordinary differential equation, which also provides theoretical insight to convergence to critical points [6, 8, 93], see also [85, 86, 87] for recent extensions.

The momentum method has recently been cast in a modern port Hamiltonian language [63, 79, 92]. Port Hamiltonian systems [120] are particularly suited to understand the long time behavior and hence convergence properties of momentum based methods.

For a long time, the control of hyperparameters in the training of neural networks has been a topic of interest in the deep learning community [11]. While learning rate schedules [27, 28] determine the setting for one specific parameter upfront, it has also been proposed to modify the dissipation parameter in momentum based optimization [8, 19, 21]. Other strategies, like the often used ADAM algorithm, rely on adaptive parameter control [12, 62].

One specific adaptive strategy however much less considered is the goal oriented search, where one pre-defines the target value to achieve during optimization, see e.g. [115].

In this chapter, we thus make the following contributions:

- For the first time, we use the port Hamiltonian language in the training of reasonably *deep* neural networks in contrast to [79, 92] where networks are shallow.

- We also introduce an adaptive, goal oriented strategy for the control of the friction constant, which goes in the opposite direction as [8, 19, 21] but is well-motivated in terms of combining exploration and exploitation in one algorithm.

- We show experimentally for standard deep learning problems in image recognition that this strategy consistently produces improvements over fixed-parameter strategies. We also provide a considerable amount of ablation studies related to our parameter settings.

3.3 The Goal Oriented PHS Method

The simple gradient descent algorithm to minimize a differentiable loss function $\mathcal{L}(\omega)$, namely $\omega_{k+1} = \omega - \alpha \nabla_\omega \mathcal{L}(\omega)$ can be seen as a first order Euler discretization of the gradient flow

$$\dot{\omega}(t) = -\nabla_\omega \mathcal{L}(\omega), \quad \omega(0) = \omega_0. \tag{3.1}$$

It is well known that under adequate conditions on $\mathcal{L}(\omega)$, the flow $\omega(t)$ converges for $t \to \infty$ to a critical point ω^* with $\nabla_\omega \mathcal{L}(\omega^*) = 0$, see e.g. [79, 92]. Likewise, the gradient descent algorithm converges for $k \to \infty$ to a critical point, provided the step length α is suitably controlled, confer [6, 8].

As mentioned in the introduction, the problem with gradient descent in the context of highly non-convex loss functions $\mathcal{L}(\omega)$, as especially in the context of the training of deep neural networks [41], lies in the fact that gradient flows and gradient descent algorithms get stuck in local minima.

To overcome the strict locality of gradient flow and gradient descent, momentum based methods have been introduced. The update rule of gradient descent is changed to

$$\begin{aligned} \omega_{k+1} &= \omega_k + \alpha \frac{1}{m} p_k \\ p_{k+1} &= p_k - \alpha \frac{\gamma}{m} p_k - \alpha \nabla_\omega \mathcal{L}(\omega) \end{aligned} \tag{3.2}$$

where $m, \gamma > 0$ are parameters called mass and friction coefficient. p_k is the so-called momentum at iteration k. In fact, (3.2) can be understood as the discretized version of the following Hamiltonian set of equations

$$\begin{aligned} \dot{\omega}(t) &= \frac{1}{m} p(t) \\ \dot{p}(t) &= -\frac{\gamma}{m} p(t) - \nabla_\omega \mathcal{L}(\omega) \end{aligned} \tag{3.3}$$

with initial conditions $\omega(0) = \omega_0$ and $p(0) = p_0$.

To understand the global properties of the Hamiltonian dynamics, it is convenient to define a state variable $x(t) = \binom{\omega(t)}{p(t)}$ and the Hamiltonial function $H(x) = \frac{\|p\|^2}{2m} + \mathcal{L}(\omega)$ and the symplectic matrix $J = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$ as well as a symmetric, positive resistive matrix $J = \begin{pmatrix} 0 & 0 \\ 0 & \frac{\gamma}{m} \end{pmatrix}$ so that we can rewrite (3.3) in the compact, port-Hamiltonian form

$$\dot{x}(t) = (J - R) \nabla_x H(x). \tag{3.4}$$

Using the chain-rule, (3.4) and $\nabla_x H(x(\tau))^\top J \nabla_x H(x(\tau)) = 0$ by the skew-symmetry of J, it is now easy to see that the following inequality holds for the dissipated total 'energy' measured by $H(x)$, where $\frac{\|p\|^2}{2m}$ takes the role of kinetic energy and the loss $\mathcal{L}(\omega)$ the role of potential energy

$$H(x(t)) - H(x(0)) = -\int_0^t \nabla_x H(x(\tau))^\top R \nabla_x H(x(\tau)) \, \mathrm{d}\tau. \tag{3.5}$$

From this exposition it is intuitive, and in fact can be proven mathematically [6, 8], that due to dissipation the state $x(t)$ ultimately has to come to a rest, if $\mathcal{L}(\omega)$ is bounded from below. Thus, if the stationary points x^* with $\nabla_x H(x^*) = 0$ of the system are isolated, $x(t)$ will asymptotically converge to a stationary point. Furthermore, for $x^* = \begin{pmatrix} \omega^* \\ p^* \end{pmatrix}$, we find $p^* = 0$ and $\nabla_\omega \mathcal{L}(\omega^*) = 0$, hence the ω-component of stationary points are in one to one correspondence to the critical points of the original optimization problem.

Energy dissipation (3.5) thus is the key component that determines how fast $x(t)$ comes to rest, which conceptually is corresponding to convergence of the optimization algorithm. Apparently, the matrix R and thus the friction coefficient γ controls dissipation.

In fact, if $\gamma \approx 0$, essentially no energy is lost and the dynamics $x(t)$ will either move on for a very long time, or, in very rare cases, get to rest on a local maximum or saddle point. This perpetual motion through the accessible part of the 'phase space' can be seen as an exploitative strategy.

In contrast, if γ gets large, the friction essentially disperses energy and momentum and the motion of $x(t)$ behaves highly viscous, i.e. determined by the equality

$$-\frac{\gamma}{m} p(t) - \nabla_\omega \mathcal{L}(\omega) \approx 0 \quad \Leftrightarrow \quad \dot{\omega}(t) \approx -\frac{1}{\gamma} \nabla_\omega \mathcal{L}(\omega), \tag{3.6}$$

from which we see that in this high viscosity regime the port Hamiltonian flow essentially behaves like gradient descent (with a modified step length). Despite working with momentum, we are thus back in the exploitation phase of local minima.

The idea of this article is to use this physics based intuition to efficiently control the behavior of our port Hamiltonian optimization strategy in a goal oriented search. We thus propose to 'keep on moving' as long as we have not yet reached a predefined reduction of the initial loss function $\mathcal{L}(\omega_0)$. In many cases, it is known that $\mathcal{L}(\omega)$ is lower bounded by zero, and we can thus demand a 90%, 95% ... reduction in $\mathcal{L}(x(t))$, before we, upon reaching this target, instantaneously increase the value of γ in order to switch over from the low-viscous exploration phase to high-viscous exploitation. In this sense, our proposed optimization algorithm resembles the 'chicken game': who breaks too early, looses.

Before we come to the implementation and numerical tests of this strategy in deep learning, we discuss some peculiarities of the loss function in this case. We would like to learn a conditional probability density $p(y|x,\omega)$ from data independently sampled from the same distribution $\{(y_i,x_i)\}_{i=1}^n$, where x_i is some input and y_i takes values in some prescribed label space $\mathcal{C}=\{c_1,\dots,c_q\}$. In applications in image recognition, $p(y|x,\omega)$ often consists of several stacked convolutional and fully connected layers and an ultimate softmax layer, cf. [41]. The 'cross entropy'/negative log likelihood loss is given by

$$\mathcal{L}(\omega)=-\frac{1}{n}\sum_{i=1}^{n}\log p(y_i|x_i,\omega). \tag{3.7}$$

The numerical problem to implement (3.7) directly lies in the memory constraints that do not permit to load the entire data set $\{(y_i,x_i)\}_{i=1}^n$ in the memory. Therefore, mini batches $\mathcal{B}_j$, i.e. small random subsets of $\{1,\dots,n\}$ are drawn and an update step of the parameters ω_k and the associated momentum is executed for a loss $\mathcal{L}_{\mathcal{B}_j}(\omega)$ with the original data set replaced by $\{(y_i,x_i)\}_{i\in\mathcal{B}_j}$. Nevertheless, as in image classification oftentimes the batch $|\mathcal{B}_j|$ is quite large ($\gtrapprox 10$), $\mathcal{L}_{\mathcal{B}_j}(\omega)$ and $\mathcal{L}(\omega)$ tend do behave similar by the law of large numbers. In our numerical experiments, we therefore observe the behavior of the algorithm in accordance with intuition.

3.4 Experiments and results

For our experiments, we use a Convolutional Neural Net (CNN) similar to the Le-Net-5 [68] which consists of two convolutional, one pooling and two fully connected layers as it is shown in figure Fig. 3.2 and has a total of 44,426 weights. For implementation, we are using the PyTorch framework [89]. This network is chosen as it is a widely used standard architecture, although it is not eligible to compete with more sophisticated ResNet [50] or Transformer [121] architectures. Furthermore, in order to focus on training exclusively, the networks are trained from scratch on the data sets and we use neither pre-training nor augmentation. The training is performed with respect to the usual cross-entropy loss without regularization.

On the hardware-side, we use a workstation with an Intel(R) Core(TM) i7-6850K 3.6GHz and two Nvidia TITAN Xp graphic units with 12GB VRAM each for our experiments.

For a comparison with SGD and PHS, i.e. the traditional momentum method, we test our goal oriented PHS search on the two data sets CIFAR10 and FashionMNIST introduced above. We furthermore run trainings for a number of different

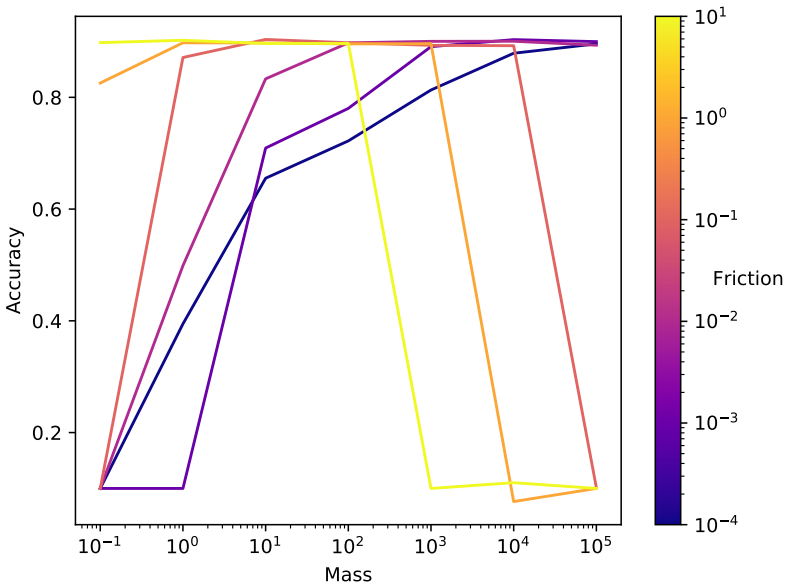


Figure 3.1: Selecting hyperparameters of learning rate (here: $\alpha = 0.1$), mass and friction based on the accuracy on the Fashion-MNIST dataset.

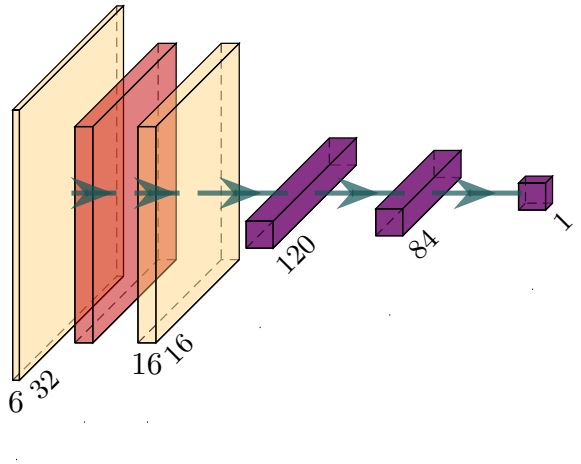
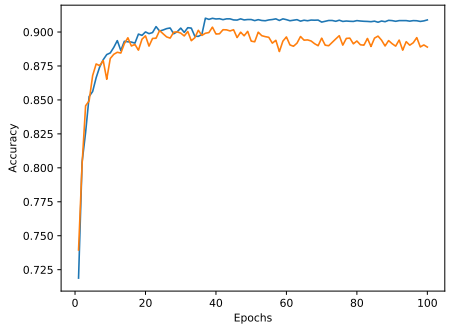
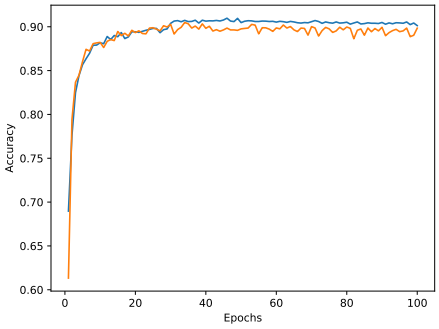


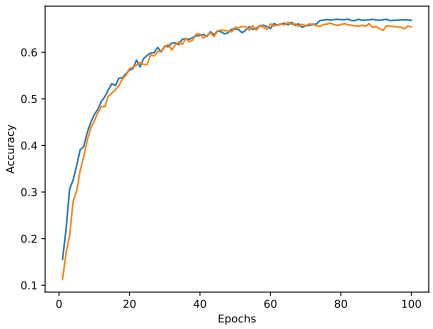
Figure 3.2: Neural Net architecture which is similar to Le-Net-5. Orange are convolutional layers with a filter size of 5, red is the pooling layer and fully connected layers are violet.



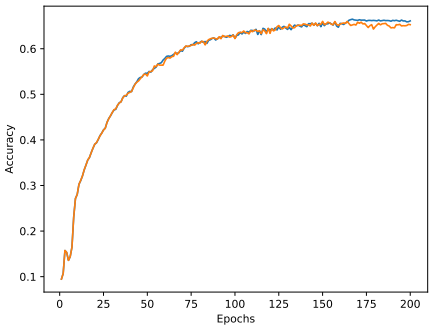
((a)) $\alpha = 0.1$, friction $= 0.1$, mass $= 10$ on Fashion-MNIST.



((b)) $\alpha = 0.01$, friction $= 1$, mass $= 0.1$ on Fashion-MNIST.



((c)) $\alpha = 0.1$, friction $= 0.1$, mass $= 100$ on CIFAR-10.



((d)) $\alpha = 0.01$, friction $= 0.1$, mass $= 25$ on CIFAR-10.

Figure 3.3: History of the accuracies over the epochs depending on the choosable hyperparameters learning rate α, friction and mass. PHS in orange, Goal-oriented approach in blue.

learning rates α and for several settings for the mass and baseline friction parameter. To establish which parameter settings are rewarding, we consider the accuracies of the PHS for different learning rates ($0.0001 \leq \alpha \leq 0.1$), that can be achieved when mass and friction are included. This is shown in Fig. 3.1 for the example of $\alpha = 0.1$ on the Fashion-MNIST dataset. As one can already see, the trainings for many parameter settings work significantly worse or not at all. Therefore, only experiments that lie in a parameter range leading to reasonable results are included in our result tables. Concerning goal orientation, we aim at a reduction of the initial loss of 65% to 90% and then increase the friction significantly by a factor between 5 and 99. The results are given in Table 3.1 for CIFAR10 and Table 3.2 for FashionMNIST.

As can be seen in Fig. 3.3(a), the accuracy of the method is consistently improved

α	Optimizer	Fric	Mass	Acc
0.1	SGD	/	/	64.82%
0.1	PHS	0.1	100	66.45%
0.1	Goal-Oriented (breaking at 0.65 with factor 49)	0.1	100	**67.1%**
0.1	PHS	0.01	100	63.52%
0.1	Goal-Oriented (breaking at 0.9 with factor 99)	0.01	100	65.52%
0.01	SGD	/	/	63.53%
0.01	PHS	0.1	25	66.01%
0.01	Goal-Oriented (breaking at 0.7 with factor 10)	0.1	25	**66.49%**
0.01	PHS	0.01	25	62.98%
0.01	Goal-Oriented (breaking at 0.7 with factor 50)	0.01	25	63.44%
0.001	SGD	/	/	65.05 %
0.001	PHS	1	0.25	66.0 %
0.001	Goal-Oriented (breaking at 0.7 with factor 20)	1	0.25	**66.37%**
0.001	PHS	0.1	0.25	62.93%
0.001	Goal-Oriented (breaking at 0.85 with factor 50)	0.1	0.25	63.54%
0.0001	SGD	/	/	64.43%
0.0001	PHS	10	0.001	65.76%
0.0001	Goal-Oriented (breaking at 0.68 with factor 5)	10	0.001	**66.39%**
0.0001	PHS	1	0.001	62.24%
0.0001	Goal-Oriented (breaking at 0.8 with factor 100)	1	0.001	63.56%

Table 3.1: Comparison of training results with SGD, PHS and Goal-Oriented approaches for the CIFAR-10 dataset.

α	Optimizer	Fric	Mass	Acc
0.1	SGD	/	/	90.04%
0.1	PHS	0.1	10	90.36%
0.1	Goal-Oriented (breaking at 0.15 with factor 50)	0.1	10	**91.02%**
0.1	PHS	0.01	10	83.31%
0.1	Goal-Oriented (breaking at 0.55 with factor 20)	0.01	10	87.19%
0.01	SGD	/	/	90.26%
0.01	PHS	1	0.1	90.49%
0.01	Goal-Oriented (breaking at 0.2 with factor 10)	1	0.1	**90.98%**
0.01	PHS	0.1	0.1	83.28%
0.01	Goal-Oriented (breaking at 0.5 with factor 5)	0.1	0.1	86.47%
0.001	SGD	/	/	89.61%
0.001	PHS	10	0.01	90.34%
0.001	Goal-Oriented (breaking at 0.15 with factor 5)	10	0.01	**90.8%**
0.001	PHS	1	0.01	90.13%
0.001	Goal-Oriented (breaking at 0.17 with factor 50)	1	0.01	90.77%
0.0001	SGD	/	/	88.86%
0.0001	PHS	10	0.001	90.17%
0.0001	Goal-Oriented (breaking at 0.2 with factor 100)	10	0.001	**90.54%**
0.0001	PHS	1	0.001	89.6%
0.0001	Goal-Oriented (breaking at 0.185 with factor 100)	1	0.001	90.12%

Table 3.2: Comparison of training results with SGD, PHS and Goal-Oriented approaches for the FashionMNIST dataset.

by breaking after reaching the goal, and the subsequent occurrence of overfitting (as happens with the PHS) is avoided. The increase in test accuracy lies around and in many cases above 0.5% throughout parameter settings and the two data sets employed, as documented in Table 3.1 for CIFAR10 and Table Table 3.2 for FashionMNIST.

The history of the test accuracy over the iteration count of the optimization procedure is shown in Fig. 3.3 for two example configurations of each dataset. As we observe, the sudden 'breaking' exploits a local minimum better and avoids overfitting (as it can be especially seen in Fig. 3.3(a)), i.e. the decrease of the ordinary PHS method in the further pursuit of the optimization. Interestingly, this hints that overfitting rather is a 'global' phenomenon associated with ongoing exploration, whereas exploitation of the local minimum seems less beset from overfitting issues. This is consistent with our observation that the training loss after 'breaking' quickly converges, whereas the training loss for SGD or PHS is further reduced. This suggests that the onset of overfitting could thus also be a useful triggering event for 'breaking' instead of goal orientation, as employed here.

3.5 Discussion and Outlook

In this work, we have introduced a new goal oriented strategy for the training of deep neural networks. By the physics-motivated interpretation of momentum in a port Hamiltonian framework, we explained how different settings for the friction / dissipation correspond to an exploration or exploitation phase in the progress of optimization. By switching from exploration to exploitation when a certain minimal reduction of the loss function of a deep neural network is achieved, we obtain improved classification accuracy of image classification networks as compared with simple stochastic gradient descent or a momentum based optimization with fixed friction.

The outlined strategy can be extended in several ways. First, for the case where the minimal reduction is never achieved for a long time, the exploitation phase could be executed nevertheless starting from the best parameter setting found so far, or the target could be adjusted. This will robustify our algorithm. Second, after a first exploitation phase, a re-acceleration could be executed, e.g. by an external force or 'port', so that multiple promising local minima can be visited.

Chapter 4

Meta-Learning-Algorithms

4.1 Definitions

What are Meta Learning Algorithms?

There are two types of algorithms that can be seen as Meta-Learning-Algorithms: The first possibility is that not only the weights of the network are optimized with regard to a specific data set, but also that some hyperparameters themselves are trained. If we now consider the network architecture itself as a hyperparameter that can be optimized, we find ourselves in the field of Neural Architecture Search (NAS). A second possibility to define a meta-learning algorithm would be to combine predictions that already result from machine learning methods with another machine learning method to form an overall prediction. In this case we focus on the first case, especially on neural architecture search in a specific searchspace.

4.2 Relevance of Meta Learning

At present, machine learning is a key technology for data-driven automation that achieved tremendous success in many applications, such as image recognition [107] and natural language processing [83].

However, the application of machine learning to ever new fields requires data scientists who in turn need to acquire the respective domain knowledge. In particular when working with deep learning [41], the question of how to choose hyperparameters and network architectures typically requires expert knowledge and a lot of engineering work. Automated Machine Learning (AutoML) [23, 51] is a research

area that aims at performing hyperparameter optimization [127] as well as neural architecture search (NAS) [33].

Many of the existing methods in the field of NAS modify existing network architectures [51]. This is due to the fact that the application of deep learning to new domains is not the only focus of this research area. For instance, runtime optimization of neural networks is a motivation for many of the works in the field of NAS [57]. Another field closely related to NAS is the task of network pruning [52, 97] where an existing network is made sparser in order to reduce the computational burden. More drastically, complete layers are removed in [76, 116].

4.3 Related work

AutoML is an active field of research.

While the survey [33] approaches NAS focussing on its different structural aspects, the survey [51] provides an overview of NAS approaches, providing a general and comprehensive view on the field of AutoML. The presented methods are classified into different categories, such as reinforcement learning, evolution-based algorithms, as well as gradient descent, random search and surrogate model-based optimization. Regarding NAS and its performance on image classification tasks, they compare the different methods especially with respect to their accuracy on the CIFAR10 and ImageNet datasets.

NAS and Reinforcement Learning. The authors of [131] present an approach where they use reinforcement learning in order to predict hyperparameters like the number of filters, filter height/width and stride for every layer up to a chosen maximum of layers using a recurrent neural network as meta model. Another reinforcement learning approach for NAS is introduced in [132] where the authors develop the NASNet search space, in which for a given problem (here CIFAR10 and ImageNet) a certain number of pooling layers are given, between which any number of feature map size-preserving blocks can be inserted. This search space is then iterated using a controller recurrent neural network to design a problem-specific architecture, while our search strategy on the ResBuilder is simpler and based on a penalization approach.

Penalization-based NAS. Our ResBuilder approach is also very related to MorphNet [42] which optimizes the number of channels of convolutional layers. A group Lasso regularization term penalizes the weights, such that channels with weights below a chosen threshold are removed. The threshold and the penalization are chosen such that the computational cost is below a given budget. In an iterative fashion, this step alternates with an expansion step wherein remaining computational budget is re-distributed proportionally to the different layers of

the network. Since MorphNet has an important part in the ResBuilder method, we will take a closer look at it in Sec. 4.4.2.1.

Similar to the regularization terms of BatchNormalization in MorphNet, gatekeeper variables are introduced in [122] that are multiplied by the output of each channel of each layer. The peculiarity here is that the actual weights are not trained during the process of finding the gatekeeper variables, but are still in the state of their random initialization, whereby the importance of the layers is calculated before the actual training of the weights starts. To keep the gatekeeper variables low, an additional regularization term is introduced depending on the cost of the layer and the status of the gatekeeper variables.

Another penalization-based approach regarding NAS can be seen in [31] where the authors present their Transformable Architecture Search (TAS). Instead of the approach of alternating training and pruning of a network architecture, the authors consider the possibility of training a large network with an excessive number of weights and then transferring the knowledge learned to another network for which the depth and width have been determined independently of the first network. The loss function also contains a penalty term that penalizes the size of the resulting network.

NAS and ResNets. ResBuilder works on the search space of ResNet architectures [50]. In [4], the authors also focus on ResNets, in particular the links between the different residual blocks, and define masks such that (in case of ResNet) the input of each residual block can take the outputs of every previous residual block.

The authors of [37] have a similar idea of a modular architecture search space with residual structures as we do, but develop an active-learning approach "incremental neural architecture search (iNAS)", which can be combined with any query strategy. They then interpret their search space as a directed acyclic graph in which they start with the smallest possible architecture in order to find a suitable, problem-specific residual architecture.

4.4 ResBuilder

4.4.1 Motivation

In this section we introduce our method ResBuilder [18] that tackles the problem of constructing ResNet [50] architectures from scratch. The goal is to find a ResNet architecture in an automated way that achieves high accuracy for a given problem while not exceeding a predefined computational budget.

More precisely, we utilize MorphNet [42] as a baseline which performs channel pruning as well as layer removal during training. We introduce a method to add and remove layers during training, dynamically controlling the network's capacity while balancing test accuracy and computational expense.

To achieve this, we focus on ResNet blocks that support layer insertion and removal during training in a natural way. Due to the skip connection, layers with weights close to zero almost act as identity layers. Such layers can thus be inserted during training without undoing the previous training progress. Similarly, layers with weights small in magnitude can be seamlessly removed during training with undoing previous progress. During architecture search, we utilize a layer LASSO approach in order to identify unnecessary layers of the parameters.

We demonstrate the efficiency of ResBuilder on six datasets, namely Animals10 [1], CIFAR10 [64], CIFAR100 [64], MNIST [69], FashionMNIST [126], EMNIST [25] (as they are also introduced in Sec. 2.3), and study its hyperparameters in-depth.

It turns out that ResBuilder easily builds ResNet architectures achieving close to state-of-the-art performance (without pre-training) on a variety of image classification benchmarks while saving computational cost compared to off-the-shelf ResNets.

4.4.2 Method

ResBuilder operates on the search space of ResNet architectures[50]. It modulates a given ResNet by optimally dropping layers while randomly inserting layers. At the same time, it is able to shrink and expand the number of filters per layer. For the latter, we utilize MorphNet which relies on a group Lasso regularization term, where the groups refer to weights corresponding to a given channel, as outlined in the previous section. We now introduce the MorphNet algorithm [42] and afterwards wrap our architecture search method around it. Our notation is also inspired by [42].

For a given image of size $H \times W$, let $F : \mathbb{R}^{H \cdot W} \to \mathbb{R}^1, H, W \in \mathbb{N}$ be a network that assigns class labels to Images:

$$F = \mathrm{SM} \circ \mathrm{FC} \circ f \circ \mathrm{CL}$$

for CL a convolutional, FC a fully connected and SM a softmax layer as well as a backbone $f : \mathbb{R}^{d_1} \to \mathbb{R}^{d_{2L}}$, $d_1, d_{2L} \in \mathbb{N}$, $L \in \mathbb{N}$, which consists of $\ell = 1, \ldots, L$ ResNet blocks B_ℓ, i.e.,

$$f = B_L \circ B_{L-1} \circ \ldots \circ B_1$$

It should be noted that also pooling layers can be part of the architecture, which are positioned in between of two residual blocks of convolutional layers. However, these are omitted from the notation for the sake of better readability.

Although our method, in principle is able to handle residual blocks of other sizes, as long as not stated otherwise we use residual blocks in our method that contain two convolutional layers, as they are used in smaller types of residual networks (e.g. ResNet18, ResNet34[50]) and are shown as the green block of layers in Fig. 4.1(b). Because of this we consider that each $B_\ell : \mathbb{R}^{d_{2\ell-2}} \to \mathbb{R}^{d_\ell}$ is composed of two convolutional layers with weight tensors ω_j, $j \in \{2\ell-1, 2\ell\}$. For $x_{\ell-1} \in \mathbb{R}^{d_{\ell-1}}$, the operations of these layers can be described as

$$x_\ell = x_{\ell-1} + \sigma(BN(\omega_{2\ell} \cdot \sigma(BN(\omega_{2\ell-1} \cdot x_{\ell-1})))) \tag{4.1}$$

where σ is the ReLU activation $\sigma(t) = \max\{0, t\}$ and BN is the batch normalization process:

$$BN(z_{i,j,\cdot}) = (\frac{z_{i,j,\cdot} - m_\ell(z)}{s_\ell(z)})\gamma_\ell + \beta_\ell, z \in \mathbb{R}^{u_\ell \times v_\ell \times c_\ell}$$
$$\forall i \in \{1, \ldots, u_\ell\}, \forall j \in \{1, \ldots, v_\ell\} \forall \ell = 1, \ldots, 2L$$

with $\gamma_\ell, \beta_\ell \in \mathbb{R}^{c_l}$ and the mean (standard deviation) $m_\ell(z)$ ($s_\ell(z)$) of the tensor z alongside its channels c_ℓ. When BN is applied to the tensor z, it is done by applying it to each $z_{i,j,\cdot} \forall i, j$ separately.

Any of the $j = 1, \ldots, 2L$ convolutions maps from $\mathbb{R}^{d_j}$ to $\mathbb{R}^{d_j+1}$, where the dimension is the product $d_j = u_j v_j c_j$ of the spatial extents u_j, v_j and the number of channels c_j. The application of the jth convolution requires

$$C_j = 2 s_j u_j v_j c_j c_{j+1}$$

floating point operations, where $s_j = \text{size}(\omega_j)/(c_j c_{j+1})$ denotes the filter size of the convolution's kernel.

4.4.2.1 Googles MorphNet

The MorphNet regularization term $\mathcal{G}_M(j)$ for a given layer j is defined as:

$$\begin{aligned} \mathcal{G}_M(\omega, j) = C_j \sum_{p=1}^{c_{j-1}} |\gamma_{j-1,p}| \sum_{k=1}^{c_j} \mathbb{1}^O_{\{\gamma_{j,\cdot} > \tau_M\}} + \\ C_j \sum_{p=1}^{c_j} \mathbb{1}^I_{\{\gamma_{j-1} > \tau_M\}} \sum_{k=1}^{c_{j-1}} |\gamma_{j,p}| \end{aligned} \tag{4.2}$$

with $\mathbb{1}^I$ ($\mathbb{1}^O$) the indicator function for a given statement on the input (output) channels. For further information on how the MorphNet regularization term is calculated have a look in [42]. The overall MorphNet regularization term $\mathcal{G}_M$ is then defined by:

$$\mathcal{G}_M = \sum_{\ell=1}^{2L} \mathcal{G}_M(\omega, \ell) \tag{4.3}$$

The MorphNet algorithm (presented in [42]) can then be stated as we can see in Script 4.1, where c'_ℓ denotes the channel width of Layer ℓ, $\mathcal{F}$ is in our case the number of FLOPs the net currently uses and ζ the maximum capacity of FLOPs the net should have.

Script 4.1: The MorphNet algorithm

1 Train the network to find

$$\omega^* = \underset{\omega}{\operatorname{argmin}}\{\mathcal{L}(\omega) + \lambda_M \mathcal{G}_M(\omega)\}$$

for suitable λ_M.
2 Find the new widths $c'_{1:2L}$ induced by ω^*.
3 Find the largest κ, such that $\mathcal{F}(\kappa \cdot c'_{1:2L}) \leq \zeta$.
4 Repeat from Step 1 for as many times as desired, setting $c^0_{1:2L} = \kappa \cdot c'_{1:2L}$.

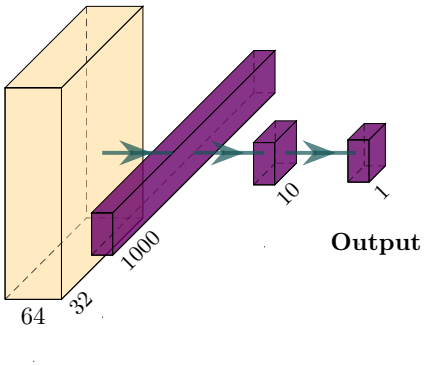
4.4.2.2 LayerLasso Basics

Our LayerLasso method uses a $L1$-regularization term $\mathcal{G}_\Lambda$, but it is based on regularization of the weights ω itself:

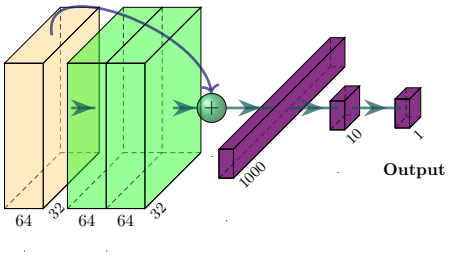
$$\mathcal{G}_\Lambda = \sum_{\ell=1}^{2L} \|\omega_\ell\|_1 \tag{4.4}$$

It should be noted that this regularization term is only applied to layers in the residual blocks and does not penalize the weights of the initial layer or the fully connected layers.

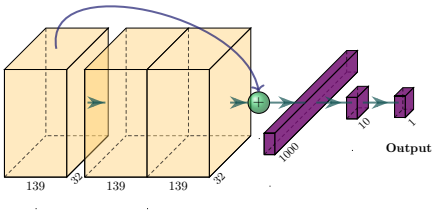
Besides the regularization terms our loss function also consists of a cross entropy loss $\mathcal{L}_{\text{CE}}$ as defined in (2.9) ($\mathcal{L}_{\text{CE}} = -\sum_i^q y_i \log F_i(x)$).



((a)) An empty start net with only one convolutional layer.



((b)) The net from 4.1(a) after inserting a block of two convolutional layers (green).



((c)) The net from Fig. 4.1(b) after one MorphNet iteration.

Figure 4.1: Example of structural changes the *ResBuilder* method uses.

4.4.2.3 Complete ResBuilder

In summary, our ResBuilder loss function consists of three different parts:

- The weight-optimizing part $\mathcal{L}_{\text{CE}}$ including a default L2-regularization term
- The MorphNet-L1-regularization $\mathcal{G}_M$ multiplied by the MorphNet regularization strength λ_M
- The LayerLasso-L1-regularization $\mathcal{G}_\Lambda$ multiplied by the MorphNet regularization strength λ_Λ

The complete loss term $\mathcal{L}$ can then be defined as:

$$\mathcal{L} = \mathcal{L}_{\text{CE}} + \lambda_\Lambda \mathcal{G}_\Lambda + \lambda_M \mathcal{G}_M \tag{4.5}$$

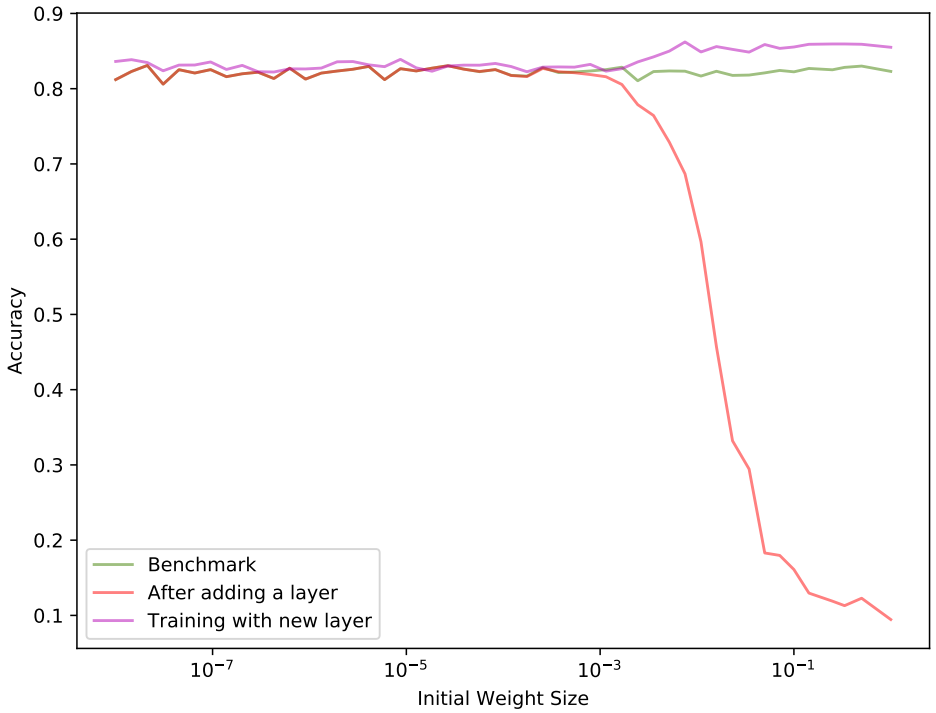


Figure 4.2: Effect of weight initialization of an inserted layer block to a network on the CIFAR10 dataset. The green line indicates the benchmark of the startnet trained normally without a new block of layers inserted. Red indicates the accuracy after adding one of these blocks to our startnet but without further training while purple shows the accuracy after further training with the new layerblock inserted.

4.4.3 Inserting ResNet blocks

Due to the construction of ResNet blocks, in particular their identity given by the addition of $x^{(B_i)}$, it is natural to initialize $\omega^{(B_i,j)}$, $j = 1, 2$, close to zero (but randomly) in order to enable a rational continuation of training from the current state of the network. Initialization close to zero implies $x^{(B_i+1)} \approx x^{(B_i)}$. Randomness is required to avoid symmetries in the weights that occur in case of constant initialization.

In order to obtain the most effective tradeoff between an initialization close to 0 and the avoidance of undesired symmetries in the network's weights, we show in Fig. 4.2 how much the accuracy of the network suffers from the insertion of residual blocks with different initial weights, but also how strong the constraints of unbroken symmetries would be when continuing to train.

Therefore, to insert a layer block B_k behind a given block B_i into our network f according to our insertion strategy (see Subsec. 4.4.5) as it can be seen in Fig. 4.1. We choose its initial weights such that the full potential of the new block can be exploited but the damage to the current knowledge of the network remains minimal. In our example, we therefore initialize the weights with an average initial weight of the order of $\omega_{init} = 10^{-2}$. From this we get the new network architecture $f' = B_n \circ \ldots \circ B_{i+1} \circ B_k \circ B_i \circ \ldots \circ B_1$.

4.4.4 Layer removal by LayerLasso

In order to also have the possibility to delete residual blocks of layers from positions where the net does not use its capacity efficiently, we introduce the **LayerLasso**: After a certain number of epochs of training every block B_i that includes at least one layer whose sum of weights $\sum_{\omega_m \in \omega^{B_i,j}} \|\omega_m\|_1, j = 1, 2$ lays under the set threshold for layer deleting τ_Λ will be erased from the net:

$$\begin{aligned} f' = B_n \circ \ldots \circ B_{i+1} \circ B_{i-1} \circ \ldots \circ B_1 \\ \forall B_i \exists j : \sum_{\omega_m \in \omega^{B_i,j}} \|\omega_m\|_1 < \tau_\Lambda. \end{aligned} \tag{4.6}$$

Because of the residual architecture, a layer with small weights resembles the identity, so we continue training after the removal of blocks from the resulting architecture with the weights for the remaining blocks equal to the values before the removal step.

4.4.5 Strategy of inserting and removing

In order to optimize the resulting network architecture we used a "random in - greedy out" optimization strategy: We insert residual blocks of convolutional layers at random positions in the net with the only constraints that the new block can not be inserted before the first convolutional layer or after the flattening of the feature maps. In order to evenly spread the blocks across the entire architecture we choose the pooling stage randomly and then a random block B_i from this stage after which a new residual block is inserted (this might also be directly behind the pooling layer).

4.4.6 Structure of the method

Our method can be defined as follows:

Fig. 4.3 shows a visualization of our training pipeline. We do n_Λ insertion steps before we start the MorphNet subroutine. Besides the training with all regularization terms active

$\omega_{reg} := \text{argmin}\,\{\mathcal{L}_{\text{CE}}(\omega) + \lambda_M \mathcal{G}_M + \lambda_\Lambda \mathcal{G}_\Lambda\}$

that determines which $\mathcal{A}' \subset \mathcal{A}$ of the architecture search space $\mathcal{A}$ are considered, we also train the actual network architecture without regularization $\omega_{noReg} := \text{argmin}\,\{\mathcal{L}_{\text{CE}}\}$ in order to achieve the best accuracy for the given net.

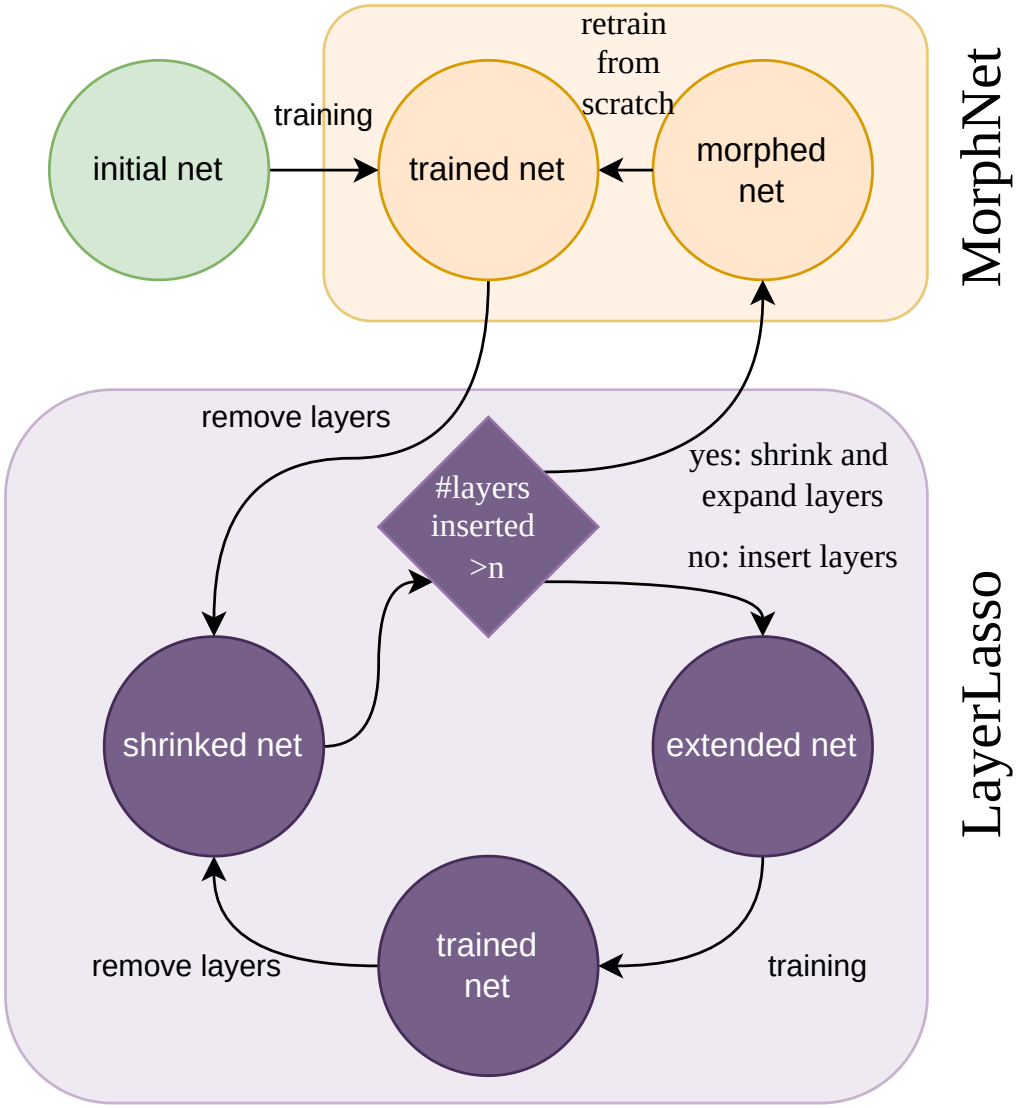


Figure 4.3: Overview of the method.

Chapter 5

Numerical results on MorphNet

In this chapter we have a look on some experiments which are executed solely with the MorphNet framework. Especially we have a closer look on how the regularization strength λ effects the tradeoff between accuracy and computation intensity. Furthermore, we are interested in whether the MorphNet algorithm is robust to bad initial layer widths.

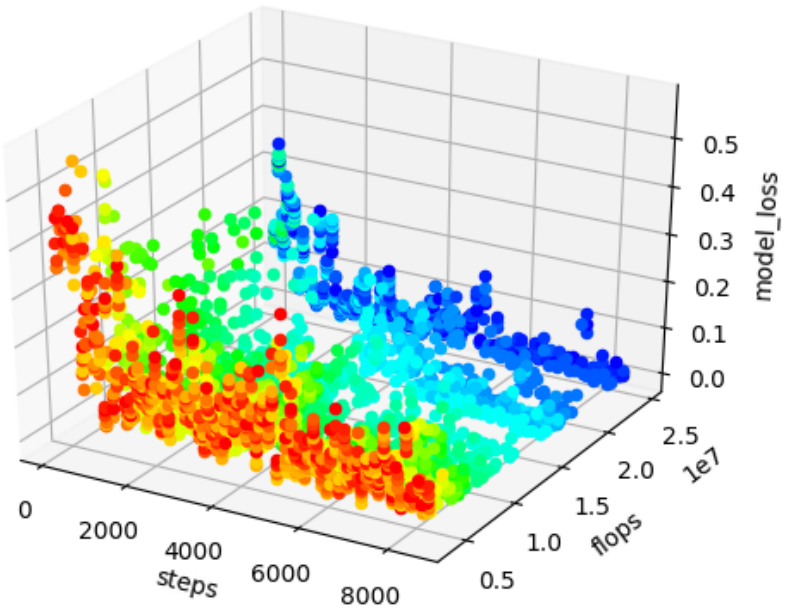
5.1 Tradeoff between Accuracy and FLOPs

This section deals with the effect of varying the regularization strength λ and the consequences for the net performance. Therefore, we have a look on the Paretofront between the accuracy the net achieves and the corresponding calculation intensity in form of floating point operations (FLOPs). All trainings have started with nets of architecture existing of one Convolutional layer with 32 channels and a second Convolutional Layer with 64 channels.

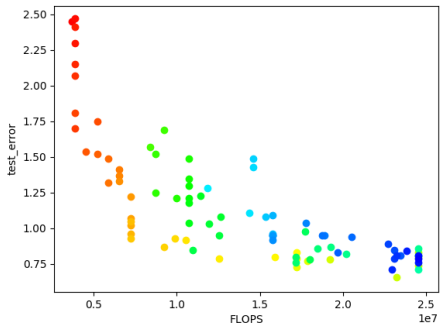
5.1.1 One single MorphNet Iteration

At first we have a closer look on performing one single MorphNet Iteration and then evaluate the suggested nets regarding the achieved accuracy and the used calculation intensity:

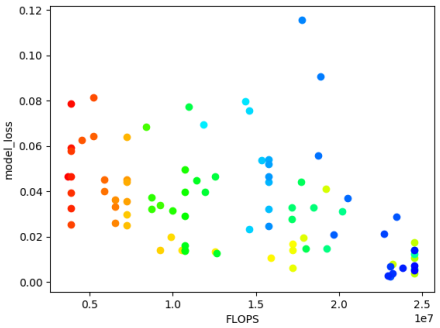
In Fig. 5.1(a) we can see some runs with different regularization strengths λ on the MNIST dataset. Red points represent a high regularization strength with its maximum value of $\lambda_{\max} = 9\text{e}^{-6}$. Blue dots show quiet small regularization strengths with dark blue dots with a minimum of $\lambda_{\min} = 1\text{e}^{-9}$.



((a)) Behavior of model loss (inclusive regularization) and FLOPs during training



((b)) Paretofront of the test error in relation to the architecture size



((c)) Paretofront of model loss (exclusive regularization loss) and FLOPs

Figure 5.1: Single MorphNet iteration with different regularization strengths

Fig. 5.1(a) shows the model loss (inclusive the regularization loss) in relation to the used FLOPs while training. As we can see, during the training the model loss decreases and the number of flops depends heavily on the regularization strength. To see how the regularization strength effects the relation between the test error and the calculation intensity at the end of such a training, we can have a look on the Paretofront shown in Fig. 5.1(b). Obviously trainings with a high regularization penalty have a higher test error but in terms of MorphNets calculation cost, these nets perform much better than runs with a low regularization term. This result is strengthened by Fig. 5.1(c) where we can see the models' loss (exclusive the regularization term) of the final net. Also here the performance of nets trained with a weak regularization strength is in general better than nets which suffered from a high value of λ.

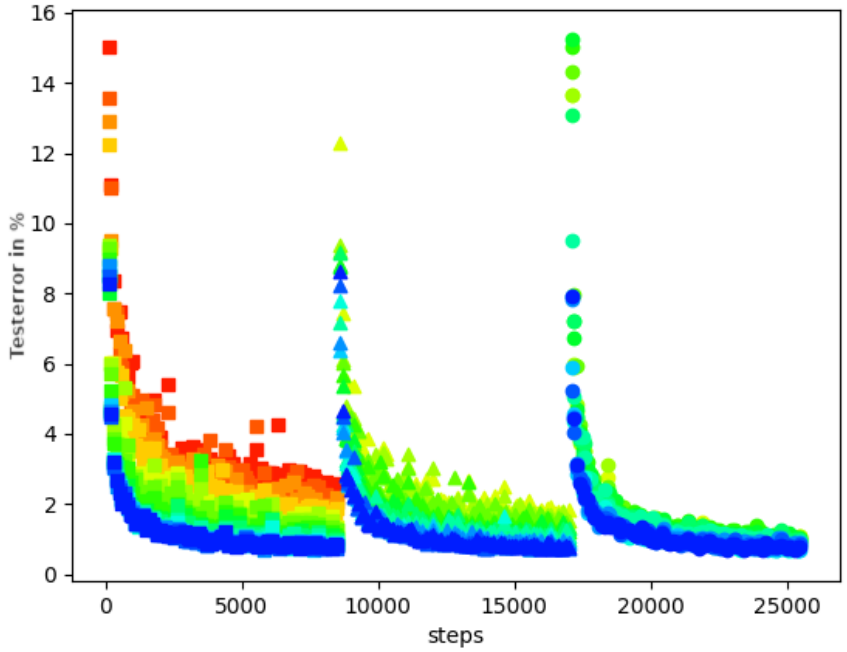
The "problem" of these results is the fact that they have been done on only one single MorphNet shrinkage step and the resulting nets are partly hard to retrain from scratch as the MorphNet algorithm suggests after the expanding routine. For example, the suggested net architecture of the highest regularization run ($\lambda_{\text{max}} = 9\text{e}^{-6}$) has been reduced from 32 channels in the first layer and 64 channels in the second layer to a size of 1/41 channels in the first/second layer, while the run with the lowest regularization strength ($\lambda_{\text{min}} = 1\text{e}^{-9}$) did not shrink the suggested net at all. So we now do multiple MorphNet steps, to see how these suggested nets behave when expanded and retrained from scratch:

5.1.2 Multiple MorphNet Iterations

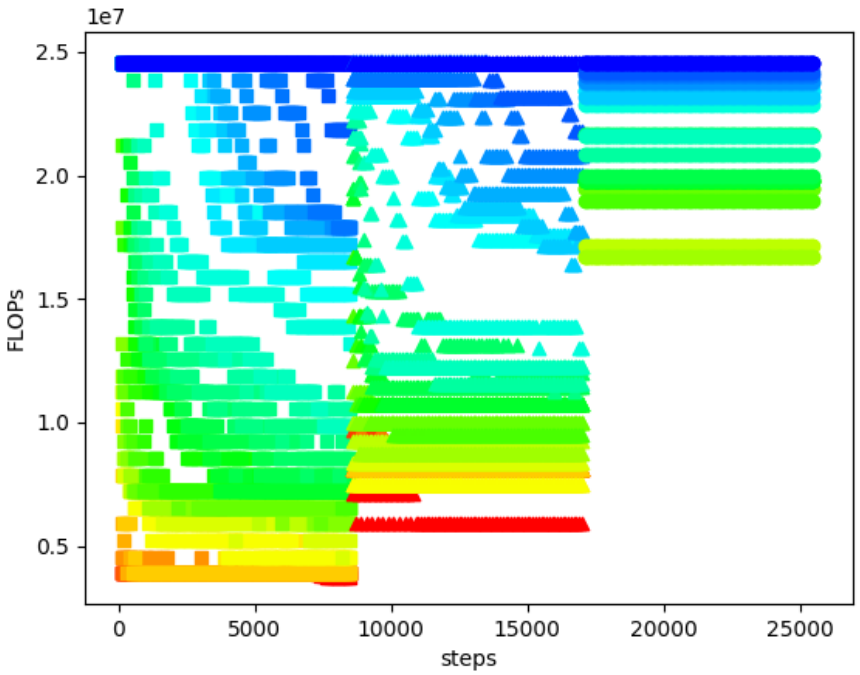
In this subsection, we run the MorphNet routine twice for different regularization strengths λ, and end each of these experimental runs by training the architecture suggested by MorphNet without a regularization term to achieve the best performance.

In Fig. 5.2 we have an overview of runs with different regularization strengths regarding its test error in Fig. 5.2(a) and its computational costs in Fig. 5.2(b). As in Fig. 5.1 the red markers identify runs with a high regularization strength ($\lambda_{\text{max}} = 9\text{e}^{-6}$) and the blue markers show runs with a low regularization strength ($\lambda_{\text{min}} = 1\text{e}^{-9}$). Here and in the following figures the squared markers tag values of the training while the first training with MorphNet, the triangles denote the second training with MorphNet and the circles indicate the final training without any regularization. After each of the two MorphNet trainings, the shrinkage and expanding routines are both performed and the new net is trained from scratch.

As we can see in Fig. 5.2(a), many runs with a high regularization strength are terminated after the first training. This happens because at least one layer of the suggested net has been broken down to zero Channels while training the second

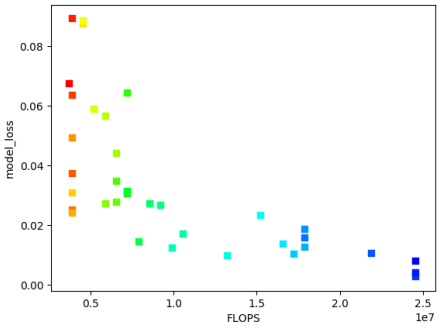


((a)) Testerror while training.

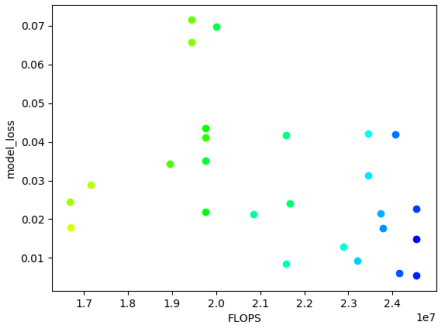


((b)) FLOPs while training.

Figure 5.2: Overview of trainings with different regularization strengths.



((a)) Paretofront of model loss and FLOPs after the first MorphNet iteration



((b)) Paretofront of model loss and FLOPs after final training

Figure 5.3: Two Paretofronts during the trainings-procedure

MorphNet iteration. So an evaluation of such a net would not be useful, because of what these runs do not show up on this and the following figures. For example the suggested net in the second MorphNet iteration with the maximum regularization strength $\lambda_{\max} = 9e^{-6}$ went down from 3/116 channels in the first/second convolutional layer [1] to 0/116 channels within 100 trainings steps.

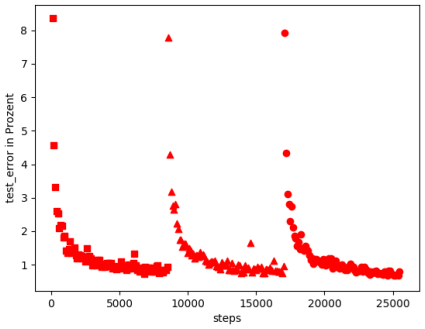
Also here it is interesting to see how the Paretofront of the model loss and the FLOPs changes over the different MorphNet iterations as shown in Fig. 5.3.

Fig. 5.4 shows the example run with the regularization strength $\lambda = 2e^{-8}$. In Fig. 5.4(a) we can see the classification error on the testdata while training. Fig. 5.4(c) shows the by MorphNet suggested architectures every 100 steps and in Fig. 5.4(b) we can see the corresponding FLOP costs of the suggested nets.

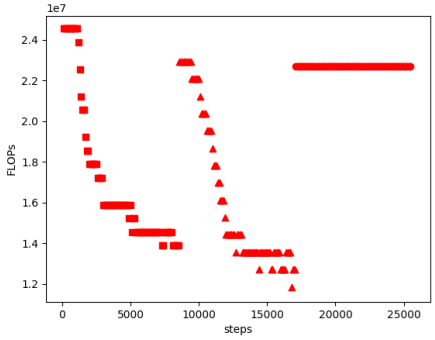
5.2 MorphNet's robustness to bad initial architectures

This section is about the question if the initial number of channels given in the startnet file has a longterm impact on the architecture suggestions of MorphNet. Therefore, we tested to train a startnet with two convolutional layers once with a large and once with a small number of channels per layer on the MNIST dataset, to see if both trainings will result in a similar result net.

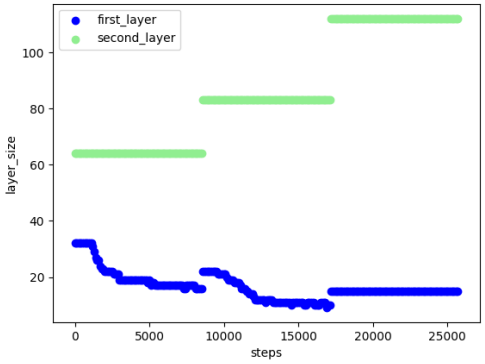
[1] The final layer sizes after the first MorphNet shrinkage have been 1/41 which have been blown up to 3/116 in the expanding step.



((a)) Testerror while training



((b)) FLOPs needed for suggested architecture while training



((c)) Suggested sizes of layers while training

Figure 5.4: Two MorphNet iterations with $\lambda = 2e^{-8}$ and a final training without regularization

As we can see in Fig. 5.5 the large architecture (red markers) instantly converges to the long term used architecture while the small architecture slowly expands in the expanding steps of the MorphNet algorithm but also has a very similar outcome architecture. In the Figure the crosses indent the first layer while the squares identify the second layer.

Therefore, for the small case where we train a two layer convolutional neural network on MNIST, we can say that the training converges to the same architecture after a sufficient number (about 20) of MorphNet iterations.

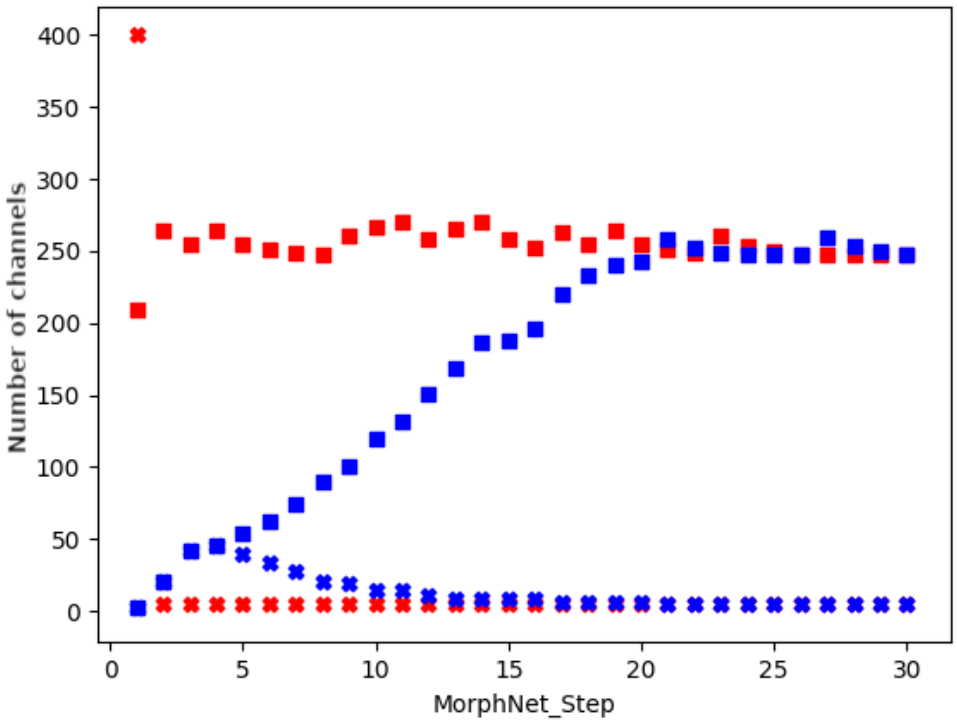


Figure 5.5: History of suggested MorphNet architectures initialized with bad architectures

Chapter 6

Numerical results on ResBuilder method

6.1 Experimental setup

We have shown the basics of our ResBuilder method in Chapter 4, and provided an overview of how it operates in Fig. 4.3. We also saw more detailed experiments on the topic of the MorphNet regularization strength λ_M in Sec. 5.1.1. Therefore, in this section we will start introducing and discussing the other adjustments like hyperparameters of the ResBuilder method.

After that we will have a look on some techniques mentioned in Chapter 2 applied to our ResBuilder approach.

Finally, we will present and discuss our main achievements with the ResBuilder approach.

6.1.1 Hyperparameter settings

6.1.1.1 Initial architectures

In our experiments, we apply our ResBuilder method to the different data sets. In one set of experiments, a ResNet18 (shown for a 32×32 pixel sized image in Fig. 6.2) is used as the initial network (RB-R18), and in the second set, a minimal network (RB-0Net), as shown in Fig. 6.1.

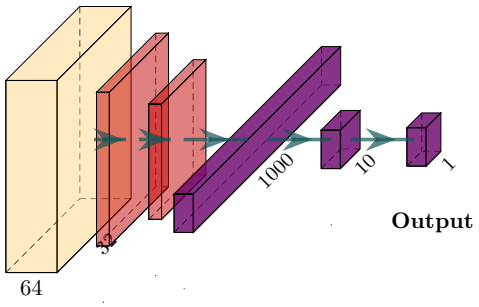


Figure 6.1: Minimal startnet with only one convolutional layer. Initial architecture for (RB-0Net).

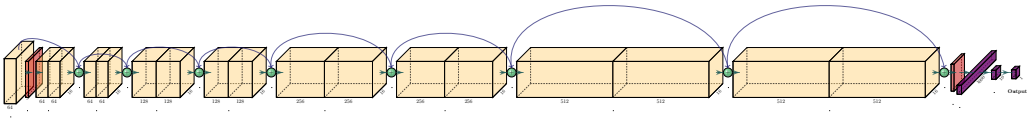


Figure 6.2: ResNet-18. Initial architecture for (RB-R18). In this case, the first pooling layer (red) is a max pooling layer and the second one is an average pooling layer like it is defined in [50].

6.1.1.2 Default parameters

Unless further specified, we use the default values given here for the hyperparameters:

- $n_\Lambda = 4$ number of insertion steps before MorphNet step.
- $n_M = 7$ maximum number of MorphNet steps.
- Set $\Theta_{LL} = True$ to activate LayerLasso-Momentum for only removing layer blocks if there was no improving by inserting new layers the last time.
- $\tau_{LL} = 0.015$ the threshold how much improvement there has to be in order to activate the LayerLasso-Momentum
- $\lambda_M = 10^{-7}$ the MorphNet regularization strength
- $I_M = 1 = 100\%$ intensity of using the MorphNet suggestions: The new channel widths after a MorphNet routine are then calculated as follows: $c_i^* = (1 - I_M) \cdot c_i + I_M \cdot c_i'$ with c_i the old layer width of layer i and c_i' the MorphNet suggested layer width.
- $\zeta = 100{,}000{,}000$ the aimed for FLOP costs, except for Animals10, for which ten times the amount of arithmetic operations was allowed ($\zeta =$

1,000,000,000), as the images in this dataset also have significantly higher resolution.

- $\lambda_\Lambda = 10^{-8}$ the LayerLasso regularization strength
- $\tau_\Lambda = 10^{-3}$ the threshold of our LayerLasso method which sets when a layer (block) is deleted (see Subsec. 4.4.4)
- $\lambda_0 = 10^{-5}$ the additional $L2$-regularization strength.
- *Adam* is used as optimizer.
- Data augmentation is active with 10% shifts in horizontal and vertical position as well as a possible horizontal flip.

6.1.2 Training types

The experiments we conduct contain three training types of each neural network architecture that is considered:

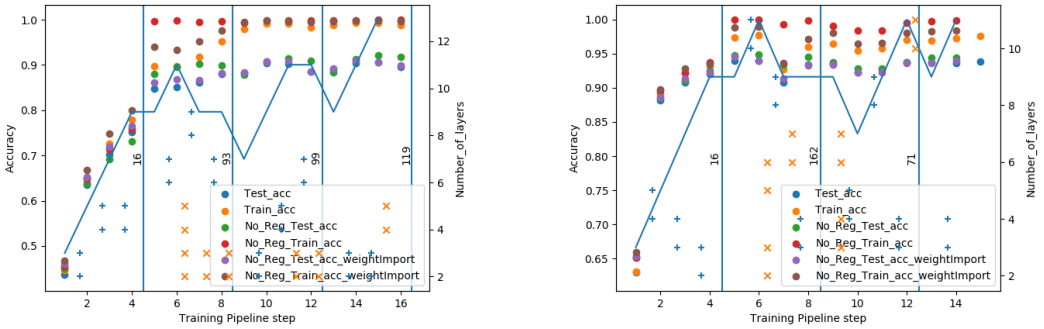
- Training Variant *With Reg*: With all possible regularization terms.
- Training Variant *No Reg RI*: Without additional regularization terms, starting from scratch, i.e. a random initialization of the weights.
- Training Variant *No Reg WI*: Without additional regularization terms, starting from the checkpoint induced by training *With Reg*.

The expression "without additional regularization terms" used here means that the training takes place without additional regularization by MorphNet or the LayerLasso, i.e. $\lambda_M = \lambda_\Lambda = 0$. An additional $L2$-regularization term λ_0, which is intended to reduce overfitting, is nevertheless applied to the training.

6.1.3 Explanations of evaluation figures

In this chapter there will be many figures like Fig. 6.3. As long as not stated otherwise, the individual components of the visualizations are as they are briefly described in this subsection:

The different architectures, the ResBuilder considers are traversed on the x-axis, whereas each unit of the x-axis represents one single architecture different from all the others, as we check while our insertion routine, that this particular architecture has not been trained in an earlier stage of the algorithm. The y-axis on the left side represents the accuracies of the different trainings for the plotted points, while the right y-axis indicates the size of the current architecture by the blue line and the plus signs (crosses) indicate at which depth of the neural network



((a)) LayerLasso Momentum on CIFAR10

((b)) LayerLasso Momentum on FashionMNIST

Figure 6.3: Demonstration of LayerLasso Momentum for two examples on different datasets.

architecture layers are added (eliminated). The vertical blue lines represent the respective MorphNet stages after which the nets were re-trained from scratch. The numbers next to the MorphNet lines show the maximum layer width of the first pooling stage of the architecture as an indicator for the width of the whole architecture.

6.1.4 LayerLasso Momentum

In order to ensure an efficient procedure when traversing the search space of architectures, we have built a momentum into the ResBuilder during the exploration, which causes the deletion routine of the ResBuilder to be skipped if an improvement of the achieved accuracy by a threshold value τ_{LL} has been achieved after the insertion of a layer block. This means that layers cannot be removed from the network even if they would actually be under the deleting threshold τ_Λ. This expands the search space of architectures before the optimization of the architecture by removing layers begins.

That can be seen in the two examples of Fig. 6.3, where we firstly tested this LayerLasso momentum. How to understand the figures is described in Subsec. 6.1.3. As can be seen, in both cases the accuracy continues to increase over the first MorphNet routine up to architecture 5, which is why in these cases no (blocks of) layers are deleted from the architecture. Only as soon as the accuracy stagnates (as here with architecture 6) it is possible for the ResBuilder algorithm to remove layers, whereby in both cases two blocks with two layers each are directly eliminated from the architecture.

6.1.5 Pre-processing the data

For the Animals10 and CIFAR10 datasets, we apply a normalization process to the data, which calculates for each pixel the difference between the current pixel value and the mean of all pixel values of all images in the current color channel, and then divides it by the standard deviation. MNIST and FashionMNIST pixel values are simply scaled from 0 to 1 and not pre-processed. We do also use data augmentation for our trainings, where we allow a shift of 10% in both dimensions and also horizontal flips.

6.1.6 Used Resources

We used different packages like Tensorflow-GPU (version 1.14.0) [3], Keras (version 2.2.4.) [22] or MorphNet (0.2.1) [42]. The visualization of neural nets like Fig. 6.1 is based on the repository of [58]. For a full list of used packages see the requirements.txt in our git repository.

For the calculations, we used a Dell Precision 7920 workstation with a Dual Intel Xeon Gold 6248R 3.0GHz and three Nvidia Quadro P 6000 graphic units with 24GB VRAM each.

6.2 Numerical results

6.2.1 Benchmarks

Dataset	Res18	MorphNet	RB-R18	RB-0Net
Animals10	92.10%	92.02%*	**92.73%***	88.72%*
CIFAR10	85.50%	88.17%	88.32%	**89.92%**
CIFAR100	53.80%	59.78%	57.69%	**62.36%**
MNIST	99.14%	99.11%	99.17%	**99.34%**
FashionMNIST	92.81%	92.97%	**93.55%**	93.71%
EMNIST	86.48%	86.70%	86.86%	**86.95%**

Table 6.1: Overview of achieved accuracies

In one of our experimental setups, we ran the ResBuilder method with the same hyperparameters (described in Sec. 6.1.1) for all our datasets under consideration and summarize these results in Table 6.1. The column with the results of the run "RB-0Net" gives the accuracies we achieve for the test series starting with a minimal network architecture as shown in Fig. 6.1, without regularization (*No*

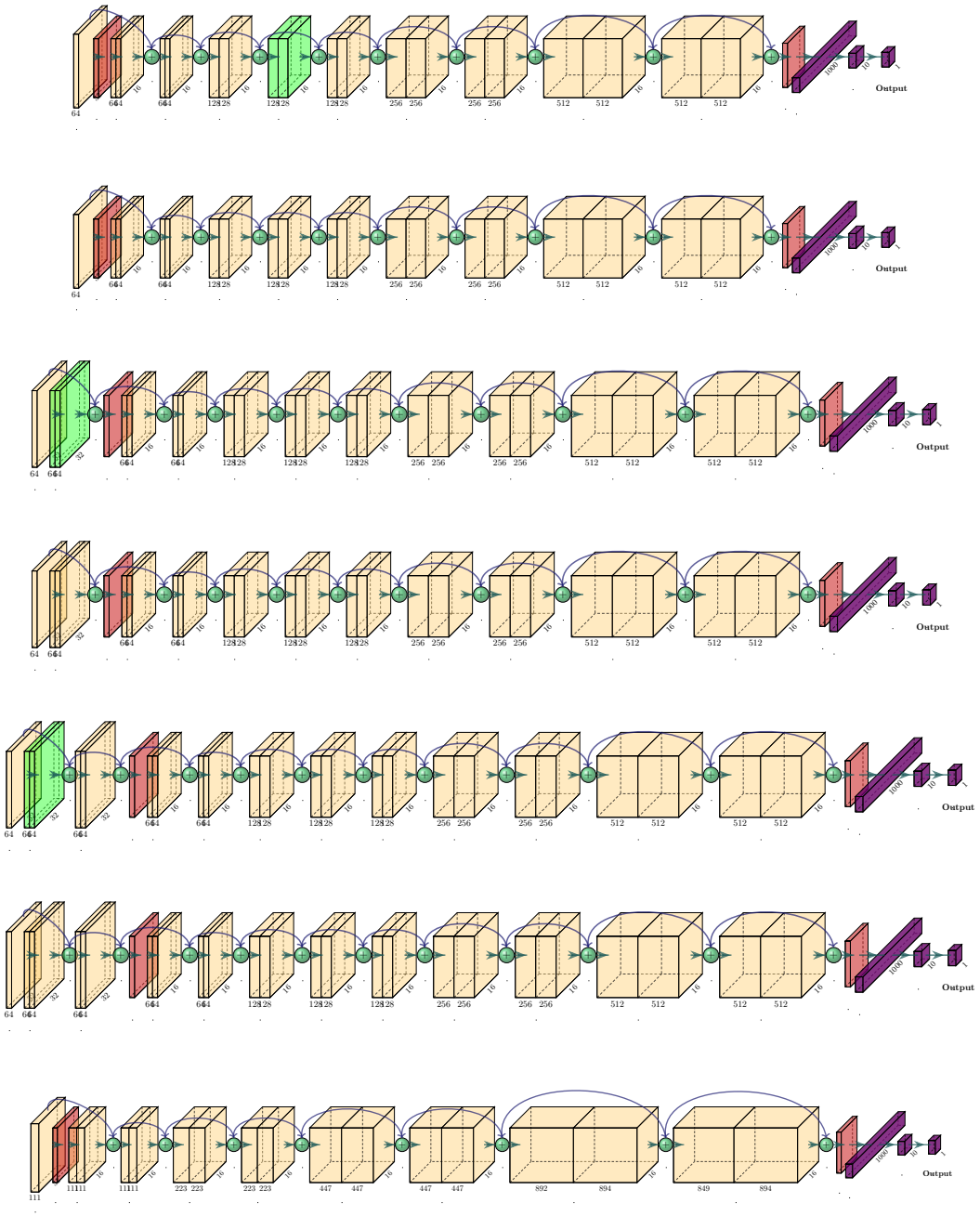


Figure 6.4: Progress of network architecture on CIFAR10 with ResNet18 as initial architecture - first morphing routine

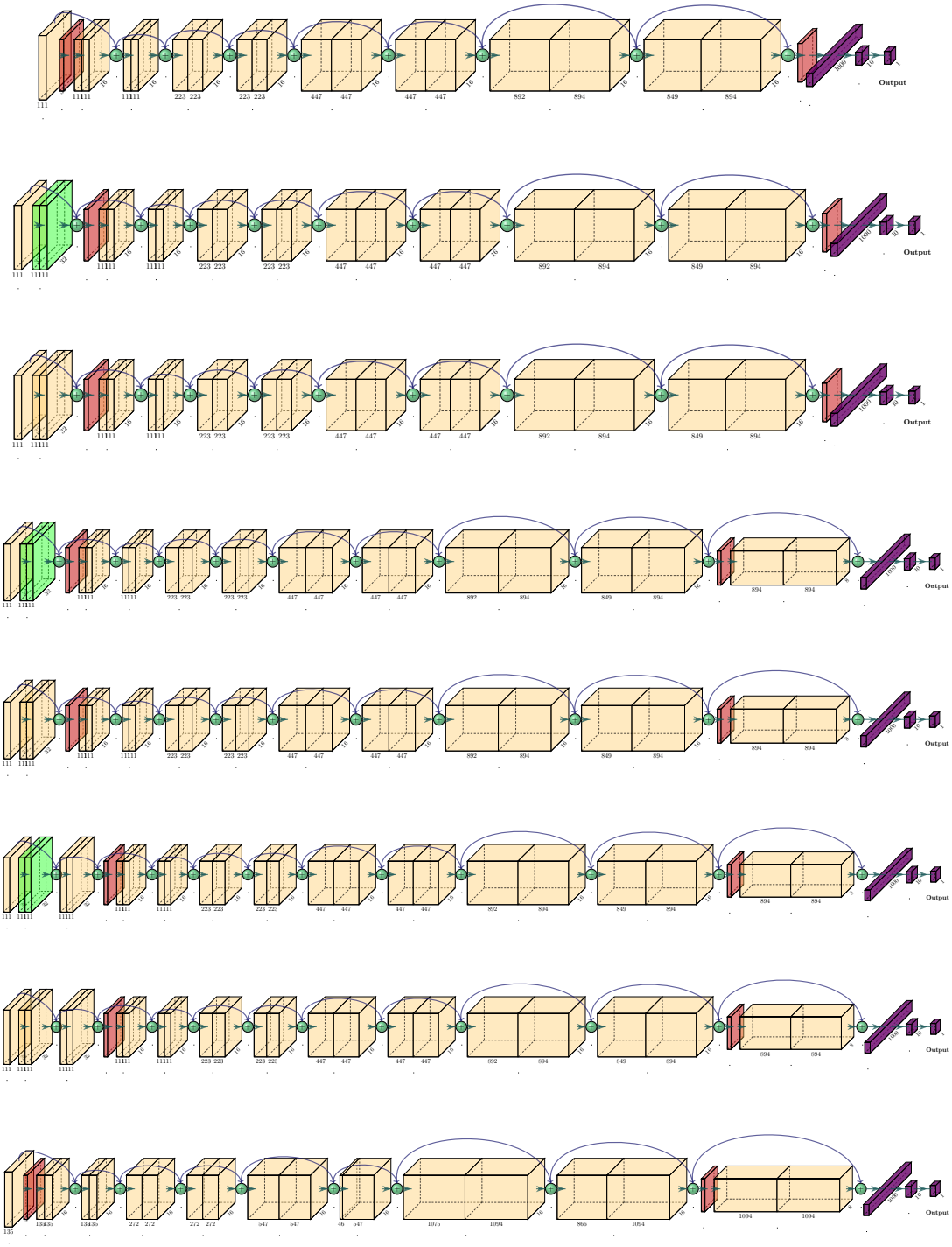


Figure 6.5: Progress of network architecture on CIFAR10 with ResNet18 as initial architecture - second morphing routine

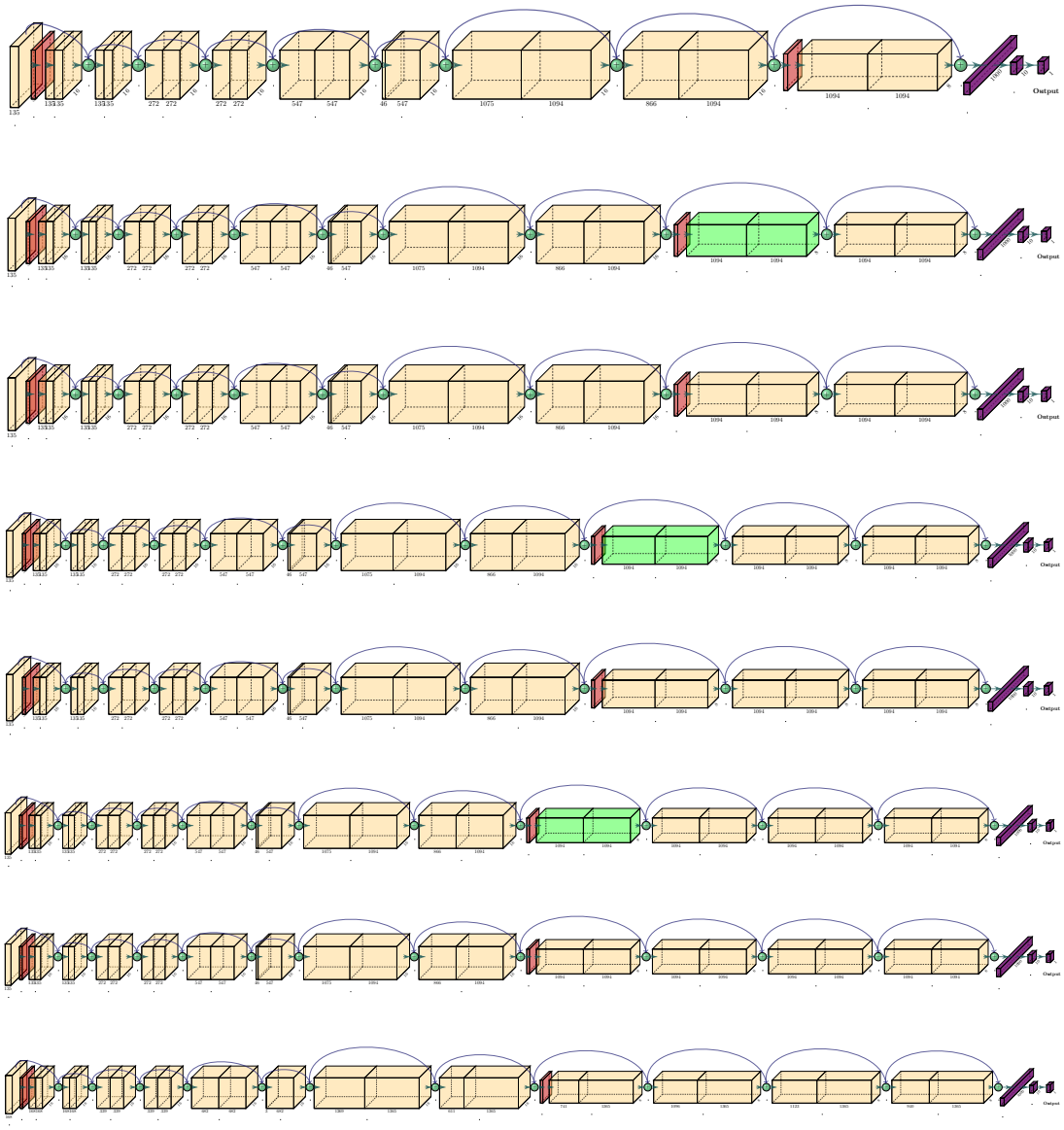


Figure 6.6: Progress of network architecture on CIFAR10 with ResNet18 as initial architecture - third morphing routine

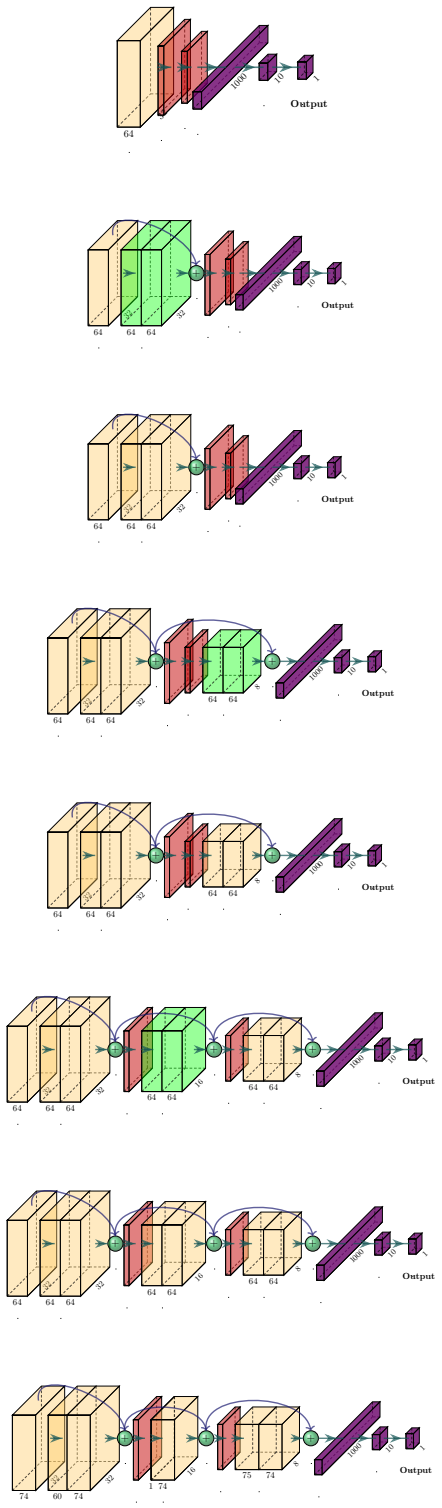


Figure 6.7: Progress of network architecture on CIFAR10 with the minimal network as initial architecture - first morphing routine

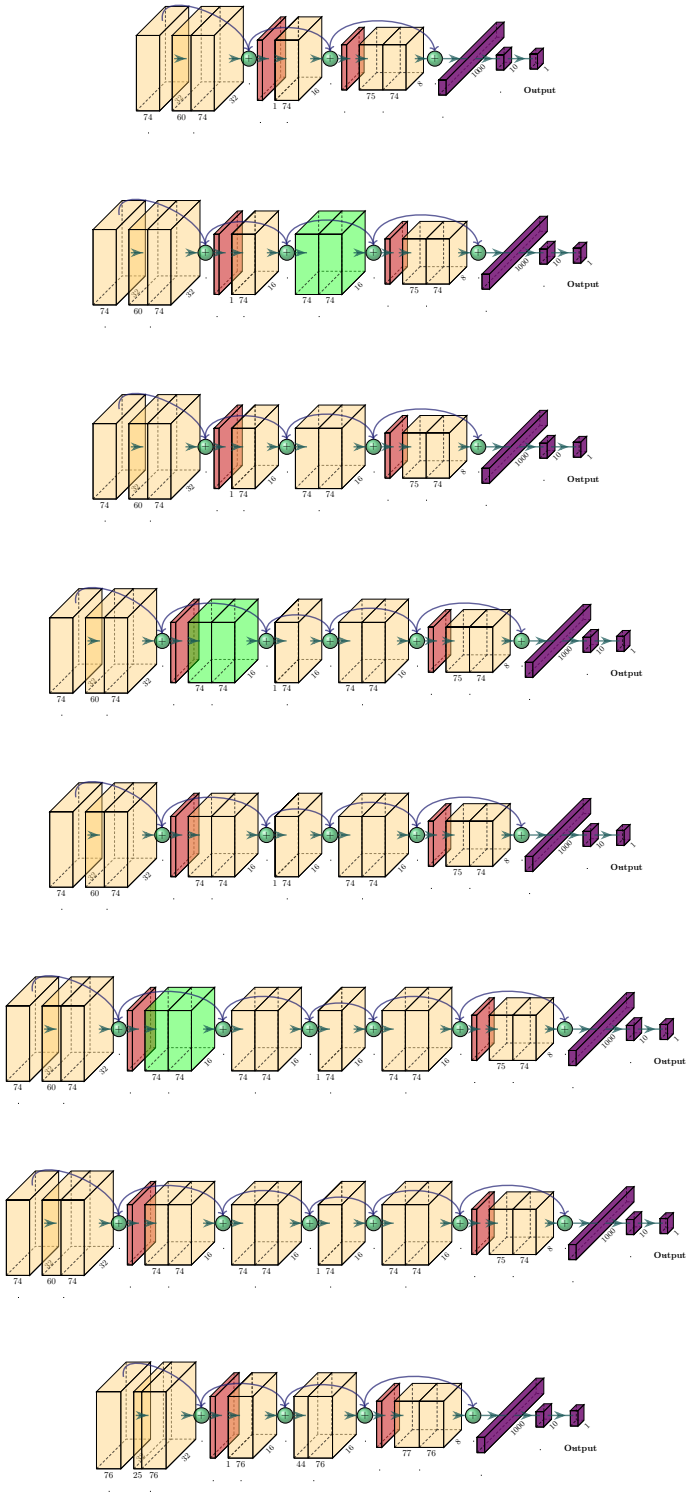


Figure 6.8: Progress of network architecture on CIFAR10 with the minimal network as initial architecture - second morphing routine

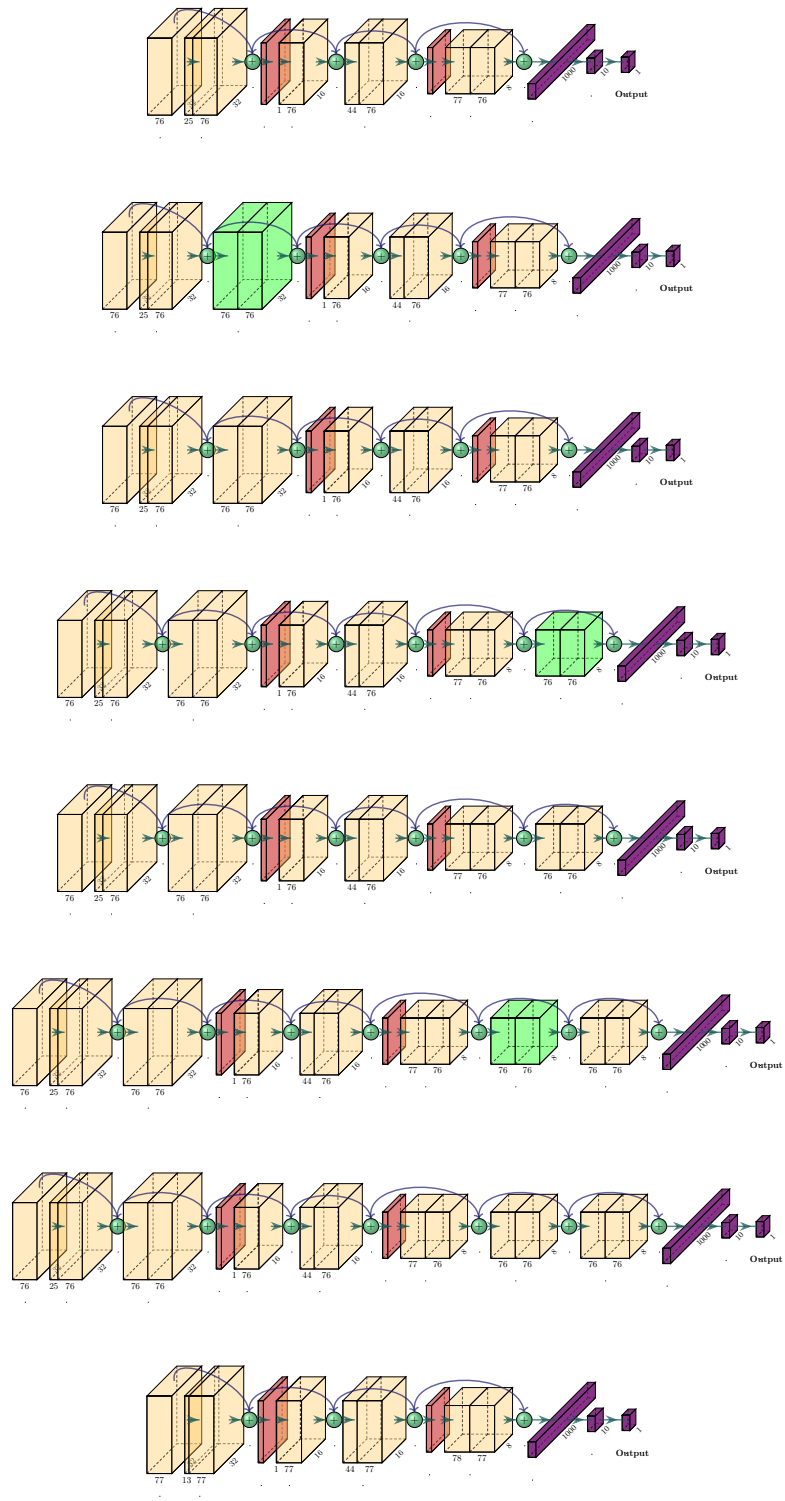


Figure 6.9: Progress of network architecture on CIFAR10 with the minimal network as initial architecture - third morphing routine

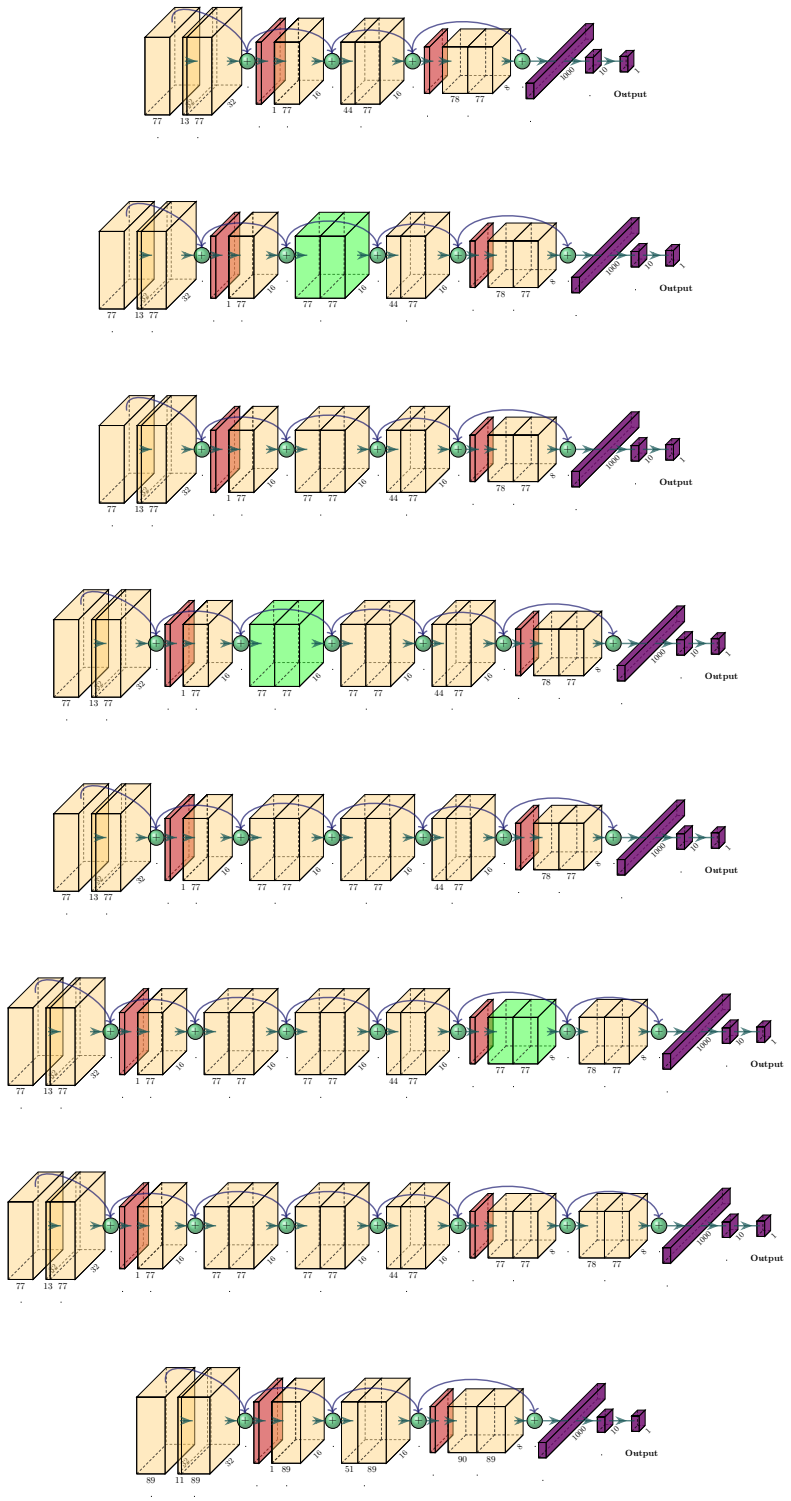


Figure 6.10: Progress of network architecture on CIFAR10 with the minimal network as initial architecture - fourth morphing routine

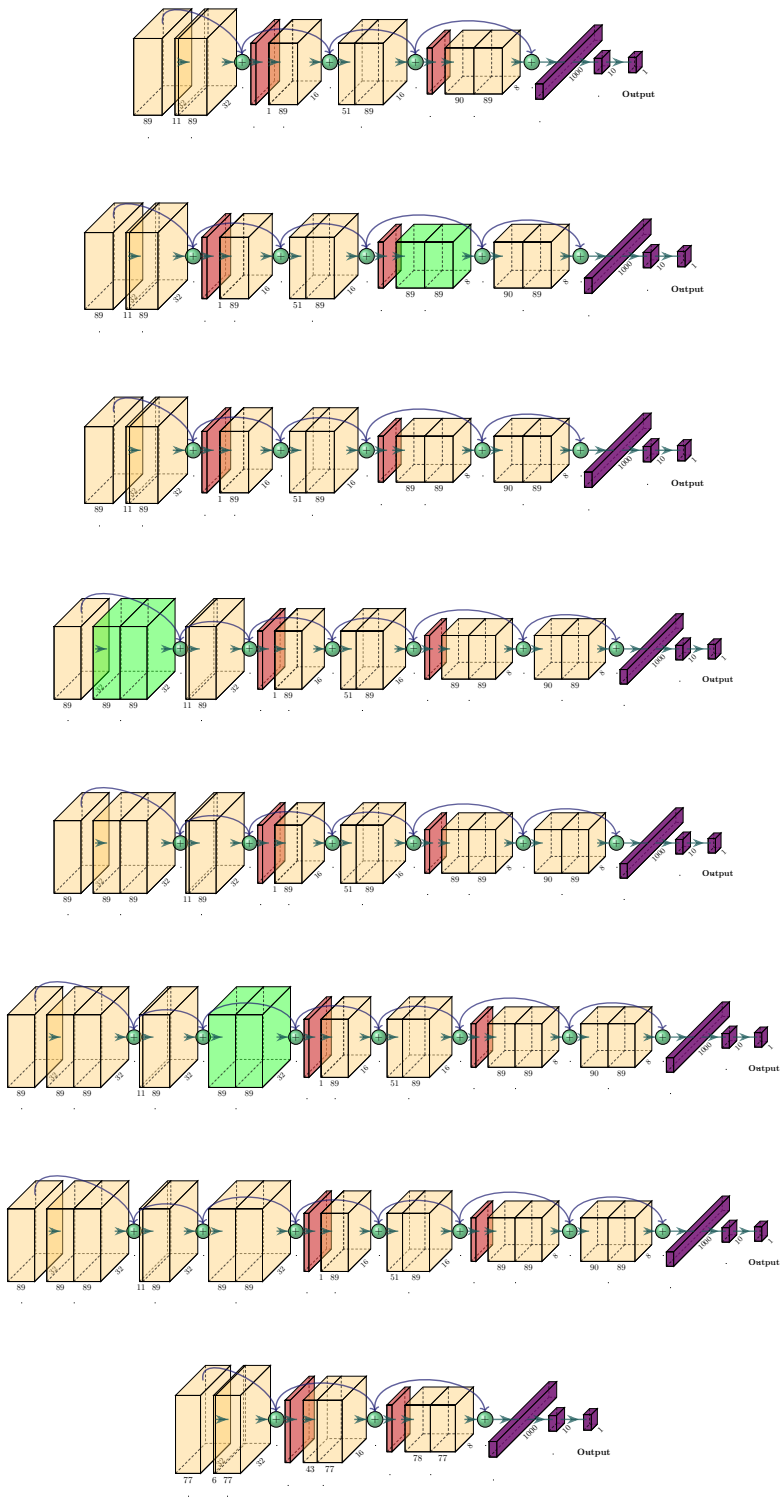


Figure 6.11: Progress of network architecture on CIFAR10 with the minimal network as initial architecture - fifth morphing routine

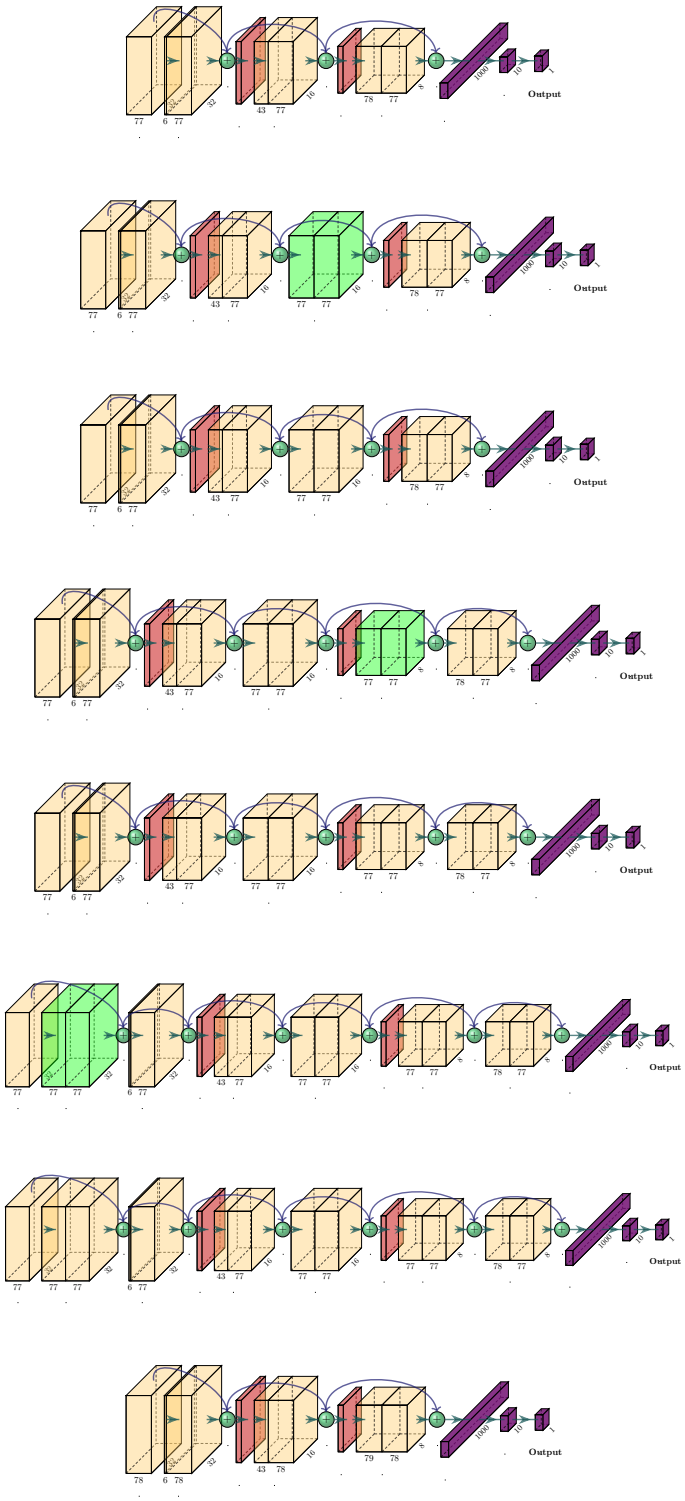


Figure 6.12: Progress of network architecture on CIFAR10 with the minimal network as initial architecture - sixth morphing routine

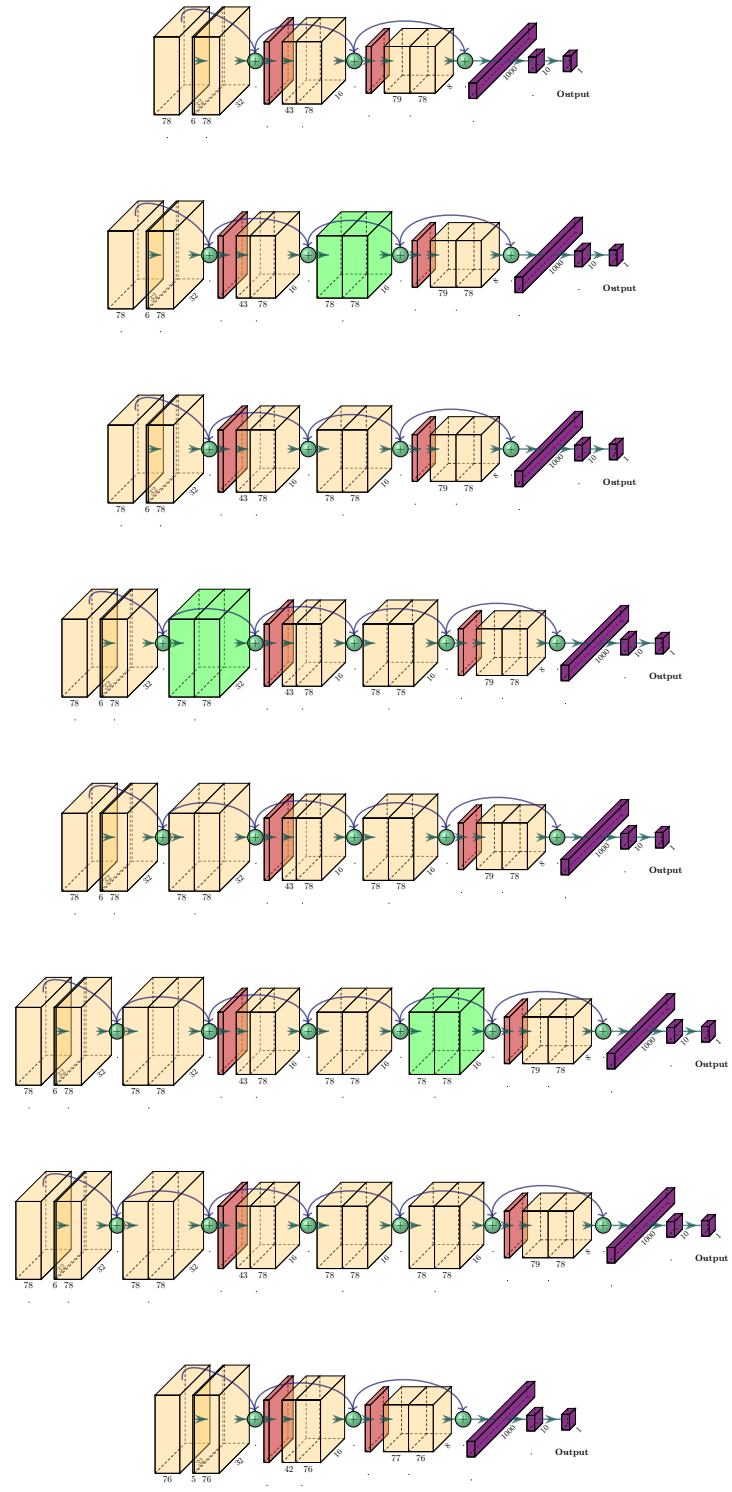


Figure 6.13: Progress of network architecture on CIFAR10 with the minimal network as initial architecture - seventh morphing routine

Reg RI and *No Reg WI*), as this would be the later use case. The results in "RB-R18" were generated analogously to the results of "RB-0Net", with the exception that a ResNet18 [50] was used as the initial architecture. We benchmark our method against the accuracy we achieve by training a ResNet18 on the specific dataset. The column "Res18" therefore gives the accuracy when it is trained with the variant "No Reg RI" which would be the typical way to go in order to train this architecture from scratch. The "MorphNet"-column shows an ablation study where we omit our additional LayerLasso routine from the architecture search process wherefore we use a ResNet18 and perform the single MorphNet routine for the given number of n_M. The training that is used for that, follows the variant "No Reg RI", as our Res18 benchmark also does.

As can be seen in Table 6.1, our ResBuilder method performs better on all datasets than both the standard ResNet18 architecture and the MorphNet approach without our depth-first search.

Fig. 6.4, Fig. 6.5 and Fig. 6.6 exemplify a complete history of the architectures during a run of the ResBuilder method, starting from ResNet18 on the CIFAR10 dataset. There we can follow the individual insertion/elimination and MorphNet steps, whereby newly inserted convolutional layers are always marked green and the widths of the individual convolutional layers can be seen in their number underneath each yellow block. Since the architecture broke down after three MorphNet iterations due to the high regularization and the loss function was thus only determined by the regularization loss term, no further architectures were considered in this run. In Fig. 6.7 to Fig. 6.13, however, the run went through all seven planned MorphNet iterations: Here, the CIFAR10 data set was also considered, but starting from our minimal start architecture. It is interesting to see that the architecture in this case at the end of each MorphNet step (the lowest architecture in the figures above) again has a similar appearance, which includes one additional block in the first pooling stage, one or two blocks in the second pooling stage and one block in the last pooling stage. Even though the blocks in this case were not eliminated from the architecture by our LayerLasso, but by the implementation of MorphNet, it can be seen that these layers of the network have no significant influence on the result, which is why the architecture converges in this way using the ResBuilder.

In Fig. 6.14 we can see in the lower panel how the accuracy for the three different training types over the iterations of architectures in the search space (shown on the x-axis) develops. Here we start searching from the minimal initial architecture (Fig. 6.1) on the CIFAR100 dataset. Meanwhile, the upper part of the figure shows how the depth of the network architecture while our ResBuilder method progresses, which is represented by the purple dashed line. Vertical red lines within the figure indicate MorphNet channel width optimization steps and the numbers next to the line in the upper part of the display the maximum channel size

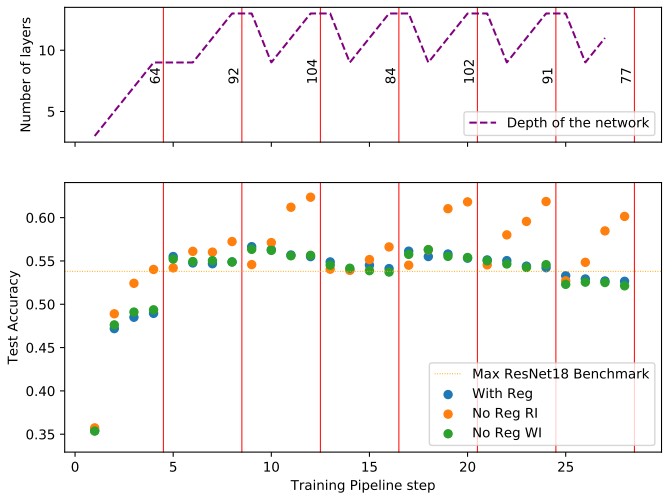


Figure 6.14: Accuracies of different architectures on CIFAR100 started with an empty net (see Fig. 6.1) as initial net.

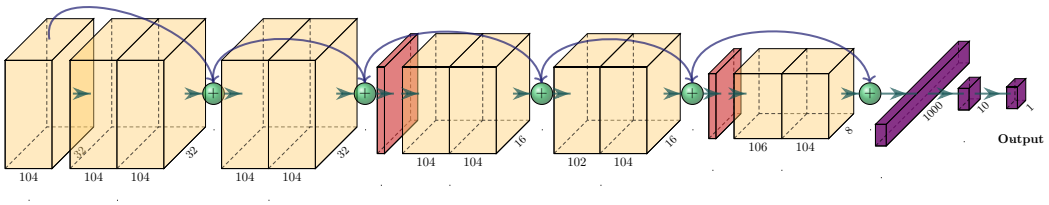


Figure 6.15: Best architecture (pipeline step 12) from Fig. 6.14.

in the first pooling block in order to get a feeling for the width of the layer. Since this figure shows an experiment that started with a minimal initial architecture, the orange line shows the benchmark that was achieved by training a ResNet18. It should be noted that even at a fairly early stage of the ResBuilder, accuracies above this benchmark could be achieved. In this experiment, the best accuracy was achieved with the twelfth architecture, which is therefore shown in Fig. 6.15. Furthermore, one can see that both the accuracy achieved and the depth and width of the network have converged very quickly in this example.

Another example is given in Fig. 6.16, which is similar to Fig. 6.14 except for the orange benchmark line, which is here the accuracy of the best MorphNet run as this run already started at the ResNet18 as initial architecture and so this benchmark can be seen from the first "column" ($x = 1$) of the figure. One other difference is the used dataset which is FashionMNIST in this case. Here it is nice to see that the size of the network in terms of both depth and width

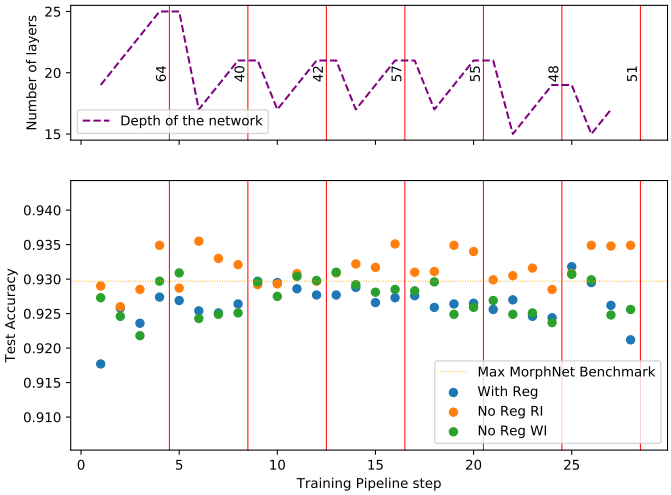


Figure 6.16: Accuracies of different architectures on FashionMNIST started with a ResNet18 architecture as initial net.

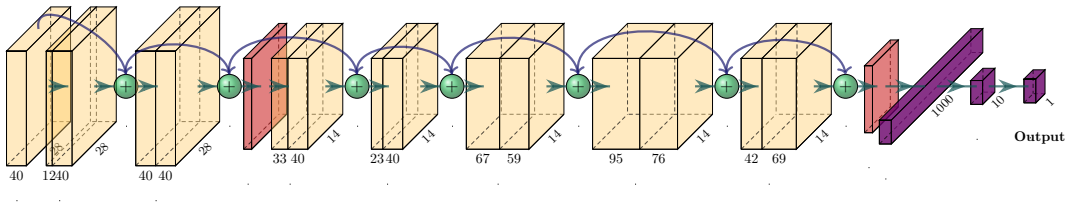


Figure 6.17: Best architecture (pipeline step 6) from Fig. 6.16.

can be steadily reduced without losing accuracy, thus saving computing resources compared to ResNet18. The best performing architecture is also shown for this example in Fig. 6.17. This may be in part due to the fact that FashionMNIST, like many of the MNIST datasets, can also be predicted well by small or cost-effective networks.

6.2.2 Removal positions

In Fig. 6.18, we show at which positions of the network our method eliminates blocks of layers. The positions between input and output are always to be seen relative to the current network size. In particular, it can be observed that for CIFAR100, which has a relatively large number of classes (100), more layers tend to be thrown out of the front part of the mesh, which could result in a focus on the classification into the individual classes instead of optimizing encoding. The

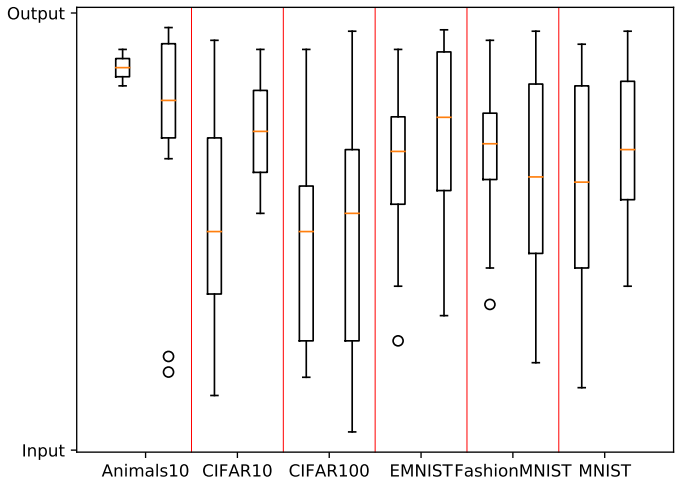


Figure 6.18: Position of removed layerblocks relative to the network size. For each dataset the left plot shows the removed positions proceeding from the minimal initial architecture and the right one proceeding from the ResNet18 architecture.

rather higher resolution of the Animals10 dataset, on the other hand, tends to throw out layer blocks at the back end of the architecture, which speaks for the importance of the front blocks for extracting information from the high resolution images.

6.2.3 Regularization parameter study

Fig. 6.19 shows the accuracies of training different architectures with various regularization strengths. To ensure better visualization, only the experiments with $\lambda_M = \lambda_\Lambda$ are considered in the figure. The shown accuracies all refer to the training variant *With Reg* without additional L2-regularization ($\lambda_0 = 0$). One can see that there is an accuracy trade-off between low FLOPs induced by a high regularization term and high FLOPs with a low regularization strength. For $\lambda_M = \lambda_\Lambda = 10^{-7}$ it can be also mentioned that the architecture broke down due to too high penalization applied.

6.2.4 Image manipulation detection in an industrial context

To challenge the approach in an industrial context, the ResBuilder method has been applied to generate an optimized model architecture, detecting manipulated

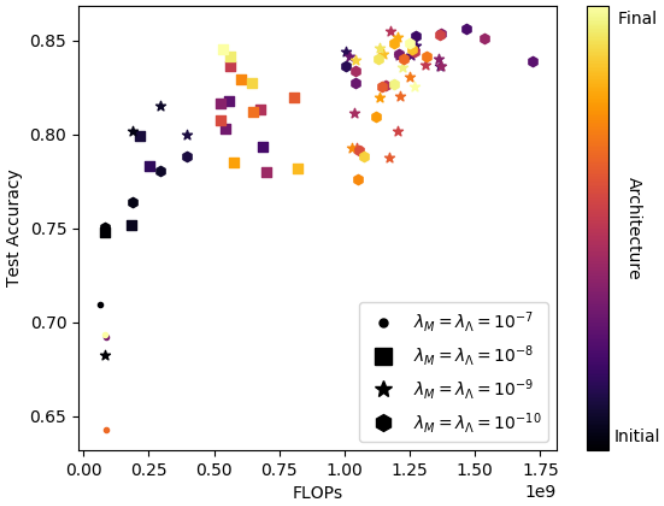


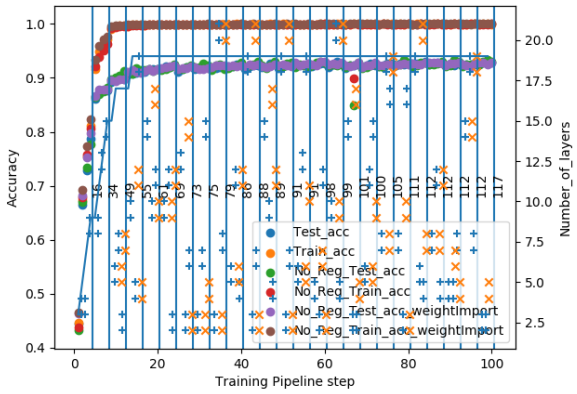
Figure 6.19: Different regularization strengths for training on CIFAR10 data.

images during insurance processes. In those processes, currently AI models predict if an image is manipulated or trustworthy, leading to according business decisions. Parts of the underlying data from these models, consisting of manipulated and authentic images from ControlExpert, have been used to test the ResBuilder method. Since the model in production solves additional tasks, and is not trained on classification data only, a fair comparison is not possible. Therefore, it has been tested how the developed approach performs compared to the training of well-established architectures (e.g. EfficientNet-b0, EfficientNet-b4, ResNet18) on these real-world data. Overall, the ResBuilder method achieves an improved accuracy (+1.2%), compared to the best performing model, that is an EfficientNet-b0, while having a smaller number of parameters.

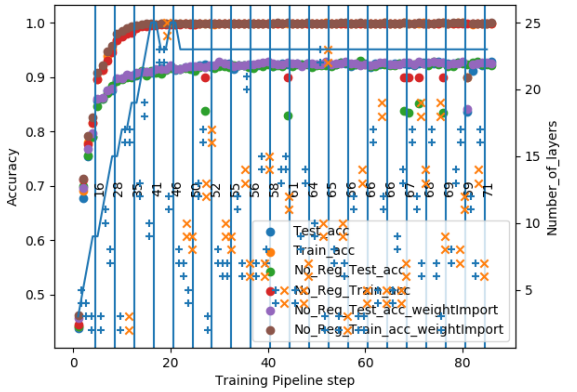
The results show that the ResBuilder method can be used to derive efficient architectures with respect to computational cost and accuracy. Another advantage of the ResBuilder method is the automatic search for efficient architectures and the minimization of the manual effort to develop an efficient model, which is an important factor in an industrial context.

6.2.5 Parameter study on long time runs

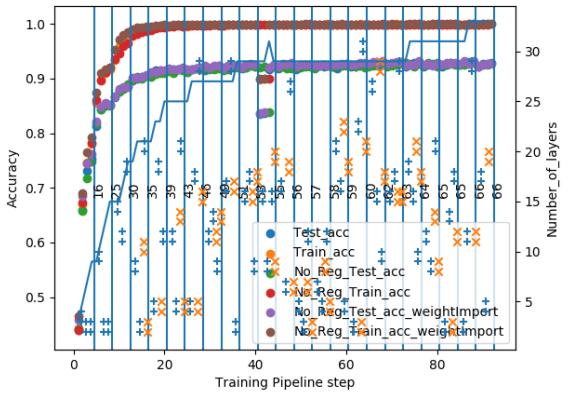
In addition to the experiments with the standard parameters as given in Subsec. 6.1.1.2, we manually adjusted hyperparameters of our ResBuilder method for the CIFAR10 dataset to obtain an accuracy value as good as possible. For this purpose, the following settings are used in the runs from Fig. 6.20:



((a)) Long run with $\lambda_M = 10^{-9}, \lambda_\Lambda = 10^{-10}, \lambda_0 = 10^{-9}$



((b)) Long run with $\lambda_M = 10^{-10}, \lambda_\Lambda = 10^{-10}, \lambda_0 = 10^{-9}$



((c)) Long run with $\lambda_M = 10^{-10}, \lambda_\Lambda = 10^{-10}, \lambda_0 = 10^{-10}$

Figure 6.20: Testing many different architectures with less MorphNet impact in ResBuilder method on CIFAR10 data.

- The number of MorphNet steps is set to $n_M = 25$.
- The L_2-regularization strength is $\lambda_0 = 10^{-10}$ and so significant lower than in our default parameter settings.
- The other two regularization strengths are also lowered to $10^{-10} \leq \lambda_\Lambda, \lambda_M \leq 10^{-9}$ according to the captions in Fig. 6.20.
- The targeted FLOPs are increased by the factor of 10: $\zeta = 1{,}000{,}000{,}000$
- The intensity of the MorphNet is reduced to $I_M = 0.1$.

Fig. 6.20 has to be read like described in Subsec. 6.1.3.

As can be seen from Fig. 6.20(c), the architectures tend to become deeper and narrower at a low regularization strength, whereas at higher regularization strengths, as in Fig. 6.20(a), wide architectures tend to be searched, provided that sufficiently large computing capacities are made available, as it is the case here.

6.2.6 Study on MorphNet intensity I_M on SmallNORB dataset with small initial architecture

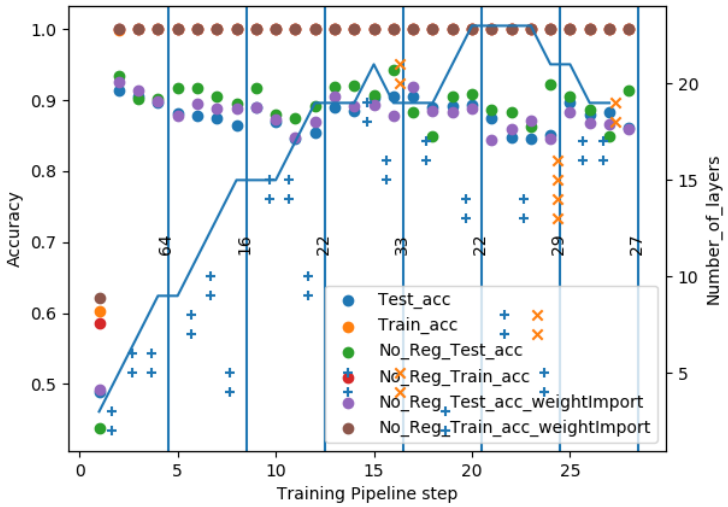


Figure 6.21: Initializing ResBuilder with default minimal initial architecture on SmallNORB dataset.

In this section, we first conducted the standard minimal architecture (as given in Fig. 6.1) on the SmallNORB dataset, which is shown in Fig. 6.21. Furthermore, we have run the ResBuilder for an even smaller architecture, whose one

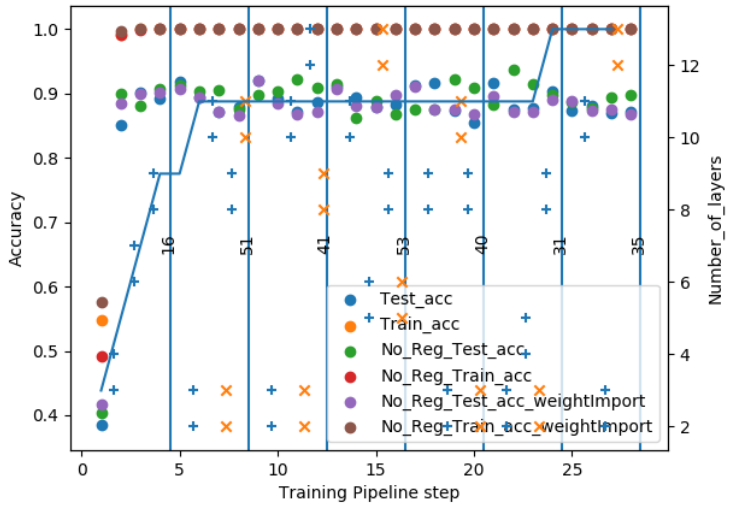


Figure 6.22: Fast depth growing architectures on SmallNORB dataset.

convolutional layer consists of only 16 channels instead of 64. The aim was to check whether ResBuilder-built networks tend to become deeper when they require fewer resources per layer. To do this, we reduced the MorphNet intensity I_M in the experiments. We benchmarked against the above test, in which we selected a MorphNet intensity of $I_M = 0.5$, providing a best accuracy of 94.23%. The two experiments with the reduced minimal architecture were also performed once with an intensity of $I_M = 0.5$ (fast-growing) and once with a lower intensity of $I_M = 0.1$ (slow-growing). In the fast-growing approach, which is shown in Fig. 6.22, it can be seen that the architectures develop quickly in width, which makes it difficult to develop in depth. After running the ResBuilder, a maximum depth of 13 layers and an accuracy of 93.72% was measured. In the slow-growing experiment, the architectures became deeper with 17 layers, but the pounding of the width development had a negative effect on the maximum accuracy, which was only 93.59%, as seen in Fig. 6.23. Thus, it can be stated for this case that it is worthwhile to start the ResBuilder with a sufficiently wide starting architecture.

6.3 Outline

In this chapter, we introduced the ResBuilder method, which provides a NAS algorithm that can generate neural networks from scratch or from existing architectures using suitable regularization techniques. On many datasets (mainly academic, but also industrial), results close to state-of-the-art (without pretraining) are achieved. To this end, ablation studies related to the omission of our

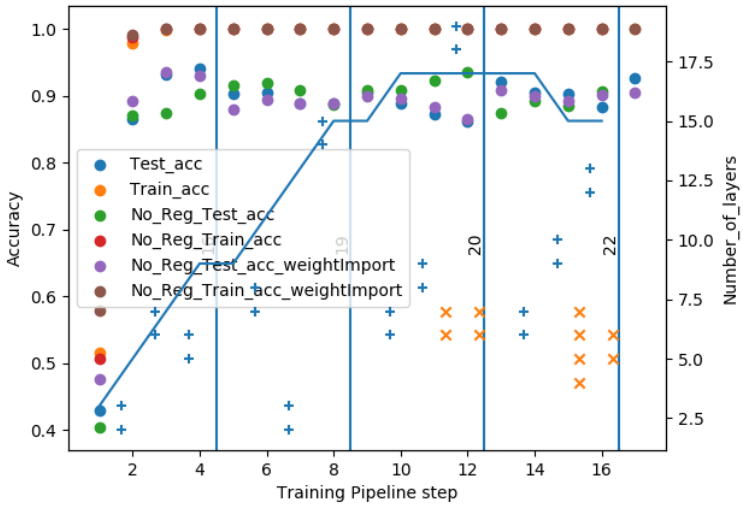


Figure 6.23: Slow depth growing architectures on SmallNORB dataset.

LayerLasso method were conducted and a parameter study on the regularization strength was performed.

Chapter 7

Neural Networks in Survival Analysis

As announced in Chapter 2, we test the applicability of machine learning not only in classification problems but also in experiments with survival analysis. In this section, we first look at the basics of survival analysis (Sec. 7.2) and related work (Sec. 7.3). After this we give details about the implementation (Sec. 7.4). The numerical results are then discussed in Sec. 7.5. This chapter is mainly based on [16].

7.1 Motivation

Clinical and pathological staging of melanoma patients only relies on tumor size (Breslow thickness), ulceration and lymph node involvement [38]. However, the patient group with the thinnest melanomas in Tumor Stage T1 and with the most favorable prognosis resulted in the most melanoma deaths in absolute numbers [67]. Additionally, patients from T3b onwards can be now offered potent adjuvant therapy [2, 77]. To identify patients with small tumors but high risk of relapse or to spare patients in advanced disease but with low mortality or recurrence risk, there is a current need for biomarkers and better prognostication [100]. The use of digitalized histological images like H&E scans, that help pathologists to better interpret the information provided by the tissue sample under the microscope, has been widely tried to use for improving diagnosis and prognosis in many tumors [43]. 3D high-resolution volumetric imaging of tissue architecture from large tissue and molecular structures at nanometer resolution are new techniques for improving early cancer detection, personalized risk assessment and potentially identifying the best treatment strategies [75]. Deep learning has been shown to

read out additional information of these stains. These models have shown to deliver independent prognostic information[26, 95] and could also predict the results of molecular biomarkers [71].

In melanoma, convolutional neural networks were shown to reach a concordance level above 80% for diagnosis compared to human pathologists[54], could outperform histopathologists in classification[55], and the result of the sentinel lymph node status [15].

Prognostication and predicting the risk of tumor recurrence could be shown for combining digital analysis with the detection of tumor-infiltrating lymphocytes by achieving a negative predicting value (NPV) of about 85% [81]. Another CNN approach for predicting disease specific survival in melanoma resulted in mixed results achieving area under the curve (AUROC) values of 90% and 88% but only NPVs of 95% and 65%, respectively, in two validation cohorts [65].

Such problems of the domain gap often arises in image recognition [41], i.e. a method that has been optimized on a data set from a certain source, but on data from a different source it provides unsatisfactory results. To address this problem, we first standardize the predictions for each dataset or data source, which already improves the accuracy. In a second approach, we use an additional regularization term directly to the neural network so that the results are in the same range. This also improves the accuracy of the predictions.

7.2 Definitions

Survival analysis is about predicting the probability of absence of an event (e.g. the death of a patient) until time t using the parameters β of a suitable model [47].

7.2.1 Cox's Proportional Hazards Model

One of the most widely used methods is Cox's Proportional Hazards Model, which predicts the hazard function $h(t|x)$ on the basis of an input vector x, see e.g., [65], [130] or [94]:

$$h(t|x) = h_0(t) \cdot \exp(\beta^T x) \Leftrightarrow \log \frac{h(t|x)}{h_0(t)} = \beta^T x \tag{7.1}$$

The baseline hazard function $h_0(t)$ indicates how large the hazard rate would be without the influence of other parameters (like β) and is therefore only dependent on the time t, which means that for our case it is eliminated from the calculation of the loss function and therefore does not need to be calculated [94].

To estimate the parameters β of the linear model the negative log partial likelihood function can be minimized:

$$l(\beta) = -\sum_{i=1}^{n} \delta_i (\beta^T x_i - \log \sum_{j \in R_i} \exp(\beta^T x_j)) \tag{7.2}$$

where δ is the delta-function which determines whether the data is censored or not, n the total number of data points and $R_i = \{j | y_j \geq y_i\}$ is the risk set which describes the data-subset of patients which do not have an event before timestamp y_i of the i-th event.

An adaption of linear models to non-linear models like (deep) neural nets was introduced by [35] in 1995 and applications can be found for example in [130] or [94]. The risk function $\beta^T x$ is replaced by the output of the neural net $\hat{h}_\omega(x)$ and we achieve the following loss function for each mini batch $\mathcal{B}$:

$$\mathcal{L}(\omega) = -\sum_{i \in \mathcal{B}} \delta_i (\hat{h}_\omega(x_i) - \log \sum_{j \in R_i} e^{\hat{h}_\omega(x_j)}) \tag{7.3}$$

7.2.2 Concordance index

For validation purposes of our methods we use Harrel's concordance index (CI) [48]. The CI is defined as the ratio between correctly ordered pairs and all possible rankable pairs [109]:

$$CI = \frac{\#\text{concordant pairs}}{\#\text{comparable pairs}} \tag{7.4}$$

A pair of observations i, j with its survival times fulfill $T_i > T_j$, is concordant if $\hat{h}_\omega(x_j) > \hat{h}_\omega(x_i)$. Also a pair i, j is not comparable if the smaller survival time is censored (i.e. $T_i > T_j \wedge \Delta_j = 0$). Otherwise, this pair is comparable. Thus,

$$CI = \frac{\sum_{i,j} \mathbb{1}(T_i > T_j) \cdot \mathbb{1}(\hat{h}_\omega(x_j) > \hat{h}_\omega(x_i)) \cdot \Delta_j}{\sum_{i,j} \mathbb{1}(T_i > T_j) \cdot \Delta_j} \tag{7.5}$$

The CI estimates the probability of concordance $P(\hat{h}_\omega(x_j) > \hat{h}_\omega(x_i) | T_i > T_j)$ for two independent observations/predictions. It can also be interpreted as a measure of the area under a time-dependent receiver operator curve [44, 53, 109]. A value of $CI = 1$ means that all observations are correctly sequenced, $CI = 0.5$ means that the method applied is no better than guessing.

7.2.3 Area Under the Receiver Operating Characteristic

In the evaluation of the results our methods generate, an important measurement value is the area under the receiver operating characteristic (AUROC)[45]. For a data point with $a = P(F_p)$ the probability of the data point being a false positive prediction and $b - 1 = P(T_p)$ the negative probability of the data point being a true positive prediction in a dataset D, [14] defines the AUROC as follows:

$$\text{AUROC} = \sum_{i \in D} \{(1 - b_i \cdot \Delta a) + \frac{1}{2}[\Delta(1 - b) \cdot \Delta a]\}$$

with

$$\Delta(1 - b) = (1 - b_i) - (1 - b_{i-1})$$

and

$$\Delta a = a_i - a_{i-1}$$

7.3 Related Work

There exists abundant work on using deep neural networks for the interpretation of medical images. In this section, we focus on works with the scope of AI based prediction of survival for patients with a melanoma diagnosis.

In prior research on this topic [65] the authors use a two stage pipeline to predict risk on the basis of primary melanoma tumor images. Within this pipeline, they first use a segmentation to classify detect and crop tumor areas in the image. These small but detailed crops of 500×500px are then fed to a network of convolutional, recurrent and fully connected layers in order to predict the risk.

[60] describes the approach to use a multivariable classifier that contains, besides clinical data, a score of a deep neural net, in order to predict the immunotherapy response of patients with advanced melanoma. Also here, clinical data is used for the model and a separation of segmentation and response classifier takes place.

The authors of [55] describe an experiment where a trained ResNet50 model outperforms 11 pathologists in classifying labeled histopathological images what shows that neural nets in general have high potential to improve correct melanoma diagnoses.

Similar to our approach, the authors of [73] used a VGG-based neural network architecture to detect cutaneous melanoma, although they use a binary classification for dead/alive patients instead of a survival analysis method. They evaluated their method on a dataset provided by The Cancer Imaging Archive of 53 patients with a given survival status.

In [111] the authors on the one hand locate molecular biomarkers in immunohistochemistry images using convolutional neural networks which can help enabling new cancer screenings. On the other hand they also classify the found biomarkers along their type.

However, there are not only machine learning approaches to melanoma classification based on H&E scans, but also based on topics such as dermoscopic image data [90] where the authors use a quiet small 5-layers CNN to classify the images to the corresponding tumor stage with applying the Adam optimizer on the Similarity Measure for Text Processing as loss function. This work is also based on [59] where the authors predict the melanomas thickness using a pretrained VGG-19 model on 400×400px preprocessed dermoscopic images.

In [130] the authors also use deep convolutional neural networks in combination with survival analysis in order to find good predictions on pathological images - here in context of lung cancer. They annotated the regions of interest of the images with help of pathologists and sampled small random high resolution crops of these regions to use in the networks whereas we used down-sampled low resolution images in our approach.

In contrast to this, in our work we extract information directly from the original image data using a VGG16-like neural network, so we don't use an additional presegmentation, which might be error-prone by itself. In this way we also avoid labeling of the regions of interest by humans annotators, which is a time-consuming process and requires highly trained annotators.

7.4 Implementation

7.4.1 Datasets

In this section of survival analysis we consider two main sets of data:

- *Dataset A*: This dataset contains 767 images of 176 patients of the American Joint Committee on Cancer (AJCC) set stages IA to IIID from four different locations in Bern (Switzerland), Bochum, Bonn and Kiel, Germany. All images have been recorded with the H&E coloring and were created by the same scanner Hamatasu NanoZoomer S210, NDP Version 2.4.

  In order to obtain a data set of the highest possible quality, the data were first manually checked by medical experts and a total of 23 data points were removed from the data set, e.g. due to broken slides. This clean-up took place before we split the data into training, validation and test data:

Characteristic	Dataset A			Dataset B
	Training	Validation	Test	Test
N(Scans)	313	36	38	242
N(Patients)	104	36	36	242
Alive / Censored	89 (86%)	31 (86%)	31 (86%)	212 (88%)
Dead / Event	15 (14%)	5 (14%)	5 (14%)	30 (12%)
MSS time (months) Mean	70.12	80.03	78.37	50
MSS time (months) Median	70	73	73	41
Relapse Free Survival Recurrence	21 (20%)	8 (22%)	7 (19%)	75 (31%)
Relapse Free Survival Non-recurrence	83 (80%)	28 (78%)	29 (81%)	167 (69%)
RFS time Mean	67	76.67	75	42
RFS time Median	70	68	73	30

Table 7.1: Important features of our survival analysis datasets [16].

The patients are assigned to train (104 patients), test (36 patients) and validation (36 patients) what results in 591 train-, 74 validation- and 102 test images. The split was chosen so that the distribution of high/low risk patients in each subset (training, validation, test) corresponds to the distribution of the full dataset A, see also Table 7.1, under the constraint that multiple images of one patient remain in the same subset.

- *Dataset B*: The second dataset contains 242 images of 242 patients with AJCC stages IIA to IIC from the Central Malignant Melanoma Registry (CMMR) in Tübingen, Germany, where the H&E-colored images have partly different coloring than the images of *Dataset A*, probably caused by another scanner type Hamatasu Nanozoomer 2.0 HAT, NDP Version 2.5 and the scans are mostly disturbed by a marker pen on the slide. To see how our algorithm performs on images with a domain gap *dataset B* is only used as a separate test dataset.

For more (clinical) information on the survival analysis dataset we use, have a look in [16].

7.4.2 Clinical description

The images used represent hematoxylin-eosin (HE) stained melanoma sections. HE is a staining technique from histology that is used to better predict the disease prognosis of a melanoma patient, among other things, the mitosis rate can be determined (S3 Leitlinie Melanom).

Overall survival of dataset A (86%) is similar to dataset B (88%). However, the MSS time for dataset B with a median of 41 months was considerably lower compared to 70/ 73 months of dataset A. Relapse free survival differed as well with around 78% and 69% for dataset A and dataset B, respectively.

7.4.2.1 Technical Description

Both datasets contain a total of 1009 images of different sizes (up to a resolution of 158720×115456 pixels) and various format ratios. Therefore, the images have to be pre-processed as neural nets on commercially available hardware are not yet able to handle images of this size at the time of writing. See section Subsec. 7.4.3 for more information on the pre-processing task. The total size of the dataset is 570,3 GB in `.ndpi` file format.

For every patient i, we have at least one image x_i and the information $\delta_i = 1$, indicating the death of the patient at time T_i. If the patient survived the observation period of this study, we set $\delta_i = 0$ and T_i is the time the patient has been observed.

7.4.3 Preprocessing the Data

When pre-processed, the images are reduced from their original format by a factor of 64 in dataset A and 128 in dataset B in each dimension. The resulting image is centered in a 2500×2000 pixel frame which is filled with white color outside the image. Images of patients with multiple images are seen as independent information in the training data.

7.4.4 Methods

Convolutional neural networks (CNN) represent the state of the art in image recognition as we have seen in Chapter 2. As neural network model, we use a modified version of a VGG16 net [113] in our experiments. Figure Fig. 7.1 gives an overview of the network architecture. Each convolutional layer has kernel sizes of 3×3 and each pooling layer is a maximum pooling with pooling size 2×2 like described in Subsec. 2.2.7.2.

As mentioned in section Sec. 7.3, we use the loss function supplied in [94] which implements a variant of the Cox's Proportional Hazards Model. In order to

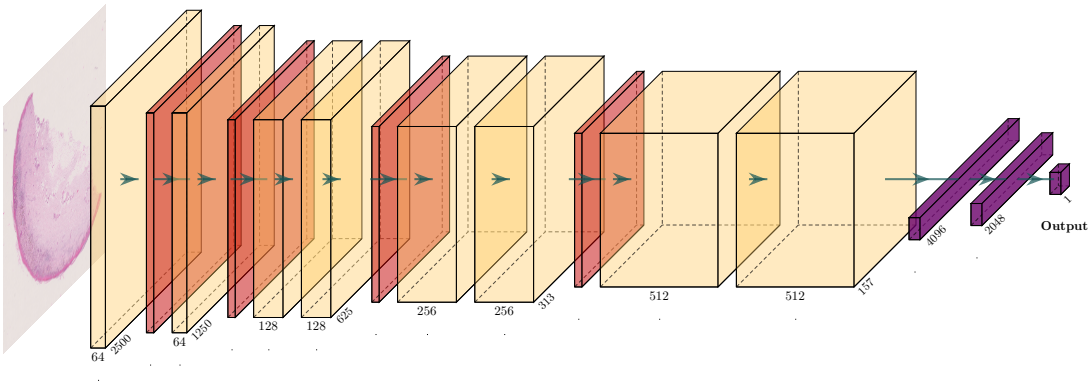


Figure 7.1: Design of used neural network. Orange are convolutional layers, red are pooling layers and purple indices dense layers.

achieve a higher generalizability and reduce the domain gap between dataset A and dataset B, we expanded our loss function in (7.3) by a regularization term:

$$\mathcal{L}(\omega) = -\sum_{i \in \mathcal{B}} \delta_i (\hat{h}_\omega(x_i) - \log \sum_{j \in R_i} e^{\hat{h}_\omega(x_j)}) + \lambda (\frac{1}{|\mathcal{B}|} (\sum_{j \in \mathcal{B}} \hat{h}_\omega(x_j))^2) \tag{7.6}$$

where λ is the chosen regularization strength. See sec Sec. 7.5.1 for further information why we use this additional regularization term.

In training, we employ the Adam optimizer over 50 epochs with a learning rate of $\alpha = 0.001$ and a batch size of 5.

For the implementation we use Openslide (version 1.1.2) [40] for importing the images, the GitHub repository of Sebp [94] and Tensorflow (version 2.6.1) [3] along with Keras (version 2.6.0) [22].

7.5 Numerical results

For the calculations and so our now presented numerical results, we used a workstation with a Dual Intel Xeon Gold 6248R 3.0GHz and three Nvidia Quadro RTX 8000 graphic units with 48GB VRAM each, whereas the different GPUs are only used for different trainings.

An overview of all the concordance indices and AUROC values of our experiments is provided in table Table 7.2.

	Without regularization				With regularization			
	Isolated		Merged		Isolated		Merged	
	A (test)	B	Naive	Std	A (test)	B	Naive	Std
C-Index	0.677	0.612	0.569	0.644	0.795	0.615	0.646	0.676
AUROC	0.615	0.635	0.544	0.614	0.789	0.578	0.601	0.642

Table 7.2: Evaluation of receiver operator curve and Harrel's C-Index with and without regularization on different testsets (Std = standardized).

7.5.1 Results of model without regularization

In our first set of experiments, we use the loss function supplied in equation ((7.3)). Although we achieve an AUROC value of 61.5% on the test subset of dataset A and also an AUROC value of 63.5% on the separated dataset B, we observe that the domains in which our predictions $\hat{h}_\omega(x_i)$ lay differ significantly from dataset A to dataset B.

This results in a bad overall AUROC value of 54.4% if one mixes these predictions naively together as it would be a complete dataset from only one datasource before evaluating the whole set. One potential reason is that the net learns features that are relevant for the task, but it also is sensitive to further properties of the image like the pixel's brightness (or in general the pixel's color distribution). While such image features do not encode meaningful medical information, they can still disturb the outcome of the network, especially if the hazards predicted on the second data set are in a different numerical range as compared with the original training data. This in particular happens by deviation from the neutral direction $\hat{h}_\omega(x_j) \to \hat{h}_\omega(x_j) + z$ which merely leads to a redefinition of the baseline hazard function $h_0(t)$ but is not sensed in the Cox loss function. So one can imagine the predictions $\hat{h}_\omega(x_i)$ shifted away from the total diagonal like schematically shown in figure Fig. 7.2.

In our case, dataset B has a slightly other color scheme (mainly because of the use of another scanner - see section Subsec. 7.4.1) and so our predictions evaluating the model trained on dataset A on this dataset leads to a significant drop in performance.

A first approach to reduce the sensitivity towards different hazards is to re-center the predicted hazards. We thus standardize (we subtract the mean and divide the result by the standard deviation) the predictions of each dataset and merge them afterwards. We display the resulting improvement in the rightest column of each approach in table Table 7.2.

The downside of this approach is that a certain set of images for each source of images/scanner, that is used to predict survival probabilities, has to be available.

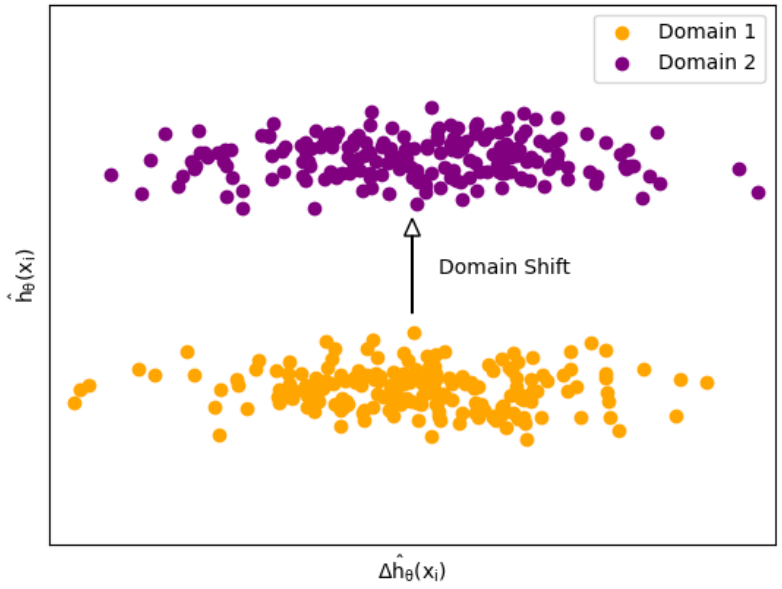


Figure 7.2: Domain shift in predicting on different datasets.

7.5.2 Results of model with regularization

To interdict shifting of hazard functions altogether, we add a L2-regularization-term, see equation ((7.6)), to shift all predictions $\hat{h}_\omega(x_i)$ in the same domain range. This does not only lead to a higher AUROC value in evaluation on the mixed data set (60.1% instead of 54.4%), but we can improve even more when combining the regularization with the above-mentioned normalization process (64.2% instead of 61.4%)

Besides, we also achieve a significantly higher ROC value on the testdata of dataset A throughout this regularization technique. Unfortunately the overall generalizability on the isolated dataset B suffered (57.8% instead of 63.5%) from that approach.

7.5.3 Conclusion

We have shown that it is possible to make survival predictions based on simplified image information using Cox's Proportional Hazard Models on neural networks and that our domain adaptation techniques succeed in merging the predictions into one range. Compared to [73] where the authors achieved an AUROC value of

76.9% our approach with regularization slightly outperforms it with an AUROC value of 79.5%.

In future work, we will have a more detailed evaluation and we have a closer look on levels of significance and correlations to clinical variables like the Breslow depth or genetic information and demographic variables such as age or gender to evaluate and improve the clinical utility of our predictions.

Chapter 8

Conclusion

In this thesis, the newly developed NAS method ResBuilder [18] was presented in Chapter 4, which can be used to create convolutional neural network architectures for classification problems from scratch or to optimize already existing architectures with regard to their accuracy or a better effectiveness of computing capacities for prediction quality.

The ResBuilder method works in such a way that in an insertion routine, residual blocks are first inserted randomly into the existing architecture without affecting the training too much, which means that in this case one does not have to retrain the architecture, but can continue training from the current state, which is a clear advantage compared to other NAS approaches. To enable the most efficient structure possible, however, it is just as necessary to remove unneeded structural elements from the network. This is done by using different types of regularization, which keeps the weights of the neural network artificially small. If the weights of a layer within the neural network are pushed below a predefined threshold, the entire residual block in which this layer lies is removed from the current architecture, as it can be assumed that it does not contribute much added value to the prediction quality of the network.

For this purpose, the computing capacity available can be determined in advance, which means that for example small companies can also build their own adapted architectures into their processes and thus have an easier entry into the topic of Deep Learning or artificial intelligence in general. That the method works well was shown in Chapter 6, where near state-of-the-art accuracies (without pre-training) were achieved for preset standard parameters on academic data sets.

The method was also convincing in practice, as seen in Subsec. 6.2.4. Since regularization techniques play a major role in the ResBuilder, several parameter studies were carried out on different regularization strengths and their influence

on each other was tested. Ablation studies were performed for the method, where our introduced sub-method LayerLasso was omitted for experiments and only the MorphNet [42] part of the ResBuilder method was applied as seen in Subsec. 6.2.1. Because MorphNet is also an important part of the ResBuilder, we also did some numerical experiments on the MorphNet routine in Chapter 5 for example to see how it handles with a bad initial architecture in Sec. 5.2.

In future experiments, it would be nice to see how well this method can be applied to other types of problems, such as object detection, semantic segmentation or similar, where convolutional networks are also used.

In addition, we have seen a physically motivated approach in Chapter 3 with the PHS optimization method [17], which is a variation of the stochastic gradient descent method with momentum. We have evaluated this by numerous experiments and further improved it by incorporating a goal-oriented braking of the heavy ball with friction. As can be seen in the numerical results in Sec. 3.4, this method outperforms the simple stochastic gradient descent algorithm as well as a momentum based optimization with fixed friction. This approach makes it possible to switch from exploration to an exploitation procedure by suddenly braking as soon as a certain point of potential energy has been reached, whereby the minimum in which the optimization algorithm then finds itself can be optimally exploited, through even overfitting is additionally avoided.

There could be, some kind of re-acceleration taking place in future experiments, allowing multiple minima to be visited within one training run. This would be possible in this perspective through the introduction of an external force/"port".

Finally, an application of machine learning in the field of survival analysis [16] was shown in Chapter 7. Here, the Cox's Proportional Hazard Model Subsec. 7.2.1 was adapted to the data of the cooperation partner NeraCare by using a neural network [94] in such a way that it determined survival scores well for predictive images. The problem of the domain gap arose because the method was evaluated on images from another data source/scanner and compared with the scores from the images from the first scanner, which were in different ranks. This problem was solved by a suitably chosen regularization technique, which significantly improved the prediction quality, at least for the two scanners. Further experiments could now determine whether this technique is also successful on other data sets, or whether, for example, it is necessary to fall back on the procedure for calibrating new scanners also presented in Subsec. 7.4.4.

List of Figures

List of Tables

List of Algorithms & Scripts

List of Notations

Throughout this thesis, the following abbreviations and notations are used across all chapters:

x	Input data (mostly an image)
y	Label of x
$\mathcal{X}$	Set of input data
$\mathcal{Y}$	Set of possible labels y
S	Sampled trainings set
F	Neural Network function
$F(x_i)$	Output before Softmax of the neural net F for an input x_i
$\hat{y}_i$	Predicted label of a neural network F for an input x_i
f	Function of a part of the neural network F
l_i	Layer at position i within a neural network
B	Block of layers within a neural network
$\omega_{a,b}^{(l_i)}$	Weight of the edge from neuron a in layer l_i to neuron b in layer l_{i+1}
α	Learning rate
$\mathbf{v}$	Momentum (e.g. in SGD)
$\mathcal{L}$	Lossfunction
$\mathcal{B}$	Mini-Batch
n_Λ	ResBuilder: Number of insertion steps before MorphNet step
n_M	ResBuilder: Maximum number of MorphNet steps
Θ_{LL}	ResBuilder: Status of LayerLasso-Momentum
τ_{LL}	ResBuilder: Threshold to activate LayerLasso-Momentum
λ_M	ResBuilder: MorphNet regularization strength
I_M	ResBuilder: Intensity of MorphNet suggestions
ζ	MorphNet and ResBuilder: Aimed for FLOP costs
λ_Λ	ResBuilder: LayerLasso regularization strength
τ_Λ	ResBuilder: LayerLasso threshold
λ_0	ResBuilder: $L2$-regularization strength
$h(t\|x)$	Hazard function
$h_0(t)$	baseline hazard function
CI	Concordance index
AUROC	Area under the receiver operating characteristic

Bibliography

[1] *Animals10 dataset.* `https://www.kaggle.com/datasets/alessiocorrado99/animals10`. Accessed: 2022-05-30.

[2] *Bristol myers squibb announces adjuvant treatment with opdivo (nivolumab) demonstrated statistically significant and clinically meaningful improvement in recurrence-free survival (rfs) in patients with stage iib/c melanoma in the checkmate -76k trial.* `https://bit.ly/3dfAO8B`. Accessed: September 16, 2022.

[3] 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.

[4] K. Ahmed and L. Torresani, *Maskconnect: Connectivity learning by gradient descent*, in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 349–365.

[5] S. Amari, *A theory of adaptive pattern classifiers*, IEEE Transactions on Electronic Computers, (1967), pp. 299–307.

[6] A. Antipin, *Second order proximal differential systems with feedback control*, Differential Equations, 29 (1993), pp. 1597–1607.

[7] G. B. Arous, R. Gheissari, and A. Jagannath, *Online stochastic gradient descent on non-convex losses from high-dimensional inference*, The Journal of Machine Learning Research, 22 (2021), pp. 4788–4838.

[8] H. Attouch, Z. Chbani, J. Peypouquet, and P. Redont, *Fast convergence of inertial dynamics and algorithms with asymptotic vanishing viscosity*, Mathematical Programming, 168 (2018), pp. 123–175.

[9] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty, *Nonlinear Programming – Theory and Algorithms*, Wiley, 3rd ed., 2006.

[10] S. Becker, Y. Zhang, et al., *Geometry of energy landscapes and the optimizability of deep neural networks*, Physical review letters, 124 (2020), p. 108301.

[11] Y. Bengio, *Practical recommendations for gradient-based training of deep architectures*, Neural Networks: Tricks of the Trade: Second Edition, (2012), pp. 437–478.

[12] S. Bock and M. Weiss, *A proof of local convergence for the adam optimizer*, in 2019 international joint conference on neural networks (IJCNN), IEEE, 2019, pp. 1–8.

[13] R. N. Bracewell and R. N. Bracewell, *The Fourier transform and its applications*, vol. 31999, McGraw-Hill New York, 1986.

[14] A. P. Bradley, *The use of the area under the roc curve in the evaluation of machine learning algorithms*, Pattern recognition, 30 (1997), pp. 1145–1159.

[15] T. J. Brinker, L. Kiehl, M. Schmitt, T. B. Jutzi, E. I. Krieghoff-Henning, D. Krahl, H. Kutzner, P. Gholam, S. Haferkamp, J. Klode, et al., *Deep learning approach to predict sentinel lymph node status directly from routine histology of primary melanoma tumours*, European Journal of Cancer, 154 (2021), pp. 227–234.

[16] J. Burghoff, L. Ackermann, Y. Salahdine, V. Bram, K. Wunderlich, J. Balkenhol, T. Dirschka, and H. Gottschalk, *Risk stratification of malignant melanoma using neural networks*, 2023.

[17] J. Burghoff, M. H. Monells, and H. Gottschalk, *Who breaks early, looses: goal oriented training of deep neural networks based on port hamiltonian dynamics*, 2023.

[18] J. Burghoff, M. Rottmann, J. von Conta, S. Schoenen, A. Witte, and H. Gottschalk, *Resbuilder: Automated machine*

learning with residual structures. Code: `https://github.com/JuliBu/ResBuilder`, 2023.

[19] A. Cabot, H. Engler, and S. Gadta, *On the long time behavior of second order differential equations with asymptotically small dissipation*, Transactions of the American Mathematical Society, 361 (2009), pp. 5983–6017.

[20] A. Cauchy et al., *Méthode générale pour la résolution des systemes d'équations simultanées*, Comp. Rend. Sci. Paris, 25 (1847), pp. 536–538.

[21] A. Chambolle and C. Dossal, *On the convergence of the iterates of the "fast iterative shrinkage/thresholding algorithm"*, J. Optim. Theory Appl., 166 (2015), pp. 968–982.

[22] F. Chollet et al., *Keras*. `https://keras.io`, 2015.

[23] T. Choudhary, V. Mishra, A. Goswami, and J. Sarangapani, *A comprehensive survey on model compression and acceleration*, Artificial Intelligence Review, 53 (2020), pp. 5113–5155.

[24] K. Chowdhary and K. Chowdhary, *Natural language processing*, Fundamentals of artificial intelligence, (2020), pp. 603–649.

[25] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik, *Emnist: Extending mnist to handwritten letters*, in 2017 International Joint Conference on Neural Networks (IJCNN), IEEE, 2017, pp. 2921–2926.

[26] M. Combalia, N. Codella, V. Rotemberg, C. Carrera, S. Dusza, D. Gutman, B. Helba, H. Kittler, N. R. Kurtansky, K. Liopyris, et al., *Validation of artificial intelligence prediction models for skin cancer diagnosis using dermoscopy images: the 2019 international skin imaging collaboration grand challenge*, The Lancet Digital Health, 4 (2022), pp. e330–e339.

[27] C. Darken, J. Chang, J. Moody, et al., *Learning rate schedules for faster stochastic gradient search*, in Neural networks for signal processing, vol. 2, Citeseer, 1992, pp. 3–12.

[28] C. Darken and J. Moody, *Note on learning rate schedules for stochastic optimization*, Advances in neural information processing systems, 3 (1990).

[29] F. M. Dekking, C. Kraaikamp, H. P. Lopuhaä, and L. E. Meester, *A Modern Introduction to Probability and Statistics: Understanding why and how*, vol. 488, Springer, 2005.

[30] R. C. Deo, *Machine learning in medicine*, Circulation, 132 (2015), pp. 1920–1930.

[31] X. Dong and Y. Yang, *Network pruning via transformable architecture search*, Advances in Neural Information Processing Systems, 32 (2019).

[32] J. Duchi, E. Hazan, and Y. Singer, *Adaptive subgradient methods for online learning and stochastic optimization.*, Journal of machine learning research, 12 (2011).

[33] T. Elsken, J. H. Metzen, and F. Hutter, *Neural architecture search: A survey*, The Journal of Machine Learning Research, 20 (2019), pp. 1997–2017.

[34] B. J. Erickson, P. Korfiatis, Z. Akkus, and T. L. Kline, *Machine learning for medical imaging*, Radiographics, 37 (2017), pp. 505–515.

[35] D. Faraggi and R. Simon, *A neural network model for survival data*, Statistics in medicine, 14 (1995), pp. 73–82.

[36] H. Fujiyoshi, T. Hirakawa, and T. Yamashita, *Deep learning-based image recognition for autonomous driving*, IATSS research, 43 (2019), pp. 244–252.

[37] Y. Geifman and R. El-Yaniv, *Deep active learning with a neural architecture search*, Advances in Neural Information Processing Systems, 32 (2019).

[38] J. E. Gershenwald and R. A. Scolyer, *Melanoma staging: American joint committee on cancer (ajcc) and beyond*, Annals of surgical oncology, 25 (2018), pp. 2105–2110.

[39] G. Goh, *Why momentum really works*, Distill, 2 (2017), p. e6.

[40] A. Goode, B. Gilbert, J. Harkes, D. Jukic, and M. Satyanarayanan, *Openslide: A vendor-neutral software foundation for digital pathology*, Journal of pathology informatics, 4 (2013), p. 27.

[41] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, MIT press, 2016.

[42] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T.-J. Yang, and E. Choi, *Morphnet: Fast & simple resource-constrained structure learning of deep networks*, (2017).

[43] M. Gurcan, L. Boucheron, A. Can, A. Madabhushi, N. Rajpoot, and B. Yener, *Histopathological image analysis: a review. ieee rev biomed eng. 2009; 2: 147–71*, 2009.

[44] K. Hajian-Tilaki, *Receiver operating characteristic (roc) curve analysis for medical diagnostic test evaluation*, Caspian journal of internal medicine, 4 (2013), p. 627.

[45] J. A. Hanley and B. J. McNeil, *The meaning and use of the area under a receiver operating characteristic (roc) curve.*, Radiology, 143 (1982), pp. 29–36.

[46] S. Hanson and L. Pratt, *Comparing biases for minimal network construction with back-propagation*, Advances in neural information processing systems, 1 (1988).

[47] F. E. Harrell et al., *Regression modeling strategies: with applications to linear models, logistic regression, and survival analysis*, vol. 608, Springer, 2001.

[48] J. Harrell, Frank E., R. M. Califf, D. B. Pryor, K. L. Lee, and R. A. Rosati, *Evaluating the Yield of Medical Tests*, JAMA, 247 (1982), pp. 2543–2546.

[49] K. He and J. Sun, *Convolutional neural networks at constrained time cost*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 5353–5360.

[50] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[51] X. He, K. Zhao, and X. Chu, *Automl: A survey of the state-of-the-art*, Knowledge-Based Systems, 212 (2021), p. 106622.

[52] Y. He, X. Zhang, and J. Sun, *Channel pruning for accelerating very deep neural networks*, in Proceedings of the IEEE international conference on computer vision, 2017, pp. 1389–1397.

[53] P. J. Heagerty and Y. Zheng, *Survival model predictive accuracy and roc curves*, Biometrics, 61 (2005), pp. 92–105.

[54] A. Hekler, J. S. Utikal, A. H. Enk, C. Berking, J. Klode, D. Schadendorf, P. Jansen, C. Franklin, T. Holland-Letz, D. Krahl, et al., *Pathologist-level classification of histopathological melanoma images with deep neural networks*, European Journal of Cancer, 115 (2019), pp. 79–83.

[55] A. Hekler, J. S. Utikal, A. H. Enk, W. Solass, M. Schmitt, J. Klode, D. Schadendorf, W. Sondermann, C. Franklin,

F. Bestvater, et al., *Deep learning outperformed 11 pathologists in the classification of histopathological melanoma images*, European Journal of Cancer, 118 (2019), pp. 91–96.

[56] K. Hornik, M. Stinchcombe, and H. White, *Multilayer feedforward networks are universal approximators*, Neural networks, 2 (1989), pp. 359–366.

[57] C.-H. Hsu, S.-H. Chang, J.-H. Liang, H.-P. Chou, C.-H. Liu, S.-C. Chang, J.-Y. Pan, Y.-T. Chen, W. Wei, and D.-C. Juan, *Monas: Multi-objective neural architecture search using reinforcement learning*, arXiv preprint arXiv:1806.10332, (2018).

[58] H. Iqbal, *Harisiqbal88/plotneuralnet v1.0.0*, Dec. 2018.

[59] J. Jaworek-Korjakowska, P. Kleczek, and M. Gorgon, *Melanoma thickness prediction based on convolutional neural network with vgg-19 model transfer learning*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, 2019, pp. 0–0.

[60] P. Johannet, N. Coudray, D. M. Donnelly, G. Jour, I. Illa-Bochaca, Y. Xia, D. B. Johnson, L. Wheless, J. R. Patrinely, S. Nomikou, et al., *Using machine learning algorithms to predict immunotherapy response in patients with advanced melanoma*, Clinical Cancer Research, 27 (2021), pp. 131–140.

[61] A. I. Khan and S. Al-Habsi, *Machine learning in computer vision*, Procedia Computer Science, 167 (2020), pp. 1444–1451.

[62] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, arXiv preprint arXiv:1412.6980, (2014).

[63] N. B. Kovachki and A. M. Stuart, *Continuous time analysis of momentum methods*, Journal of Machine Learning Research, 22 (2021), pp. 1–40.

[64] A. Krizhevsky, G. Hinton, et al., *Learning multiple layers of features from tiny images*, (2009).

[65] P. M. Kulkarni, E. J. Robinson, J. Sarin Pradhan, R. D. Gartrell-Corrado, B. R. Rohr, M. H. Trager, L. J. Geskin, H. M. Kluger, P. F. Wong, B. Acs, et al., *Deep learning based on standard h&e images of primary melanoma tumors identifies patients at risk for visceral recurrence and deathdeep learning–based prognostic biomarker for melanoma*, Clinical Cancer Research, 26 (2020), pp. 1126–1134.

[66] S. Kullback and R. A. Leibler, *On information and sufficiency*, The annals of mathematical statistics, 22 (1951), pp. 79–86.

[67] S. M. Landow, A. Gjelsvik, and M. A. Weinstock, *Mortality burden and prognosis of thin melanomas overall and by subcategory of thickness, seer registry data, 1992-2013*, Journal of the American Academy of Dermatology, 76 (2017), pp. 258–263.

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

[69] ——, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE, 86 (1998), pp. 2278–2324.

[70] Y. LeCun, F. J. Huang, and L. Bottou, *Learning methods for generic object recognition with invariance to pose and lighting*, Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2 (2004), pp. II–104 Vol.2.

[71] S. H. Lee and H.-J. Jang, *Deep learning-based prediction of molecular cancer biomarkers from tissue slides: A new tool for precision oncology*, Clinical and Molecular Hepatology, (2022).

[72] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, *Multilayer feedforward networks with a nonpolynomial activation function can approximate any function*, Neural networks, 6 (1993), pp. 861–867.

[73] A. Li, X. Li, W. Li, X. Yu, M. Qi, and D. Li, *Application of deep learning on the prognosis of cutaneous melanoma based on full scan pathology images*, BioMed Research International, 2022 (2022).

[74] M. Li, T. Zhang, Y. Chen, and A. J. Smola, *Efficient mini-batch training for stochastic optimization*, in Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, 2014, pp. 661–670.

[75] Y. Liu and J. Xu, *High-resolution microscopy for imaging cancer pathobiology*, Current pathobiology reports, 7 (2019), pp. 85–96.

[76] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, *Rethinking the value of network pruning*, arXiv preprint arXiv:1810.05270, (2018).

[77] J. J. Luke, P. Rutkowski, P. Queirolo, M. Del Vecchio, J. Mackiewicz, V. Chiarion-Sileni, L. de la Cruz Merino, M. A. Khattak, D. Schadendorf, G. V. Long, et al., *Pembrolizumab*

versus placebo as adjuvant therapy in completely resected stage iib or iic melanoma (keynote-716): a randomised, double-blind, phase 3 trial, The Lancet, 399 (2022), pp. 1718–1729.

[78] D. Mandic and J. Chambers, *Recurrent neural networks for prediction: learning algorithms, architectures and stability*, Wiley, 2001.

[79] S. Massaroli, M. Poli, F. Califano, A. Faragasso, J. Park, A. Yamashita, and H. Asama, *Port–hamiltonian approach to neural network training*, in 2019 IEEE 58th Conference on Decision and Control (CDC), IEEE, 2019, pp. 6799–6806.

[80] T. Mitchell, *Machine learning*, (1997).

[81] M. R. Moore, I. D. Friesner, E. M. Rizk, B. T. Fullerton, M. Mondal, M. H. Trager, K. Mendelson, I. Chikeka, T. Kurc, R. Gupta, et al., *Automated digital til analysis (adta) adds prognostic value to standard assessment of depth and ulceration in primary melanoma*, Scientific reports, 11 (2021), pp. 1–11.

[82] N. Morgan and H. Bourlard, *Generalization and parameter estimation in feedforward nets: Some experiments*, Advances in neural information processing systems, 2 (1989).

[83] R. Nakano, J. Hilton, S. Balaji, J. Wu, L. Ouyang, C. Kim, C. Hesse, S. Jain, V. Kosaraju, W. Saunders, et al., *Webgpt: Browser-assisted question-answering with human feedback*, arXiv preprint arXiv:2112.09332, (2021).

[84] Y. Nesterov, *A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2)*, in Doklady an ussr, vol. 269, 1983, pp. 543–547.

[85] P. Ochs, *Local convergence of the heavy-ball method and iPiano for non-convex optimization*, Journal of Optimization Theory and Applications, 177 (2018), pp. 153–180.

[86] P. Ochs, Y. Chen, T. Brox, and T. Pock, *iPiano: Inertial proximal algorithm for non-convex optimization*, SIAM Journal on Imaging Sciences, 7 (2014), pp. 1388–1419.

[87] P. Ochs and T. Pock, *Adaptive Fista for non-convex optimization*, SIAM Journal on Optimization, 29 (2019), pp. 2482–2503.

[88] F. of life institute, *Pause giant ai experiments: An open letter*. `https://futureoflife.org/open-letter/pause-giant-ai-experiments/`. Accessed: 2023-04-18.

[89] 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, Curran Associates, Inc., 2019, pp. 8024–8035.

[90] R. Patil and S. Bellary, *Machine learning approach in melanoma cancer stage detection*, Journal of King Saud University-Computer and Information Sciences, 34 (2022), pp. 3285–3293.

[91] D. L. Pham, C. Xu, and J. L. Prince, *Current methods in medical image segmentation*, Annual review of biomedical engineering, 2 (2000), pp. 315–337.

[92] M. Poli, S. Massaroli, A. Yamashita, H. Asama, and J. Park, *Port-hamiltonian gradient flows*, in ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations, 2020.

[93] B. Polyack, *Some methods of speeding up the convergence of iterative methods*, Z. Vylist Math. Fiz., 4 (1964), pp. 1–17.

[94] S. Pölsterl, *Survival analysis for deep learning, survival-cnn-estimator*. `https://github.com/sebp/survival-cnn-estimator/blob/master/tutorial_tf2.ipynb`, 2020.

[95] T. Qaiser, C.-Y. Lee, M. Vandenberghe, J. Yeh, M. A. Gavrielides, J. Hipp, M. Scott, and J. Reischl, *Usability of deep learning and h&e images predict disease outcome-emerging tool to optimize clinical trials*, NPJ precision oncology, 6 (2022), pp. 1–12.

[96] N. Qian, *On the momentum term in gradient descent learning algorithms*, Neural networks, 12 (1999), pp. 145–151.

[97] M. Reiners, K. Klamroth, F. Heldmann, and M. Stiglmayr, *Efficient and sparse neural networks by pruning weights in a multiobjective learning approach*, Computers & Operations Research, 141 (2022), p. 105676.

[98] S. Ren, K. He, R. Girshick, and J. Sun, *Faster r-cnn: Towards real-time object detection with region proposal networks*, Advances in neural information processing systems, 28 (2015).

[99] S. M. Riad, *The deconvolution problem: An overview*, Proceedings of the IEEE, 74 (1986), pp. 82–85.

[100] E. M. Rizk, A. M. Seffens, M. H. Trager, M. R. Moore, L. J. Geskin, R. D. Gartrell-Corrado, W. Wong, and Y. M. Saenger, *Biomarkers predictive of survival and response to immune checkpoint inhibitors in melanoma*, American journal of clinical dermatology, 21 (2020), pp. 1–11.

[101] H. Robbins and S. Monro, *A stochastic approximation method*, The annals of mathematical statistics, (1951), pp. 400–407.

[102] O. Ronneberger, P. Fischer, and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, in Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18, Springer, 2015, pp. 234–241.

[103] F. Rosenblatt, *The perceptron: a probabilistic model for information storage and organization in the brain.*, Psychological review, 65 (1958), p. 386.

[104] R. Rubinstein, *The cross-entropy method for combinatorial and continuous optimization*, Methodology and computing in applied probability, 1 (1999), pp. 127–190.

[105] R. Y. Rubinstein, *Optimization of computer simulation models with rare events*, European Journal of Operational Research, 99 (1997), pp. 89–112.

[106] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning representations by back-propagating errors*, nature, 323 (1986), pp. 533–536.

[107] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al., *Imagenet large scale visual recognition challenge*, International journal of computer vision, 115 (2015), pp. 211–252.

[108] D. Saad, *Online algorithms and stochastic approximations*, Online Learning, 5 (1998), p. 6.

[109] M. Schmid, M. N. Wright, and A. Ziegler, *On the use of harrell's c for clinical risk prediction via random survival forests*, Expert Systems with Applications, 63 (2016), pp. 450–459.

[110] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*, Cambridge university press, 2014.

[111] F. Sheikhzadeh, M. Guillaud, and R. K. Ward, *Automatic labeling of molecular biomarkers of whole slide immunohistochemistry images using fully convolutional networks*, arXiv preprint arXiv:1612.09420, (2016).

[112] C. Shorten and T. M. Khoshgoftaar, *A survey on image data augmentation for deep learning*, Journal of big data, 6 (2019), pp. 1–48.

[113] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2014.

[114] D. Singh and B. Singh, *Investigating the impact of data normalization on classification performance*, Applied Soft Computing, 97 (2020), p. 105524.

[115] A. Sobester, A. Forrester, and A. Keane, *Engineering design via surrogate modelling: a practical guide*, John Wiley & Sons, 2008.

[116] S. Srinivas and R. V. Babu, *Data-free parameter pruning for deep neural networks*, arXiv preprint arXiv:1507.06149, (2015).

[117] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, *Dropout: a simple way to prevent neural networks from overfitting*, The journal of machine learning research, 15 (2014), pp. 1929–1958.

[118] Tagesschau, *Experten fordern pause bei ki-entwicklung.* `https://www.tagesschau.de/wissen/musk-tech-pause-ki-entwicklung-101.html`. Accessed: 2023-04-18.

[119] T. Tieleman, G. Hinton, et al., *Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude*, COURSERA: Neural networks for machine learning, 4 (2012), pp. 26–31.

[120] A. Van Der Schaft, D. Jeltsema, et al., *Port-hamiltonian systems theory: An introductory overview*, Foundations and Trends® in Systems and Control, 1 (2014), pp. 173–378.

[121] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, *Attention is all you need*, Advances in neural information processing systems, 30 (2017).

[122] Y. Wang, X. Zhang, L. Xie, J. Zhou, H. Su, B. Zhang, and X. Hu, *Pruning from scratch*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, 2020, pp. 12273–12280.

[123] P. J. Werbos, *Applications of advances in nonlinear sensitivity analysis*, in System Modeling and Optimization: Proceedings of the 10th IFIP Conference New York City, USA, August 31–September 4, 1981, Springer, 2005, pp. 762–770.

[124] S. C. Wong, A. Gatt, V. Stamatescu, and M. D. McDonnell, *Understanding data augmentation for classification: when to warp?*, in 2016 international conference on digital image computing: techniques and applications (DICTA), IEEE, 2016, pp. 1–6.

[125] S. Wright, J. Nocedal, et al., *Numerical optimization*, Springer Science, 35 (1999), p. 7.

[126] H. Xiao, K. Rasul, and R. Vollgraf, *Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms*, CoRR, abs/1708.07747 (2017).

[127] Q. Yao, M. Wang, Y. Chen, W. Dai, Y.-F. Li, W.-W. Tu, Q. Yang, and Y. Yu, *Taking human out of learning applications: A survey on automated machine learning*, arXiv preprint arXiv:1810.13306, (2018).

[128] M. D. Zeiler and R. Fergus, *Visualizing and understanding convolutional networks*, in Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13, Springer, 2014, pp. 818–833.

[129] Y.-T. Zhou and R. Chellappa, *Computation of optical flow using a neural network.*, in ICNN, 1988, pp. 71–78.

[130] X. Zhu, J. Yao, and J. Huang, *Deep convolutional neural network for survival analysis with pathological images*, in 2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), IEEE, 2016, pp. 544–547.

[131] B. Zoph and Q. V. Le, *Neural architecture search with reinforcement learning*, arXiv preprint arXiv:1611.01578, (2016).

[132] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, *Learning transferable architectures for scalable image recognition*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 8697–8710.