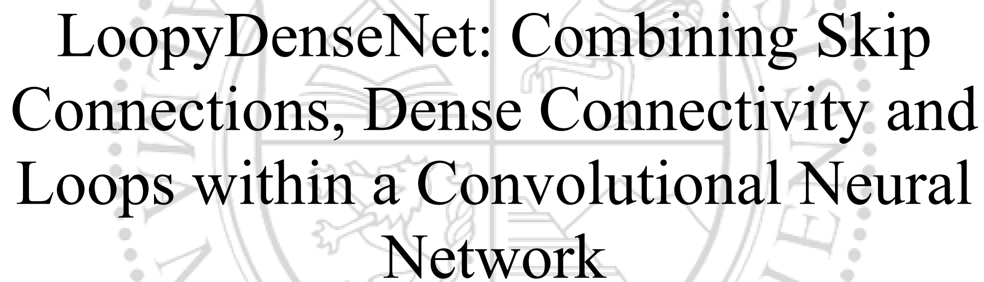
Chair of Information Technology

Master's Thesis



LoopyDenseNet: Combining Skip Connections, Dense Connectivity and Loops within a Convolutional Neural Network

Peter Niederl, BSc

May 2022

MONTANUNIVERSITÄT LEOBEN
www.unileoben.ac.at

AFFIDAVIT

I declare on oath that I wrote this thesis independently, did not use other than the specified sources and aids, and did not otherwise use any unauthorized aids.

I declare that I have read, understood, and complied with the guidelines of the senate of the Montanuniversität Leoben for "Good Scientific Practice".

Furthermore, I declare that the electronic and printed version of the submitted thesis are identical, both, formally and with regard to content.

Date 20.05.2022

Signature Author
Peter Niederl

MONTANUNIVERSITÄT LEOBEN
www.unileoben.ac.at

BSc
Peter Niederl
St. Marxen 46
9122 St. Kanzian am Klopeiner See

To the Dean of graduate Studies of the Montanuniversitaet Leoben

Declaration of Approval for the Digital Publication of Scientific Theses

I am aware that the thesis entitled "LoopyDenseNet: Combining Skip Connections, Dense Connectivity and Loops within a Convolutional Neural Network" will be subject to a plagiarism assessment and may be stored by Montanuniversität Leoben for an unlimited period of time.

I agree that the University Library of Montanuniversität Leoben may publish the thesis open access in the World Wide Web. For embargoed theses this will be done after the embargo expires.

Note: in case you refuse the open access publication in the World Wide Web, the thesis will only be published in printed form (after a possible embargo has expired) in the University Library (dissertations also in the Austrian National Library).

I hereby agree with the open access publication of my thesis on the World Wide Web:

X Yes

O No

Date 20.05.2022

Signature Author

Acknowledgement

First of all, I would like to thank my thesis advisor Univ.-Prof. Dipl.-Ing. Dr.techn. Peter Auer of the Chair of Information Technology at the University of Leoben. I am gratefully indebted for his valuable advices on this thesis and for his guidance.

Additionally, I must express my very profound gratitude to my family and to my girlfriend Nina for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Abstract

Convolutional neural networks (CNNs) have achieved remarkable results in visual object recognition. By using convolutional layers, filters are trained in order to detect distinct features, which enable the network to correctly classify different objects. A traditional CNN follows a hierarchical structure, where every layer is used exactly once. In this work a new network architecture is proposed which utilizes convolutional layers multiple times by looping them. By doing so the following convolutional layers receive more refined feature-maps of different origins. It is shown experimentally, that looping convolutional operations can have a shifting-effect on the detected features, such that the network focuses on certain features in certain regions of the input, depending on the filter. Furthermore, a new type of skip connection is presented, which makes more information available at the flatten layer and is strengthening feature propagation. By looping convolutions the network is very parameter efficient, while still being able to create divers feature-maps. In order to build deeper models with the proposed network architecture some methods are given in order to reduce computational costs and parameters. The proposed network architecture is compared to the traditional CNN architecture on 5 different datasets (MNIST, Fashion-MNIST, CIFAR-10, Fruits-360, Hand gesture), showing superior or similar results on most datasets while having comparable computational costs.

Kurzfassung

Convolutional Neural Networks haben bemerkenswerte Ergebnisse bei der visuellen Objekterkennung erzielt. Mit Hilfe von Convolutional Layers wurden Netzwerke trainiert, um bestimmte Features zu erkennen, die es dem Netz ermöglichen, verschiedene Objekte korrekt zu klassifizieren. Ein traditionelles CNN folgt einer hierarchischen Struktur, bei der jede Schicht genau einmal verwendet wird. In dieser Arbeit wird eine neue Netzwerkarchitektur vorgestellt, bei der die Convolutional Layers mehrfach verwendet werden, indem sie in Schleifen zum Einsatz kommen. Auf diese Weise erhalten die nachfolgenden Convolutional Layers verfeinerte Feature-Maps unterschiedlicher Herkunft. Es wird experimentell gezeigt, dass das wiederholte Anwenden von Convolutional Operationen einen Verschiebungseffekt auf die erkannten Features haben kann, sodass sich das Netzwerk je nach Filter auf bestimmte Merkmale in bestimmten Regionen des Inputs konzentriert. Darüber hinaus wird eine neue Art von Skip-Connection verwendet, die mehr Informationen auf der Flatten-Schicht verfügbar macht und die Feature-Propagation verstärkt. Durch die Verwendung von Convolutional-Loops ist das Netzwerk sehr parameter-effizient und kann dennoch vielfältige Feature-Maps erstellen. Um tiefere Modelle mit der vorgeschlagenen Netzwerkarchitektur zu bauen, werden einige Methoden angegeben, um den Rechenaufwand und die Parameter zu reduzieren. Die vorgeschlagene Netzwerkarchitektur wird mit der traditionellen CNN-Architektur auf 5 verschiedenen Datensätzen (MNIST, Fashion-MNIST, CIFAR-10, Fruits-360, Hand gesture) verglichen und zeigt bessere oder ähnliche Ergebnisse bei den meisten Datensätzen bei vergleichbarem Rechenaufwand.

Contents

Affidavit **I**

Declaration of Approval **II**

Acknowledgement **III**

Abstract **IV**

Kurzfassung **V**

Table of content **VI**

1 Introduction **1**

1.1 Relevance of the topic . . . 1

1.2 Research Questions . . . 1

1.2.1 Research Question 1 . . . 1

1.2.2 Research Question 2 . . . 2

1.2.3 Research Question 3 . . . 2

1.3 The structure of the work . . . 2

2 Related work **4**

2.1 DenseNets . . . 4

2.2 Loopy Neural Nets . . . 6

2.2.1 Advantages of loops in neural networks . . . 8

2.3 Recurrent Neural Networks . . . 8

2.4 Recurrent Convolutional Neural Networks . . . 10

2.5 FractalNet . . . 11

2.6 Highway Networks and ResNets . . . 12

3 Introduction to own work **13**

3.1 Flat-Skips . . . 13

3.1.1 Benefits of Flat-Skips . . . 14

3.2 Backpropagation with Flat-Skips . . . 15

3.3 Modified DenseNet . . . 16

3.3.1 Adaptive pooling . . . 17

3.4 LoopyDenseNet . . . 19

3.4.1 Forward propagation in LoopyDenseNet . . . 19

3.4.2 Backpropagation of a LoopyDenseNet 24
3.4.3 Effects of loops in the LoopyDenseNet 25
3.4.4 Looping limit 30
3.4.5 Bottleneck layer 32

4 Datasets 34
4.1 MNIST 34
4.1.1 Data Augmentation and preparation on MNIST 34
4.2 Fashion-MNIST 36
4.3 Fruits-360 37
4.4 Hand gesture 37
4.4.1 Preprocessing of the images 38
4.4.2 Splitting the data 38
4.5 CIFAR-10 38

5 Experiment 40
5.1 Base model 40
5.2 Model setup and model selection 43

6 Results 50
6.1 Experiment on the MNIST dataset 50
6.2 Experiment on the Fashion-MNIST dataset 52
6.3 Experiment on the hand gesture dataset 54
6.4 Experiment on the Fruits-360 dataset 57
6.5 Experiment on the CIFAR-10 dataset 59
6.6 Summary of the results 62

7 Prospects of LoopyDenseNets 64
7.1 Deep LoopyDenseNets 64
7.2 LDNs without pooling 67
7.3 LDNs with bigger filter sizes 68

8 Conclusion 71
8.1 Research Question 1 - Flat-Skips 71
8.2 Research Question 2 - Modified DenseNet 71
8.3 Research Question 3 - LoopyDenseNet 72

References 74

List of Tables 82

List of Figures 84

1. Introduction

1.1 Relevance of the topic

Since the introduction of the LeNet5 in 1998 [36], more complex network architectures for visual object recognition have been developed. While the AlexNet, which consists of 8 layers and has about 60M parameters [6] [34], and the VGG-19, which consists of 19 layers and is twice the size of the AlexNet [65], were stacking more and more layers on top of each other, more sophisticated architectures were proposed. In order to get deeper networks ResNets, Highway Networks and FractalNets use special architectures, skip connections of different length and residual learning [26] [35] [67]. Despite a lot of effort in the investigation of new and more powerful architectures there is still room for further investigation. One of which is the use of convolutional loops in object recognition. In the paper ”Loopy Neural Nets: Imitating Feedback Loops in the Human Brain” Caswell et al. use feedback loops within the convolutional part of the network, showing that looping the convolutional part can indeed improve the expressive capacity of the model [29]. Furthermore, the Recurrent Convolutional Neural Network uses convolutional layers recursively and by doing so can create a network with arbitrary depth while having a constant number of parameters. Those Recurrent Convolutional Neural Network achieve high performance on many popular benchmarking datasets while being extremely parameter efficient [44]. However, there are still a lot of open questions regarding loops. What effect do convolutional loops have on feature-maps? How can loops be integrated in the network architecture? How many loops should be executed? How many layers should be looped? Is looping convolutions advantageous for the performance? The proposed network architecture is the attempt to create a model which can directly be applied to a traditional CNN and loops single convolutional layers. When applying this architecture on an ordinary CNN the number of loops are defined by the depth of the model. This architecture represents a possible way on how to implement convolutional loops in a CNN.

1.2 Research Questions

1.2.1 Research Question 1

Skip connections have proven to be an important tool to improve the performance of neural networks. They appear in several different forms. In DenseNets skip connections of different length connect every convolutional layer with every following convolutional layer

by concatenating their feature-maps [26]. Highway networks have special gating units that can transfer information over many layers, making it possible to create extremely deep networks which can be trained by stochastic gradient descent [67]. The skip connections of those architectures have all in common, that they connect layers within the convolutional part of the network. However, what if convolutional layers get connected with the flatten layer? Can a direct connection between each convolutional layer and the flatten layer be beneficial for the performance of such a network?

1.2.2 Research Question 2

DenseNets have shown, that skip connections which connect every convolutional layer with every following convolutional layer can massively improve the performance of deep neural networks. This is called a dense connectivity pattern [26]. What happens when this method is applied on small sized networks consisting of few convolutional layers? Furthermore, can this method also benefit from the skip connection mentioned in the first research question? How does such network compare to ordinary CNNs with comparable computational costs?

1.2.3 Research Question 3

The traditional CNN architecture consists of hierarchically ordered layers. The first layer of the network receives an input and then computes an output which is fed to the second layer. The second layer does the same thing. It receives the input from the first layer and delivers its output to the third layer. This is done until the output layer of the network is reached. This is commonly known as the feedforward pass. Within this feedforward pass every layer gets utilized exactly once. However, what happens when a layer gets used multiple times at different times during the feedforward pass in form of a convolutional loop? How can such a behavior be integrated in the traditional CNN architecture? How does it perform in combination with the skip connection which is covered in the first research question?

1.3 The structure of the work

This work consists of 8 chapters. The introduction outlines the research questions explored in this thesis and the relevance of this work. In the second chapter related work is outlined, which heavily influenced the design of the proposed methods and the network architecture. After that, a new type of skip connection gets proposed, as well as, a modified version of the DenseNet gets explained [26]. Those models should serve as a comparison. Furthermore, the proposed network architecture is introduced. By doing so the feedforward and the the backpropagation are explored. Possible variations of the

network architecture and parameters are discussed.
After the new network architecture is introduced the datasets on which the different models are evaluated on will be explained. In addition to the description of the datasets the data augmentation which is used during training is described. In the next chapter the experiment and the evaluation procedure gets outlined. The results of the experiment is discussed afterwards. Since the computational resources of this work are very limited, a prospect is given in which areas in the context of the proposed network architecture further investigations might be interesting. This includes ideas on how to create deeper models. In the end the final results are summarized and the research questions get answered.

2. Related work

The proposed network architecture was inspired by many different models and concepts, which have shown incredible results in computer vision tasks. In this chapter those architectures and concepts are explained and summarized. Furthermore, an explanation is given on how these architectures affected the proposed neural network architecture.

2.1 DenseNets

DenseNets were introduced by Huang et al in 2016 and obtained significant improvements at several benchmark tasks at that time. They embrace the concept of having a dense connectivity pattern within the convolutional part of the network. In comparison to traditional convolutional neural networks where every convolutional layer is just connected to its predecessor and its successor, in DenseNets convolutional layers are connected to every previous and every following convolutional layer. This means, that a DenseNets with L layers have $\frac{L(L+1)}{2}$ connections, while a normal CNN with L layers just has L connections. Those additional connections are created by feature-map concatenation, which means that the input feature-maps of a convolutional layer is a collection of the feature-maps which were generated by all preceding layers. Since a normal convolutional operation changes the dimensions of the feature-maps, zero-padding was used in order to keep the dimensions of the feature-maps the same. That way feature-maps of different layers can be concatenated. By doing so the depth of the input feature-maps increases for every following layer. In the following figure the concept of a dense connectivity pattern is illustrated more clearly [26]:

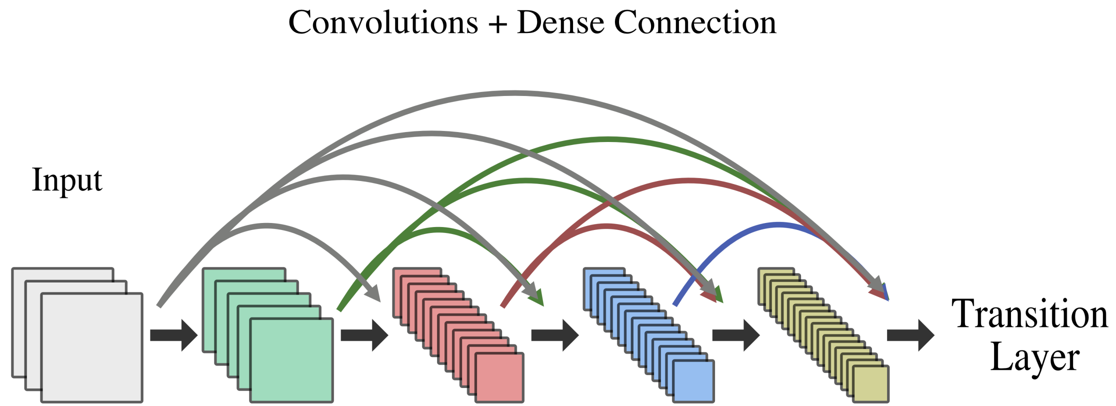


Figure 2.1: A 5 layered dense block: The input of a layer consists of the output of all previous layers. Each layer is connected to all following layers [26].

Since the feature-map dimensions do not change when using convolutional operations the authors claim that it is still very important to have pooling layers to reduce the size of the feature-maps. To accomplish this the network was divided in smaller blocks, which the author called dense blocks. Within a block the generated feature-maps share the same dimensions and therefore can be concatenated. Between those dense blocks transition layers were used, which consist of a 1×1 convolutional layer and a 2×2 pooling layer. Since the feature-maps at the end of a dense block consists of all the feature-maps generated in this particular dense block, a 1×1 helps to reduce the number of feature-maps and decrease computational costs. The number of feature-maps that get generated in a dense block depends on the *growth rate*. This rate defines the number of feature-maps that get generated in a convolutional layer. When using multiple convolutional layers during a dense block, the number of feature-maps can get very large. In order to be more computational efficient, bottleneck layers can be used before the actual convolution. This bottleneck layer is a 1×1 convolution, which reduces the number of feature-maps and has the same purpose as the 1×1 convolution in the transition layer. DenseNets are also less prone to overfit since they are very parameter efficient, especially when incorporating bottleneck layers before the actual convolution. The dense connectivity pattern creates a form of collective knowledge, as called from the authors. Since every layer has all the information available which was generated by all the previous layers. By doing so many more paths were created, which eliminates the vanishing gradient problem. This might also be the key to the DenseNet's success [26].
DenseNets have heavily influenced the design of the proposed neural network architecture, since it also includes some form of a dense connectivity pattern. Additionally, LoopyDenseNets use special skip connections, which also create a collective knowledge. However, those skip connections are operating differently then the skip connections used in the DenseNet. Furthermore, the LoopyDenseNet uses feature-map concatenation in

order to concatenate feature-maps of different layers, just like the DenseNet does.

2.2 Loopy Neural Nets

In the paper ”Loopy Neural Nets: Imitating Feedback Loops in the Human Brain” the authors Caswell et al. propose a new network architecture, which mimics feedback loops within the brain. Although artificial neural networks are the attempt to imitate the human brain, which according to neuroscience has many feedback loops, traditional neural networks are acyclic computational graphs. Therefore the principle architecture of artificial neural networks differs from its real model. With the proposed loopy neural network model the authors investigate on the effects of having feedback loops in an artificial neural network, which is supposed to approximate the functioning of a real human brain [29].
In order to be able to include loops within the neural network the feature-maps, which are generated at the end of the convolutional part, undergo element-wise addition or multiplication with the input of the network, before feeding it to the first convolutional layer again. To make this possible the feature-maps need to be compressed to the size of the input. This is done via a 3×3 convolution. Assuming that the input is a colored image, three 3×3 convolutions have to be applied to the feature-maps, before they get merged with the input. In comparison to an ordinary convolutional neural network where each convolutional layer gets used just once, in a loopy neural network a convolutional neural network can be utilized multiple times at different times during calculation. To avoid an infinite loop in the network the authors introduced the unroll factor, which defines the number of loops that will be executed [29].

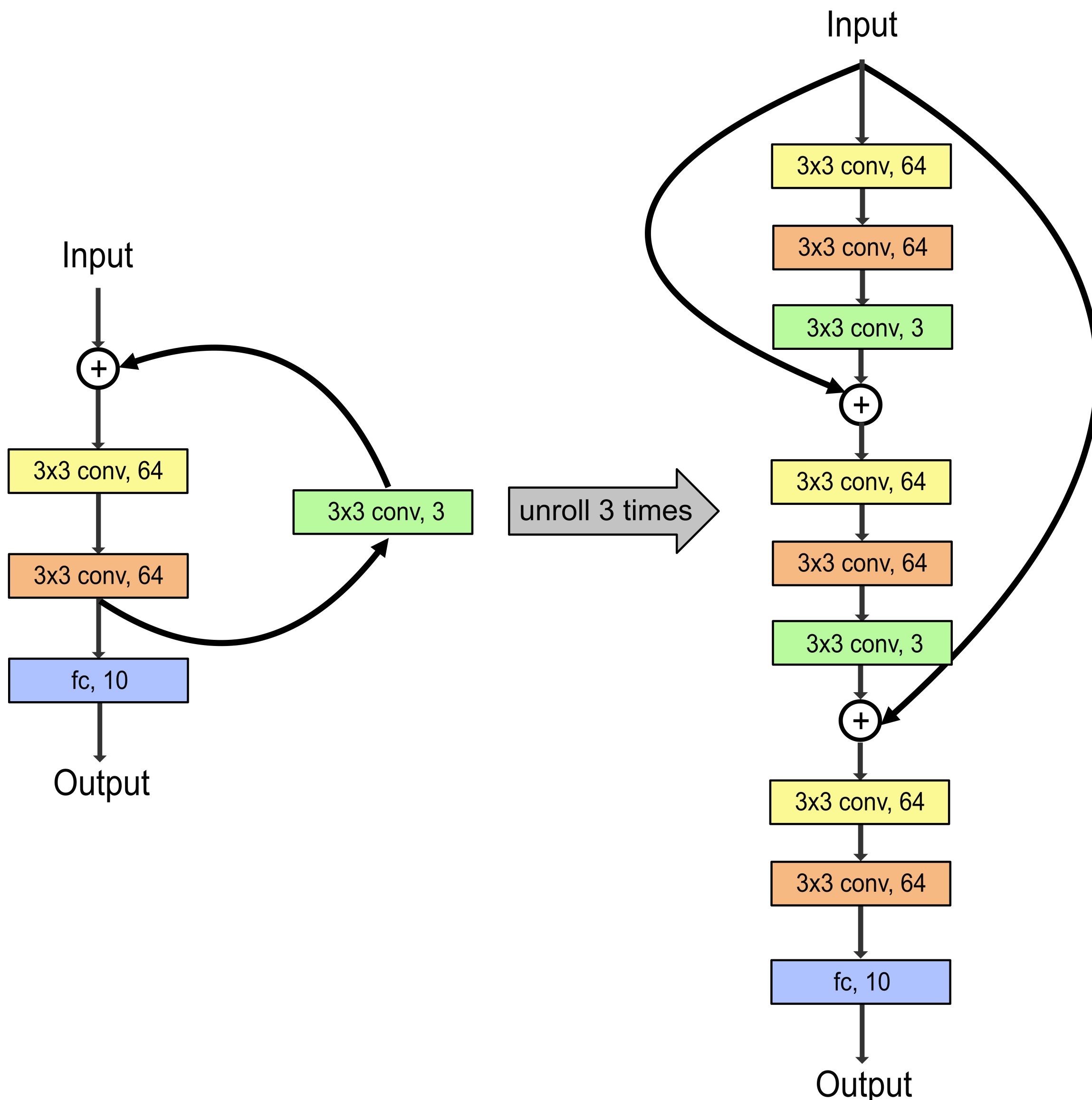


Figure 2.2: A loopy neural network with an unroll factor of 3. On the right is the unrolled network [29].

When knowing the number of loops, the network can be represented as an unrolled neural network which reuses the same layers again as can be seen in the figure 2.2. Therefore the forward propagation for a loopy neural network is the same as for an ordinary neural network, however, it has to be unrolled first. Similarly to the forward propagation the backpropagation is performed in the common manner as well. Nevertheless, since the same layers were used multiple times the final gradient for a convolutional layer is the sum of all the individual gradients of each loop [29].
In the original work the loopy neural network was compared to the corresponding deep neural network, as well as the non-loopy version of the network on the CIFAR-10 and MNIST datasets. It managed to outperform the non-loopy version, as well as the deep representation of the loopy neural network on the CIFAR-10 dataset. Furthermore, the loopy neural network met the performance of the non-loopy and the deep models on the

MNIST dataset, while having the same amount of parameters as the plain network and almost three times fewer parameters compared to the deep network, since a unroll factor of three was chosen. This indicates, that loopy neural networks can have the same or better expressive capacities as their deep representations [29].
The LoopyDenseNet also incorporates the concept of loops in its architecture. However, it uses loops in order to generate a dense connectivity pattern by only looping single layers and not the whole convolutional part of the network. This process is described in further detail in section 3.4.1.

2.2.1 Advantages of loops in neural networks

One aspect of using feedback loops in neural networks is, that information from later layers get fed to earlier layers. This should enable the network to make a more sophisticated choice for the weights of earlier layers. Furthermore, loopy neural networks are very parameter efficient. Depending on the number of unrolls, a loopy neural network may represent a deep convolutional neural network, while having much fewer parameters, because of the filter reuse. Additionally a loopy neural network is easier to train then its deep representation. The deep representation of a 4 layered loopy neural network, which gets looped 3 times, would consist of 12 layers which is harder to train, because of the vanishing gradient problem. When the derivatives are large then with each layer the gradient will grow until it explodes and learning will not be possible. On the other side when the derivative is small, then the gradient will shrink with each layer to the point backpropagation has basically no effect on the parameters of the network [52] [71]. Because of that training a smaller network which uses loops can ease learning. This is especially true since the parameters of each layer get updated multiple times. Caswell et al. also state, that the looped output combined with the input image results in an attention map, indicating, that loops learn to weight the important regions of the image [29].

2.3 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) differentiate themselves from Feedforward Neural Networks (FNN) since they incorporate loops within their architecture. While CNNs have a hierarchical structure where the output of one layer is fed directly to the next layer, RNNs have cycles, which means, that they pass information back to themselves. Traditionally this property is used to feed sequential data to the network, since it is able to account for previous inputs. Because of that RNNs are used in many different domains, such as language modeling, speech recognition and image description generation. The following figure should demonstrate the difference between a Feedforward Neural Network and a Recurrent Neural Network [62] [11].

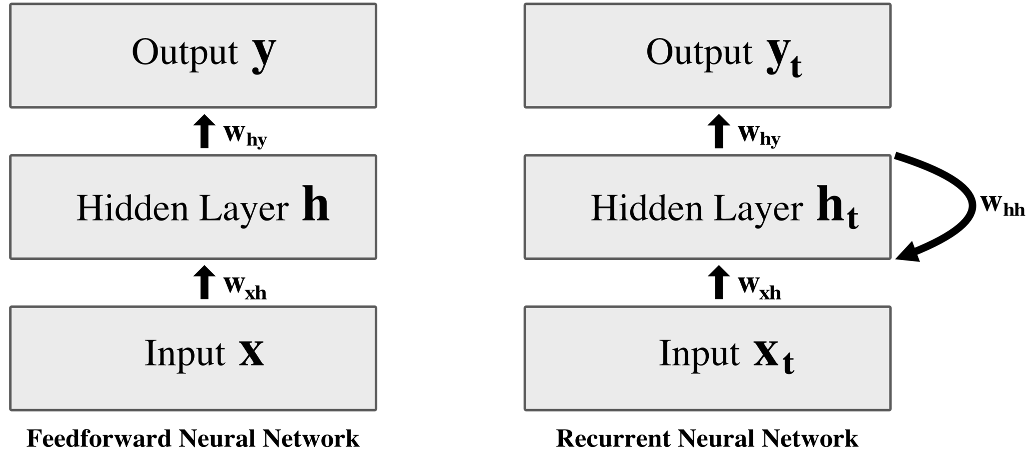


Figure 2.3: Comparison between a Feedforward Neural Network and a Recurrent Neural Network [62].

When unrolling the Recurrent Neural Network, the following structure can be seen.

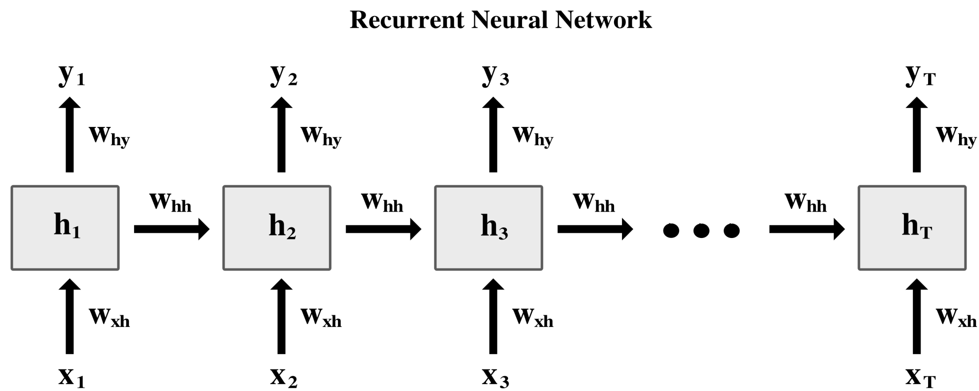


Figure 2.4: Recurrent Neural Network unrolled [62].

In order to see the differences between FNNs and RNNs more clearly, equations were given, which calculate the different variables. The hidden variable of the Feedforward Neural Network is calculated by:

$$h = \sigma(xw_{xh} + b_h) \tag{2.1}$$

and the output is calculated with the following equation:

$$y = \sigma(xw_{hy} + b_y). \tag{2.2}$$

In this equation σ stands for any activation function. Usually the sigmoid or tanh functions where used in the RNNs, while in CNNs the ReLU function is most often used [22] [38] [62] [11]. In comparison to that, the calculation of the hidden state in the RNN looks as follows:

$$h^t = \sigma(x^t w_{xh} + h^{t-1} w_{hh} + b_h) \tag{2.3}$$

This time the hidden variable at time step t is not only dependent on the input at time step t , but also on the previous hidden state h^{t-1}. For completion the output variable at time step t is calculated as follows:

$$y^t = \sigma(h^t w_{hy} + b_y) \tag{2.4}$$

Here it is possible to see another key difference between FNN and RNN. The RNN can calculate different outputs over time. Therefore the total loss of a RNN with T time steps is the sum of all losses over all outputs. The loss function can be chosen for the specific problem that needs to be addressed [62].

$$L = \sum_{t=1}^{T} L^t \tag{2.5}$$

Not only the feedforward pass is different from a FNN to a RNN, also the backpropagation is executed differently. While the weights of a convolutional layer are only used once in a FNN, in the RNN the weights can be used an arbitrary amount of times. This depends on the number of unrolls. In order to consider this Backpropagation Through Time (BPTT) has to be used. During backpropagation the weights w_{xh}, w_{hh} and w_{hy} have to be updated in respect to the calculated loss. For the proposed network architecture the update for w_{hh} is especially interesting and would look like this:

$$\frac{\partial L^t}{\partial w_{hh}} = \frac{\partial L^t}{\partial y^t}\frac{\partial y^t}{\partial h^t}\sum_{k=1}^{t}\frac{\partial h^t}{\partial h^k}\frac{\partial h^k}{\partial w_{hh}}. \tag{2.6}$$

Important to node is, that w_{hh} is used multiple times to calculate the current state h^t, as well as all previous states [62].
The equation 2.6 will be important when doing backpropagation in the LoopyDenseNet, since it contains looped convolutions. In fact, standard backpropagation and Backpropagatin Through Time were used in the LoopyDenseNet. This will be explained in more detail in the section 3.4.2.

2.4 Recurrent Convolutional Neural Networks

Recurrent Convolutional Neural Networks (RCNN) were introduced by Liang et al. in 2015 and can be seen as a combination of a Recurrent Neural Network and a Convolutional Neural Network. While it has recurrent layers like a recurrent multilayered perceptron (RMLP), the layers itself are not fully connected, however, share local connections in the form of convolutions [16] [36]. They incorporate recurrent connections inspired by the Recurrent Neural Network for static object recognition and classification, which were called recurrent convolutional layers. The authors claim, that those recurrent convolutional lay-

ers reinforce the ability of the network to incorporate context information. Furthermore, the RCNN is extremely parameter efficient, while it can be arbitrary deep depending on the number of unrolls [44]. The following graphic should illustrate the difference between the common Convolutional Neural Network, the Recurrent Multi-Layered Perceptron and the Recurrent Convolutional Neural Network.

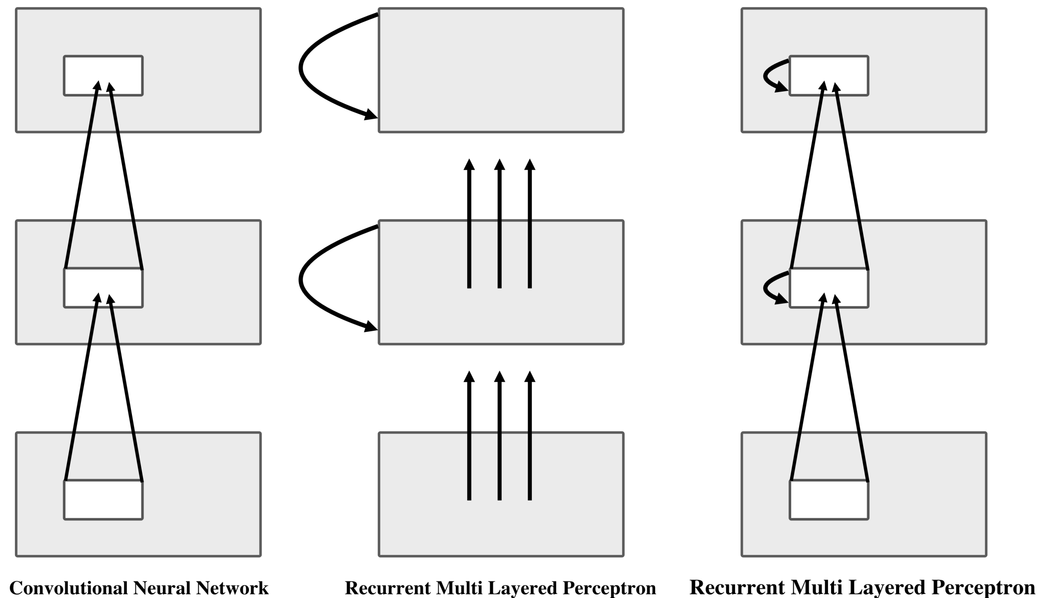


Figure 2.5: Comparison between the Convolutional Neural Network, the Recurrent Multi-Layered Perceptron and the Recurrent Convolutional Neural Network [44].

The proposed neural network also loops convolutional layers, similar to the RCNN. While a looped recursive layer receives two inputs, the original input to this layer coming from the previous layer and the state of the previous loop, the LoopyDenseNet only uses the previous state as input.

2.5 FractalNet

FractalNet is a network architecture, which was introduced in 2017 by Larsson et al. This architecture consists of many different subpaths with alternating length and is based on self-similarity. FractalNets get generated by using a simple expansion rule, creating many paths of different length. What is special about FractalNets is that each internal signal is processed by a filter and a nonlinearity before being passed to the following layer. This distinguishes FractalNets from other network architectures like ResNets [21]. Despite not using residual representations very deep Fractal Networks achieve similar results as popular residual networks, suggesting that the ability to transition from a shallow to a deep network may be the key to achieve high performance [35], rather than learning residuals.

All the internal signals of the proposed LoopyDenseNets are also processed by a filter, followed by a nonlinearity. It does not use residual representations like ResNets.

2.6 Highway Networks and ResNets

Increasing the depth of neural networks has shown to be crucial to achieve better performance. However, increasing the number of layers of the network makes training harder. Therefore oftentimes the performance of deeper models do not match shallower ones. Highway networks try to add highway connections, which should act as information highways and are inspired by the Long Short Term Memory of recurrent networks [25]. These highway connections, come in the form of information gates ensuring, that information can flow over several layers. They enable to train networks of tremendous size with simple stochastic gradient descent. Highway networks show, that information routing can be beneficial for training and for the performance of a neural network, since they enable larger models, which are capable of learning more complex behaviors. Models consisting of hundreds of layers can still be optimized, while plain networks of the same size fail to achieve similar results. Additionally the authors mention, that highway networks converge faster then plain networks [67] [68].
Similarly to Highway Networks, ResNets also try to benefit from the increasing depth of neural networks. However, instead of using highway connections, ResNets introduce a residual learning framework, which enables to train much deeper models. They claim, that it is easier to optimize the residual mapping then learning unreferenced functions. That way ResNets can benefit from increased depth and achieve better performance then shallower networks [21].

3. Introduction to own work

The following chapter consists of three major parts. In the first section a new type of skip connection is described, which increases the connectivity between the convolutional part of the network and its flatten layer. It can easily be applied to an ordinary CNN to increase its connectivity and create multiple paths for the information to flow. Next a modified version of the DenseNets is described, which brings the idea of a dense connectivity pattern to smaller convolutional neural networks, as well as the above introduced skip connections. This modified DenseNet should serve as a comparison to the proposed architecture and includes many concepts which influenced the design of the LoopyDenseNet architecture. In the final section the proposed network architecture with the name *LoopyDenseNet* gets presented. This model includes similar ideas as the modified DenseNet and furthermore incorporates loops in the convolutional part of the network.

3.1 Flat-Skips

DenseNets, ResNets, HighwayNets and FractalNets and many more network architectures have a similar characteristic. They all have some sort of shortcut, which transfers information from an earlier layer to a later layer and that way modifies the effective depth of the network [26] [21] [67] [35]. In this section the concept of skip connections, which directly connect each convolutional layer with the flatten layer, is discussed. First the functionality of the proposed skip connection, which in the course of the work is called *Flat-Skips*, is described. Furthermore, possible benefits of those skip connections are illustrated. In order to evaluate these benefits the performance of CNN's with and without *Flat-Skips* were compared. In comparison to other skip connections, Flat-Skips have the attributes, that they directly transfers the output of each convolutional layer to the flatten layer. This is done via feature-map concatenation. The generated feature-maps of each convolutional layer are passed to the flatten layer, where they get flattened, just like the feature-maps of the last convolutional layer of an ordinary CNN. By doing so, a form of "common knowledge" is created, which is comparable to the "common knowledge" in DenseNets [26]. As a result the flatten layer is a collection of all the feature-maps that were generated during the convolutional part of the network.

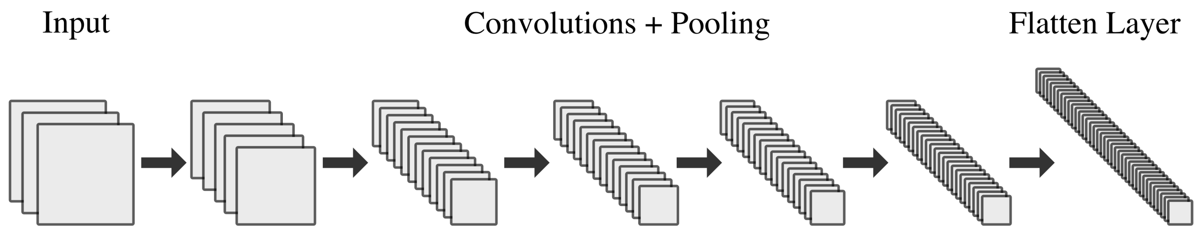


Figure 3.1: Convolutional part of an ordinary CNN: The flatten layer just consists of the feature maps of the last convolutional layer.

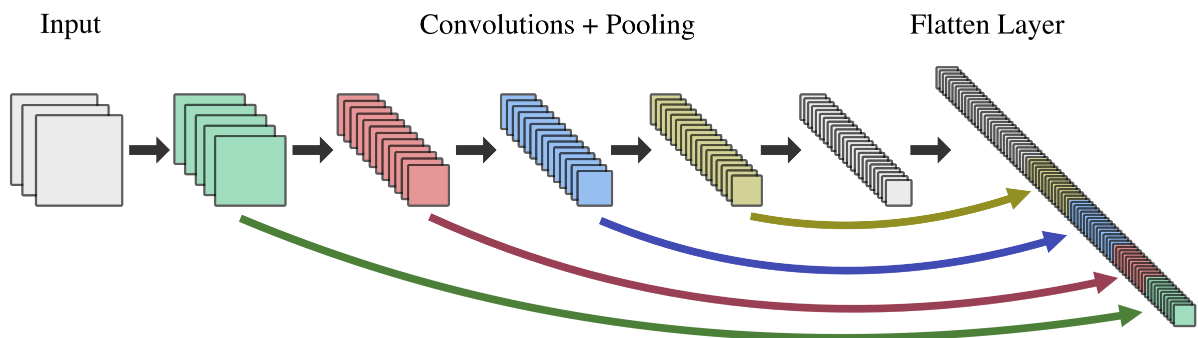


Figure 3.2: Convolutional part with Flat-Skip connections: The flatten layer consists of the concatenated feature maps of all convolutional layers.

As the figure 3.2 illustrates, using Flat-Skip connections increases the length of the flatten layer significantly. Since the feature-maps of earlier layers typically have a larger size, it will be necessary to use pooling operations before passing the feature-maps to the flatten layer in order to reduce the number of parameters. This is especially true when the flatten layer is the input layer for a fully-connected layer. However, when using global average-pooling for the final prediction, pooling might not be necessary [40]. During testing max-pooling seems to perform slightly better then average-pooling, however, for certain scenarios average-pooling might be advantageous. All the networks which are tested in the course of this work only own a single fully connected layer, which connects the flatten layer with the output layer of the network, which is making the final prediction.

3.1.1 Benefits of Flat-Skips

Flat-Skips have three major benefits. First of which is, that it makes more information available in the flatten layer. Similar to the skip connections of DenseNets, Flat-Skips concat the feature-maps of different convolutional layers [26]. Instead of using this as the input for the next convolutional layer, those feature maps get flattened and then used as input for a fully connected layer or an global average-pooling layer [40]. The underlaying idea is the same. The belief is, that also feature-maps of earlier layers include useful information for the final prediction, which in the end lead to more accurate predictions. This should be especially true, when combining the information of all feature maps of all

layers. A L layered CNN with Flat-Skips has the same amount of information available as L CNNs with 1 to L layers.
Many recent network architecture, like residual networks (ResNets), fractal networks and highway networks [26] [67] [35], have shown, that the ability to adapt the depth of a convolutional network can be crucial for its performance. In the paper [74] the authors claim, that residual networks can be seen as a collection of many paths of different length. ResNets do this by their shortcut connections, which enables to pass information through multiple layers, allowing for ResNets with hundreds of layers [21]. Fractal networks consist of interacting subpaths of different lengths, suggesting that being able to transition from shallower to deeper models during training might be crucial to the performance of a neural network [35]. Expanding CNNs with Flat-Skips is a simple way to add the ability to adapt the depth of a CNN, since those skip connections can directly shorten the depth of a CNN.
The next advantage of Flat-Skips is the improved gradient flow. By using Flat-Skips the vanishing gradient problem gets eliminated, since the gradient of the flatten layer can directly be passed to the corresponding layer [18] [15]. Furthermore, with Flat-Skips the gradient has many more paths to propagate back through the network. The advantage of these skip connections is that there are more and more paths for the gradient, the earlier the layer is. This means that the effect of the connections becomes stronger as the number of layers increases. Taking into account that the vanishing gradient problem also becomes greater as the number of layers increases, this is an advantageous property. Considering a CNN with 5 convolutional layers, there is only one path for the gradient of the first convolutional layer. However, if we add Flat-Skips, there are 5 paths from which the final gradient of the first layer results.

3.2 Backpropagation with Flat-Skips

Since Flat-Skips add additional connections to the network architecture and therefore creates more paths for the gradient to flow, the backpropagation has to be looked at in more detail. There are two approaches on how to backpropagate the gradient when Flat-Skips are involved. The first of which is based on the perception, that Flat-Skips transform a L layered CNN is a collection of 1 to L layered CNNs and for each of these CNNs backpropagation is performed individually. This means, that L backpropagation cycles have to be executed in order to respect the gradient of each Flat-Skip connection. First, backpropagation is performed on the normal L layered CNN as if there were no Flat-Skips. Afterwards the feature-maps of the Flat-Skips which connects the $(L-1)$-th layer with the flatten layer were considered and backpropagation on a $(L-1)$-th layered CNN is performed. This continuous until the first Flat-Skip of the network. By doing so, the final gradient for each parameter is the result of L backpropagation processes. This way of using backpropagation is simple to implement, especially when not every

convolutional layer is connected with the flatten layer via Flat-Skips, however, it is not very computational efficient.
The second way to do backpropagation with Flat-Skips is to add the gradient of the Flat-Skips to the corresponding layer and perform backpropagation in a single run. This is faster then the approach described above, since only a single backpropagation run has to be made. However, the implementation can be more difficult.

3.3 Modified DenseNet

DenseNets have shown great results in computer vision tasks. The idea is to use a dense connectivity pattern within the convolutional part of the network, which creates a form of ”common knowledge”. In order to apply this pattern for very deep neural networks the authors of the original paper [26] use dense blocks. Within these dense blocks every convolutional layer is connected with every following convolutional layer via feature-map concatenation. Since the dimensions of the generated feature maps within such block are always the same, it is very simple to concat those feature maps. However, there is no feature-map concatenation between different blocks. In order to reduce the dimensionality of the feature-maps, transition layers were used between those dense blocks. These transition layers consist of a 1×1 convolutional and a pooling layer [26].
In this work the DenseNet architecture is adapted for smaller networks and is called modified DenseNets. Modified DenseNets do not use any transition layers, however, they use the dimension-reducing property of the convolutional operation. Therefore pooling is not as required as in the original DenseNet. In order to reduce the dimensions of the feature-maps no padding is used during a convolutional operation. This results in different dimensions between the input and the computed feature maps. To still be able to get a dense connectivity pattern within the convolutional part of the network an adaptive pooling method was developed, which can reduce the dimensions of any feature map to a desired number. In this work the just described adaptation of dense connectivity was applied to small sized convolutional neural networks, ranging from 3 to 7 convolutional layers. Furthermore, Flat-Skips, which were described in the section 3.1, were added to the network architecture. So every convolutional layer had an direct connection to the flatten layer, resulting in a maximum information flow within the network. That way the modified DenseNet with L convolutional layers consists of $\frac{L(L+1)}{2} - (L-1)$ from the dense connectivity pattern and additional $L-1$ connections from the Flat-Skips. In comparison, a standard convolutional layer with L layers just has L connections, one between every convolutional layer and just one connection to the flatten layer.

3.3.1 Adaptive pooling

In order to be able to concat feature-maps of different dimensions a simple adaptive pooling operation was developed. The idea is to reduce the dimension of bigger sized feature-maps to the desired dimension using a pooling operation. This can be done with a max or average-pooling operation. During the tests max-pooling performed slightly better. The adaptive pooling operation works as follows:
Let d_1 be the the desired dimension and d_2 the dimensions of the feature-maps that should be shrunk ($d_1 \leq d_2$). First, the required pooling window size is calculated by the rounded dividend between d_1 and d_2. Next it is required to calculate the number of times this pooling window is applied on the feature map in order to get the desired format. In order to clarify this, the procedure gets explained by an example. Let the input to an adaptive pooling operation have the dimensions 8×8. The goal is to get a output matrix with the dimensions 3×3. Therefore the window size ω is calculated as followed: $\omega = \lceil 8/3 \rceil = 3$. So the pooling window size that has to be used is 3×3. The idea is to use this window size as often as possible starting from the upper left corner of the matrix and then switch to a smaller window size when needed. In the case described above the window size 3×3 would be used 2 times, before switching to a pooling window size of 2. Then the same process would apply for the next row. Of course padding has to be considered when doing so. The output matrix of an adaptive max-pooling operation would look like this:

$$
\begin{array}{|ccc|ccc|cc|}
\hline
5 & 3 & 2 & 4 & 0 & 2 & 1 & 6 \\
0 & 2 & 5 & 0 & 1 & 1 & 0 & 0 \\
2 & 0 & 1 & 1 & 2 & 0 & 1 & 1 \\
\hline
2 & 3 & 2 & 3 & 3 & 2 & 2 & 3 \\
3 & 0 & 2 & 6 & 0 & 1 & 3 & 5 \\
2 & 1 & 0 & 0 & 0 & 3 & 3 & 2 \\
\hline
0 & 1 & 0 & 1 & 2 & 2 & 1 & 3 \\
1 & 3 & 0 & 4 & 4 & 2 & 0 & 1 \\
\hline
\end{array}
\rightarrow
\begin{array}{|ccc|}
\hline
5 & 4 & 6 \\
3 & 6 & 5 \\
3 & 4 & 3 \\
\hline
\end{array}
$$

Figure 3.3: 8×8 input matrix for adaptive pooling

The advantage of adaptive pooling is, that any output dimension can be achieved. It is still important to note, that in some cases the information in the top right corner gets compressed more then the information in the lower left corner. Backpropagation works similarly when using a normal pooling operation, however, it is necessary to consider the changing pooling window size. Since the output of an adaptive pooling operation is defined by the dimensions of the input feature maps and the desired output dimensions, the result is unambiguous. Therefore assigning the gradient to the correct values during backpropagation is unambiguous too.
The Java code for an adaptive pooling algorithm would look like this:

Algorithm 1 Adaptive max-pooling

Function: *adaptive_max_pooling(matrix, n)*

Input:

matrix . . . input matrix with dimensions $m \times m$

n . . . desired output dimension

Output:

result . . . output matrix with the desired dimensions $n \times n$

```
1:
2: int m = matrix.length;
3: double step_size = (double) m / (double) n;
4: int transition_counter = m - (int) step_size * n;
5: int sec_step_size = (int) step_size;
6: if (step_size - (int) step_size > 0) step_size = (int) step_size + 1;
7: double[][] result = new double[n][n];
8: int step_row = (int) step_size;
9: int row = 0;
10: for(int i=0; i<m; i=i+step_row) {
11:     if (row == transition_counter) {
12:         step_row = sec_step_size;
13:     }
14:     int step_column = step_size;
15:     int column = 0;
16:    for(int j=0; j<m; j=j+step_column){
17:         if (column == transition_counter) {
18:             step_column = sec_step_size;
19:         }
20:         double max = Double.NEGATIVE_INFINITY;
21:         for(int l=0; l<step_row; l++) {
22:             for(int k=0; k<step_column; k++) {
23:                 max = Math.max(max, matrix[i+l][j+k]);
24:             }
25:         }
26:         result[row][column] = max;
27:         column = column + 1;
28:     }
29:     row = row + 1;
30: }
31: return result;
```

3.4 LoopyDenseNet

LoopyDenseNets (LDN) are the result of combining many different concepts, which have shown to be very capable on its own. The base of a LoopyDenseNet is a standard feedforward convolutional neural network. The unique characteristic of the LoopyDenseNet lies in its implementation of the dense connectivity pattern and the implementation of the loops within the convolutional part of the network. Important to node here is, that all the convolutional operations which are processed in the LDN do not use any form of padding, because the dimension reducing property of convolutions should be utilized. Because of that the feature-map size decreases by each layer. Nevertheless, similarly to the original DenseNet a dense connectivity pattern via feature-map concatenation should be achieved [26]. To accomplish this, the feature-map sizes have to be the same. In the modified DenseNet architecture, described in the section 3.3, this was done via an adaptive pooling operation. LoopyDenseNets in contrast achieve this by looping convolutional operations multiple times until the desired feature-map size is reached. Then the feature-maps can be concatenated. Compared to the loops which were described in the paper "Loopy Neural Nets: Imitating Feedback Loops in the Human Brain" [29] and was covered in section 2.2, where several convolutional layers are looped (in fact the whole convolutional part), LDNs only loop a single layer a specific amount of times. This means, that the output of a layer directly gets fed to the same layer again and the same convolutional operation of this layer gets applied, resulting in the same number of feature-maps as with the original output of that layer. However, the size of the feature-maps gets smaller. That way these feature-maps can be concatenated with feature-maps of different layers. Considering a CNN with L convolutional layers, the feature-maps, of the second layer, can be concatenated to the feature maps that have gone through the first convolutional layer twice. For this to work, the filters of all convolutional layers must have the same size. Similarly the feature-maps of the third convolutional layer have the same size as the feature-maps that were looped three times through the first convolutional layer and the feature-maps that come from the second convolutional layer, which were looped through the second layer again. In order to clarify this the forward propagation of a LoopyDenseNet is looked at more closely in the following section.

3.4.1 Forward propagation in LoopyDenseNet

As described in the introduction of the LoopyDenseNet architecture, the functionality of the LDN is to perform convolution operations in a loop to reduce the size of the feature-maps so that the feature-maps of different layers can be concatenated. That way a dense connectivity pattern should be achieved which is similar to the one of a DenseNet [26]. In this section the interaction between loops and additional connections between layers will be examined.

During feedforward propagation, a distinction is made between normal layers and looped layers. The forward propagation of a non-looped layer (normal convolutional layer) is the same as in a traditional CNN and can be described with the following equation:

$$h_l = \sigma(x_l w_l + b_l) \tag{3.1}$$

In this equation h_l is the output of the l-th convolutional layer with the input x_l. The l-th convolutional layer has the weight w_l and the bias b_l. The activation function is represented by σ. The forward propagation of a looped layer looks differently and can be described as follows:

$$h_l^t = \sigma(h_l^{t-1} w_l + b_l) \tag{3.2}$$

In this equation h_l^{t-1} stands for the output of the $(t-1)$-th loop of the l-th layer. So the state h_l^t is dependent on the previous state h_l^{t-1} of the same hidden unit. In comparison to a Recurrent Neural Network the looped convolution of a LDN does not receive a second input.

The forward propagation will be shown in more detail in the following section. For this let I_l be the input feature-maps of the l-th layer and O_l the output of the l-th layer. The activation function at the l-th layer is represented by σ_l. That means, that in order to get the output of the i-th layer, the input has to go through the convolutional operations of this layer: $O_l = \sigma_l(I_l)$. Considering a input with the size $n \times n$, the output of this layer would have a format of $n - m + 1 \times n - m + 1$ when using a $m \times m$ convolution. For the following example a 5 layered LoopyDenseNet is looked at. The input to the first convolutional layer has the size 28×28 and only 3×3 convolutions were used through out the network. After the first convolutional layer the feature-maps have a size of 26×26. The input to the second convolutional layer is the output of the first: $I_2 = \sigma_1(I_1) = O_1$. This is of course a simplification since usually normalization layers or other types of layers are used in between. However, the dimensions of the feature-maps do not change. The input to the third convolutional layer consist of all the feature-maps which have the size 24×24. In the case of the example these feature-maps are the output of the second layer plus the feature-maps which result from looping the output of the first layer: $I_3 = \sigma_2(I_2) \oplus \sigma_1(\sigma_1(I_1)) = O_2 \oplus \sigma_1(O_1)$. The $\oplus$ is used to indicate the feature-map concatenation. Continuing with the input for the fourth layer, which consists of the output of the third layer plus the results of the three times looped first convolution layer plus the looped output from the second layer: $I_4 = \sigma_3(I_3) \oplus \sigma_2(\sigma_2(I_2)) \oplus \sigma_1(\sigma_1(\sigma_1(I_1))) = O_3 \oplus \sigma_2(O_2) \oplus \sigma_1(\sigma_1(O_1))$. In order to simplify the following terms an expression like $\sigma_l(\sigma_l(\sigma_l(I_l)))$ will be written as $\sigma_l(I_l)^3$. For completion the input to the fifth layer looks like this: $I_5 = \sigma_4(I_4) \oplus \sigma_3(I_3)^2 \oplus \sigma_2(I_2)^3 \oplus \sigma_1(I_1)^4$. In general, the following applies to the input of the l-th layer:

$$I_l = \oplus_{j=1}^{l-1} \sigma_j(I_j)^{l-j} \quad with \quad l \; > \; 1 \tag{3.3}$$

In this equation the $\oplus_{j=1}^{l-1}$ stands for the feature map concatenation and should be read like a $\sum$ sign. The table below shows the input of the l-th layer in relation to the input of the network I_1.

Layer	Input of the l-th layer in relation to $\mathbf{I_1}$
1	I_1
2	$I_2 = \sigma_1(I_1)$
3	$I_3 = \sigma_2(\sigma_1(I_1)) \oplus \sigma_1(I_1)^2$
4	$I_4 = \sigma_3(\sigma_2(\sigma_1(I_1)) \oplus \sigma_1(I_1)^2) \oplus \sigma_2(\sigma_1(I_1))^2 \oplus \sigma_1(I_1)^3$
5	$I_5 = \sigma_4(\sigma_3(\sigma_2(\sigma_1(I_1)) \oplus \sigma_1(I_1)^2) \oplus \sigma_2(\sigma_1(I_1))^2 \oplus \sigma_1(I_1)^3) \oplus \sigma_3(\sigma_2(\sigma_1(I_1)))^2 \oplus \sigma_2(\sigma_1(I_1))^3 \oplus \sigma_1(I_1)^4$
6	$I_6 = \sigma_5(\sigma_4(\sigma_3(\sigma_2(\sigma_1(I_1)) \oplus \sigma_1(I_1)^2) \oplus \sigma_2(\sigma_1(I_1))^2 \oplus \sigma_1(I_1)^3) \oplus \sigma_3(\sigma_2(\sigma_1(I_1))^2 \oplus \sigma_2(\sigma_1(I_1))^3 \oplus \sigma_1(I_1)^4)) \oplus \sigma_4(\sigma_3(\sigma_2(\sigma_1(I_1)) \oplus \sigma_1(I_1)^2) \oplus \sigma_2(\sigma_1(I_1))^2 \oplus \sigma_1(I_1)^3)^2 \oplus \sigma_3(\sigma_2(\sigma_1(I_1))^3 \oplus \sigma_2(\sigma_1(I_1))^4 \oplus \sigma_1(I_1)^5$
7	$I_7 = \ldots$

Table 3.1: Input of the l-th layer in relation to the input I_1 of a LoopyDenseNet

As can be seen in the table 3.1 the input of a layer gets more complex as the network gets deeper. In comparison, a table of the input of an ordinary CNN follows:

Layer	Input of the l-th layer in relation to $\mathbf{I_1}$
1	I_1
2	$I_2 = \sigma_1(I_1)$
3	$I_3 = \sigma_2(\sigma_1(I_1))$
4	$I_4 = \sigma_3(\sigma_2(\sigma_1(I_1))$
5	$I_5 = \sigma_4(\sigma_3(\sigma_2(\sigma_1(I_1))$
6	$I_6 = \sigma_5(\sigma_4(\sigma_3(\sigma_2(\sigma_1(I_1))$
7	$I_7 = \ldots$

Table 3.2: Input of the l-th layer in relation to the input I_1 of a CNN

When taking a closer look, the feature-maps that were generated in the CNN are included in the LoopyDenseNet. The feature-map depth of the LDN grows faster then the feature-map depth of the CNN. This is especially true, when network consist of many convolutional layers. The feature-map depth of the input to a layer results from the sum of filters used in order to generate the feature-maps. Let F_l be the number of filters of the l-th layer. So the feature-map depth of the l-th layer gets calculated like this: $\sum_{j=1}^{j=l-1} F_l$. Since the feature-map depth of the input of a convolutional layer grows, also the depth of the filters has to increase, resulting in more computational effort. In order to be able

to use the LoopyDenseNet architecture for very deep neural networks, in section 3.4.5 and 3.4.4 some methods were described in order to deal with the increasing depth of the feature-maps.
The next question that arises is how to incorporate pooling layers into a LDN. As mentioned in the introduction of the LoopyDenseNet, the base of this network architecture is the traditional convolutional neural network. Since CNNs often times also include pooling layers, the LoopyDenseNet also has to be able to apply them. Although the LoopyDenseNet utilizes the dimension reducing properties of convolutions, pooling still might be required. Pooling layers have the ability to greatly reduce the dimensionality of the feature-maps and therefore reduce the computational costs. A 2×2 pooling operations reduces the size of the feature-maps to 25%, which greatly effects the computation time. Max-pooling and average-pooling are the most common pooling operations. With max-pooling the strongest activation in each pooling window gets captured, while average-pooling looks at the average activation in the pooling window [48] [76] [30]. Since pooling operations effect the dimensions of the feature-maps they have to be considered when looping the convolutions so that the resulting feature-maps can be concatenated. Basically the feature-maps that were looped go through the same convolution and pooling schedule as the feature-maps of the corresponding CNN. To explain this, the following neural network, which consists of 2 3×3 convolutional layers, a 2×2 pooling layer followed by 2 more convolutional layers, another 2×2 pooling layer and a fifth convolutional layer, will be looked at. The first convolutional layer has 16 filters, the second convolutional layer 32 filters, the third layer 64, the fourth 96 filters and the fifth layer 128 filters. For this example a input with the format $28 \times 28 \times 3$ gets used. In the following table the progression of the feature-map dimensions were given.

	Main	**Conv 1 (Loops)**	**Conv 2 (Loops)**	**Conv 3 (Loops)**
Input	$28 \times 28 \times 3$	-	-	-
Conv 1	$3 \times 3 \times 3 \times 16$	-	-	-
Output	$26 \times 26 \times 16$	-	-	-
Input	$26 \times 26 \times 16$	$26 \times 26 \times 16$	-	-
Conv 2	$3 \times 3 \times 16 \times 32$	$3 \times 3 \times 3 \times 16$	-	-
Output	$24 \times 24 \times 32$	$24 \times 24 \times 16$	-	-
Pooling	$12 \times 12 \times 32$	$12 \times 12 \times 16$	-	-
Input	$12 \times 12 \times 48$	$12 \times 12 \times 16$	$12 \times 12 \times 32$	-
Conv 3	$3 \times 3 \times 48 \times 64$	$3 \times 3 \times 3 \times 16$	$3 \times 3 \times 16 \times 32$	-
Output	$10 \times 10 \times 64$	$10 \times 10 \times 16$	$10 \times 10 \times 32$	-
Input	$10 \times 10 \times 112$	$10 \times 10 \times 16$	$10 \times 10 \times 32$	$10 \times 10 \times 64$
Conv 4	$3 \times 3 \times 112 \times 96$	$3 \times 3 \times 3 \times 16$	$3 \times 3 \times 16 \times 32$	$3 \times 3 \times 48 \times 64$
Output	$8 \times 8 \times 96$	$8 \times 8 \times 16$	$8 \times 8 \times 32$	$8 \times 8 \times 64$
Pooling	$4 \times 4 \times 96$	$4 \times 4 \times 16$	$4 \times 4 \times 32$	$4 \times 4 \times 64$
Input	$4 \times 4 \times 208$	-	-	-
Conv 5	$3 \times 3 \times 208 \times 128$	-	-	-
Output	$2 \times 2 \times 128$			

Table 3.3: Feature-map generation of a 5 layered LoopyDenseNet.

As can be seen in the table 3.3 above, the feature-maps which were used as the input for the third layer have to have the dimensions 12×12. When looping the first convolutional layer twice the resulting feature-maps would just have the size 24×24. In order to reduce it to 12×12 a pooling operation has to be used. Similarly it works for the feature-maps which were used as the input for the fifth convolutional layer. After looping the first convolutional layer three times, a pooling operation has to be applied before the feature-maps can be concatenated with the feature-maps resulting from the fourth layer.
In this work two versions of LoopyDenseNets will be tested. On the one hand the LDN which was already described and at the other hand the LDN with Flat-Skips. When using Flat-Skips in the LoopyDenseNet architecture the flatten layer is more complex then the flatten layer of a traditional CNN. As discussed in the section 3.1 Flat-Skips connect each convolutional layer directly with the flatten layer. That way the flatten layer consists of the sum of the feature-maps which were generated during the convolutional part of the network making the whole information of the network available in the flatten layer. Because of that the flatten layer can get very large. Therefore it might be sufficient to use additional pooling layers before adding the flattened feature-maps to the flatten layer. This is especially true for the feature-maps which were generated in early layers, because their dimensions might be bigger.
To summarize the forward propagation of the LoopyDenseNet a figure is given, which illustrates how looping convolutions creates a dense connectivity pattern. In particular the looping process and the effects of Flat-Skips were shown. In the figure every arrow pointing to new feature-maps represents a convolutional operation and the arrows at the bottom illustrate the Flat-Skips, which transfer the feature-maps of the convolutional

layers to the flatten layer. Therefore the flatten layer is extended. Furthermore, the color of the arrow, indicates which convolutional operation of which layer gets applied. When multiple arrows point on the same convolutional operation, they get concatenated before. With each layer more loops were executed and as a result the depth of the concatenated feature-maps increases.

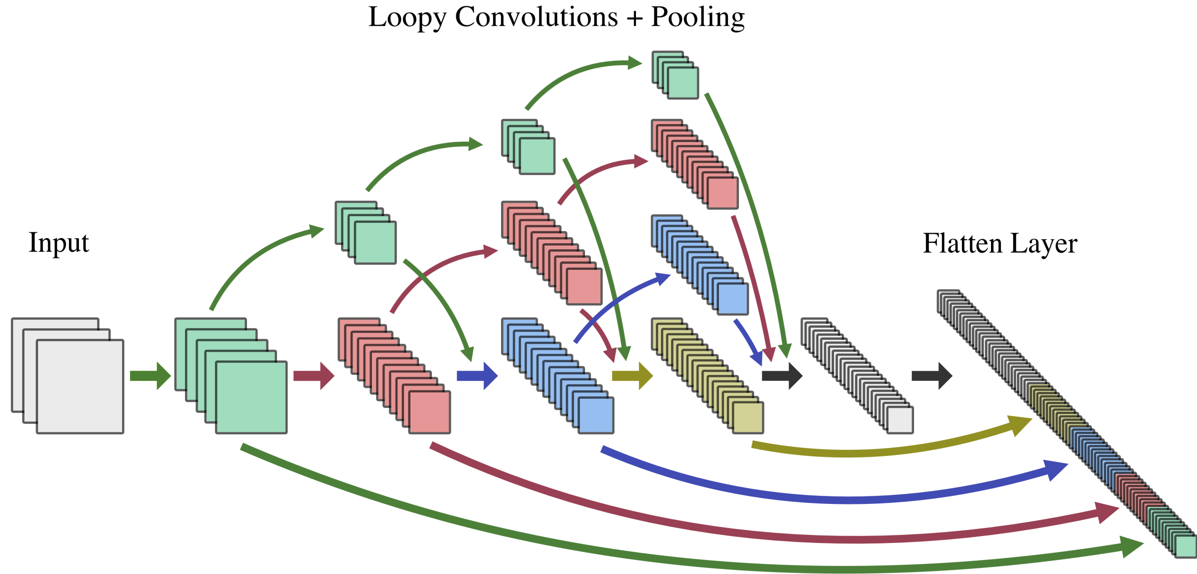


Figure 3.4: Convolutional part of a LoopyDenseNet with Flat-Skips

3.4.2 Backpropagation of a LoopyDenseNet

Since the forward propagation of the network includes loops the traditional backpropagation algorithm can not be applied for the whole network without making some adaptations. As described in the section 3.4.1, the convolutional operations of individual layers are used repeatedly. Since just one layer gets looped the output of one layer is also the input to the same exact layer. Consequently there have to be multiple updates to the weights of one layer, since they are used multiple times during the forward propagation. During backpropagation a distinction is made between a "normal" convolutional layer and a looped layer. A "normal" convolutional layer is every layer, which is used for the first time. This means, that during the standard backpropagation each layer receives an update coming from the standard backpropagation. However, every time a layer is looped, Backpropagation Through Time (BPTT) is used. In the case of a LoopyDenseNet this would look as follows. In the following consideration only a single looped layer is looked at. The output of the l-th layer at loop t, h_l^t, looks as follows:

$$h_l^t = \sigma(h_l^{t-1} w_l + b_l) \tag{3.4}$$

To compute $\frac{\partial L}{\partial w_l}$ the following equation is used:

$$\frac{\partial L}{\partial w_l} = \frac{\partial L}{\partial h_l^t} \frac{\partial h_l^t}{\partial w_l} \tag{3.5}$$

The term $\frac{\partial h_l^t}{\partial w_l}$ can then be written as:

$$\frac{\partial h_l^t}{\partial w_l} = \sigma'(h_l^{t-1} w_l + b_l)(h_l^{t-1} + w_l \frac{\partial h_l^{t-1}}{\partial w_l}) \tag{3.6}$$

The term $\frac{\partial h_l^{t-1}}{\partial w_l}$ is very familiar and is basically the same as the term $\frac{\partial h_l^t}{\partial w_l}$, however, it is considering one less time step. The equation 3.6 expands recursively until it reaches h_l^1 which is not dependent of w_l.
Depending on the number of layers, a layer receives several updates, which are added up. This is done similarly as in the paper "Loopy Neural Nets: Imitating Feedback Loops in the Human Brain" [29]. Let dW_l be the gradient for the l-th layer, which is equivalent to $\frac{\partial L}{\partial w_l}$. dW_l consists of the sum of the gradients from all the loops of the l-th layer. Considering that the l-th layer gets looped T times, the following therm applies:

$$dW_l = \sum_{t=1}^{T} dW_l^t \tag{3.7}$$

From the equation above, it can be seen that earlier layers receive more contributions to the gradient than later layers, since they are looped more often. This might be beneficial when dealing with the vanishing gradient problem. Additionally there is the option to also use Flat-Skips in the LDN, which also have to be considered during backpropagation, which is described in the section 3.2. All in all, the gradient has many different paths to propagate back, which hopefully enhances learning.

3.4.3 Effects of loops in the LoopyDenseNet

In the previous section the LoopyDenseNet architecture was introduced. Furthermore, the forward and backward propagation were explained. In this section the possible benefits and effects of looping convolutions are discussed. The loops of the LoopyDenseNet bring information from the affected layer to later layers by concatenating the looped feature-maps with feature-maps of later layers. That way a dense connectivity pattern is created. While the neural network described in the paper "Loopy Neural Nets: Imitating Feedback Loops in the Human Brain" loops multiple convolutional layers during a loop, the LoopyDenseNet only loops a single convolutional layer. By doing so the authors hope, that lower-level layers make a more refined choice of their weights, since they know the weights of higher-level features [29]. With the LoopyDenseNet this is not the case, since it is just looping a single layer. The LoopyDenseNet tries to pick up on the idea of the

original DenseNet, which is to encourage feature reuse and strengthen feature propagation. DenseNets create a collection of the generated feature-maps and therefore has a "collective knowledge", which gets increased after each convolutional layer [26]. Since LoopyDenseNets use Flat-Skips which combine all the feature-maps, which were generated during the convolutional part, in the flatten layer and therefore creates a different form of "collective knowledge", the loops should take over a different role. The goal of the loops is to detect more complex features, which should improve the performance of the network. Furthermore, there is less redundancy in feature-maps inside the convolutional part of the network. Looping convolutions makes networks very parameter efficient. Even a small LoopyDenseNet is capable to generate many different feature-maps and therefore detect many different features.
In order to understand the effects of convolutional loops more clearly, tests were made on a grayscale image of the digit five, which is one example of the MNIST dataset. In these tests convolutional operations were looped by using specific filters. Additionally no padding gets used during the convolution just like in the LoopyDenseNet architecture. Because of that the dimensions of the input and the resulting feature-maps differ. The convolutional operation gets applied seven times and the result of each loop gets visualized. For illustration, the results of the following filters are shown:

$$F_1 = \begin{bmatrix} 2 & 2 & 2 \\ 0 & -2 & 2 \\ 0 & 0 & 2 \end{bmatrix} \tag{3.8}$$

$$F_2 = \begin{bmatrix} -1 & 0 & 0 \\ 2 & -1 & 0 \\ 2 & 2 & -1 \end{bmatrix} \tag{3.9}$$

$$F_3 = \begin{bmatrix} 0 & 2 & -1 \\ 0 & 2 & -1 \\ 0 & 2 & -1 \end{bmatrix} \tag{3.10}$$

$$F_4 = \begin{bmatrix} 2 & -1 & 0 \\ 0 & 2 & -1 \\ -1 & 0 & 2 \end{bmatrix} \tag{3.11}$$

The first filter 3.8 detects upper right corners. With each loop the resulting feature-map seems to shift its focus to the upper right corner, which suggests that looping convolutions might have a shifting effect on the feature-maps. This means, that when looping certain filters the focus of the image might shift to a different region. Paired with the zooming effect, that results from not using any form of padding during convolutions, this can be especially helpful to find more detailed features in specific regions of the image. In the following representation the first image in the upper left corner represents the input image. Every following image stands for the output of another convolutional loop.

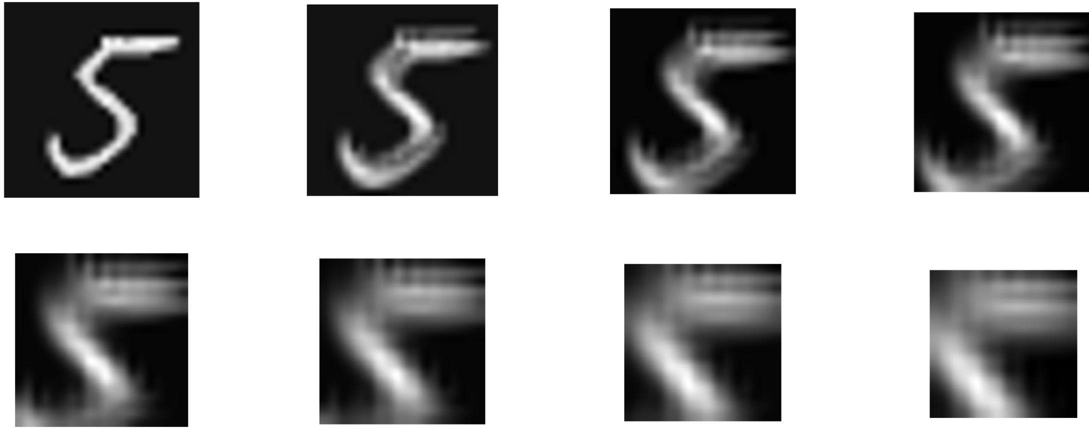


Figure 3.5: Feature-maps of the looped filter 3.8.

In order to show, that the shifting effect is not unique to one specific type of filter, filter 3.9 shows similar characteristics. In this case filter 3.9 detects lower left corners in the input and consequently the focus of the feature-maps shifts to the lower left corner. This effect seems to be even stronger then with the filter before.

Figure 3.6: Feature-maps of the looped filter 3.9.

Filter 3.10 detects horizontal edges. This can clearly be seen in the feature-maps of the first and second convolution. When looping this filter no significant shifting can be observed, however, a zooming effect is visible. This can especially be seen in the transition from the the fifth to the sixth convolutional loop. While the resulting feature-map of the fifth convolution include a horizontal bar at the upper left edge, the feature-map of the sixth layer does not include this bar anymore. Furthermore, since not only horizontal edges of the input image get detected, but also from the resulting feature-maps, much more complex features can be detected by only using one filter.

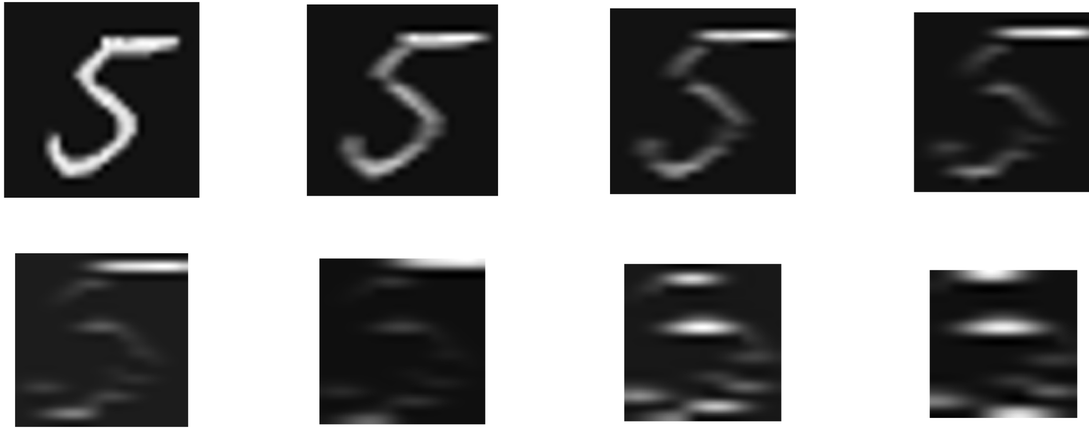


Figure 3.7: Feature-maps of the looped filter 3.10.

The last filter that will be examined, detects diagonal edges. Again the focus of the resulting feature-maps does not shift, only the zoom gets bigger with each convolutional loop.

Figure 3.8: Feature-maps of the looped filter 3.11.

Those visualization were just examples in order to show case the observation on a lot of testing on convolutional loops. The main findings on looping convolutional operations is that depending on the filter, the focus of the feature-maps might shift with each loop. This is especially interesting since paired with the zooming effect that is a result of the convolution, some areas of the image can be looked at in more detail. This shifting effect is not only true for those handcrafted filters mentioned above. Also randomly initialized filters tend to shift their focus to areas with high activations in the input feature-map. In order to show case this randomly initialized filters were looped on the same example as in the figures before. The values of the filters were randomly selected from an equal distribution between -1 and 1. In order to show, that the focus of the feature-maps is shifting to regions with higher activations, the input image gets shifted to the left before applying the convolutions.

Figure 3.9: Feature-maps of the randomly initialized filter applied on the image of the digit five which was shifted 6 pixels to the left.

Despite the image being shifted the computed feature-maps tend to focus on the region with high activations, which is certainly an interesting and advantageous behavior. While the object, in this case the digit five, is off center in the input image, with each convolution the activations are more centered. To show case, that this is not an incident the feature-map progression of a second randomly initialized filter is given.

Figure 3.10: Feature-maps of the randomly initialized filter applied on the image of the digit five which was shifted 8 pixels to the left.

All in all, looping convolutions can dynamically change the focus of the feature-maps to more interesting regions with higher activations. Furthermore, looping convolutions might result in the detection of more complex features, despite using the same parameters.

3.4.4 Looping limit

LoopyDenseNets get a dense connectivity pattern by concatenating the output of convolutional loops with later layers. By doing so the feature-map depth increases each layer and with every layer the computation time gets longer too. For shallow networks the increasing feature-map depth might not be a problem, however, in very deep networks with many layers the feature-map depth would grow extremely fast. The authors of the original DenseNet architecture had the same problem, which they resolved by using dense blocks. The dense connectivity pattern is used only in these blocks to keep the feature-map depth under control. Between those blocks transition layers were used which consist of a 1×1 convolution, which reduced the depth of the feature-map, followed by a pooling layer [26]. Theoretically LoopyDenseNets could also use dense blocks, however, the need for dense blocks in DenseNets does not come from the need to reduce the feature-map depth. Dense blocks were introduced in order to incorporate pooling layers, which otherwise would not be possible, since pooling layers would change the dimensions of the feature-maps and feature-map concatenation would not be possible [26]. Because of that LoopyDenseNets do not relay on dense blocks since using pooling layers do not face a problem for the LoopyDenseNet architecture.
In order to control the feature-map depth of deep LoopyDenseNets a new parameter τ is defined, which is called *looping limit*. This number indicates the maximum number of times a layer can be looped. To see the effects of introducing such a parameter, an example will be looked at. Considering a 8 layered LoopyDenseNet with the following filter numbers: 16 - 16 - 32 - 64 - 128 - 128 - 128 - 128. The input has the dimensions $28 \times 28 \times 3$. First a feature-map progression without a looping limit is shown.

Layer	Input	Filter	Output
Conv 1	$28 \times 28 \times 3$	$3 \times 3 \times 3 \times 16$	$26 \times 26 \times 16$
Conv 2	$26 \times 26 \times 16$	$3 \times 3 \times 16 \times 16$	$24 \times 24 \times 16$
Conv 1 (Loop)	$26 \times 26 \times 16$	$3 \times 3 \times 3 \times 16$	$24 \times 24 \times 16$
Conv 3	$24 \times 24 \times 32$	$3 \times 3 \times 32 \times 32$	$22 \times 22 \times 32$
Conv 1 (Loop)	$24 \times 24 \times 16$	$3 \times 3 \times 3 \times 16$	$22 \times 22 \times 16$
Conv 2 (Loop)	$24 \times 24 \times 16$	$3 \times 3 \times 16 \times 16$	$22 \times 22 \times 16$
Conv 4	$22 \times 22 \times 64$	$3 \times 3 \times 64 \times 64$	$20 \times 20 \times 64$
Conv 1 (Loop)	$22 \times 22 \times 16$	$3 \times 3 \times 3 \times 16$	$20 \times 20 \times 16$
Conv 2 (Loop)	$22 \times 22 \times 16$	$3 \times 3 \times 16 \times 16$	$20 \times 20 \times 16$
Conv 3 (Loop)	$22 \times 22 \times 32$	$3 \times 3 \times 32 \times 32$	$20 \times 20 \times 32$
Conv 5	$20 \times 20 \times 128$	$3 \times 3 \times 128 \times 128$	$18 \times 18 \times 128$
Conv 1 (Loop)	$20 \times 20 \times 16$	$3 \times 3 \times 3 \times 16$	$18 \times 18 \times 16$
Conv 2 (Loop)	$20 \times 20 \times 16$	$3 \times 3 \times 16 \times 16$	$18 \times 18 \times 16$
Conv 3 (Loop)	$20 \times 20 \times 32$	$3 \times 3 \times 32 \times 32$	$18 \times 18 \times 32$
Conv 4 (Loop)	$20 \times 20 \times 64$	$3 \times 3 \times 64 \times 64$	$18 \times 18 \times 64$
Conv 6	$18 \times 18 \times 256$	$3 \times 3 \times 256 \times 128$	$16 \times 16 \times 128$
Conv 1 (Loop)	$18 \times 18 \times 16$	$3 \times 3 \times 3 \times 16$	$16 \times 16 \times 16$
Conv 2 (Loop)	$18 \times 18 \times 16$	$3 \times 3 \times 16 \times 16$	$16 \times 16 \times 16$
Conv 3 (Loop)	$18 \times 18 \times 32$	$3 \times 3 \times 32 \times 32$	$16 \times 16 \times 32$
Conv 4 (Loop)	$18 \times 18 \times 64$	$3 \times 3 \times 64 \times 64$	$16 \times 16 \times 64$
Conv 5 (Loop)	$18 \times 18 \times 128$	$3 \times 3 \times 128 \times 128$	$16 \times 16 \times 128$
Conv 7	$16 \times 16 \times 384$	$3 \times 3 \times 384 \times 128$	$14 \times 14 \times 128$
Conv 1 (Loop)	$16 \times 16 \times 16$	$3 \times 3 \times 3 \times 16$	$14 \times 14 \times 16$
Conv 2 (Loop)	$16 \times 16 \times 16$	$3 \times 3 \times 16 \times 16$	$14 \times 14 \times 16$
Conv 3 (Loop)	$16 \times 16 \times 32$	$3 \times 3 \times 32 \times 32$	$14 \times 14 \times 32$
Conv 4 (Loop)	$16 \times 16 \times 64$	$3 \times 3 \times 64 \times 64$	$14 \times 14 \times 64$
Conv 5 (Loop)	$16 \times 16 \times 128$	$3 \times 3 \times 128 \times 128$	$14 \times 14 \times 128$
Conv 6 (Loop)	$16 \times 16 \times 128$	$3 \times 3 \times 256 \times 128$	$14 \times 14 \times 128$
Conv 8	$14 \times 14 \times 512$	$3 \times 3 \times 512 \times 128$	$12 \times 12 \times 128$

Table 3.4: Feature-map progression without looping limit.

As can be seen the feature-map depth at the end of the 8 layered LoopyDenseNet is 512. Furthermore, the average feature-map depth per convolutional layer is 176. Now the same network will be shown, however, this time τ with 2 will be used. That means, a convolutional layer can be looped 2 times.

Layer	Input	Filter	Output
Conv 1	$28 \times 28 \times 3$	$3 \times 3 \times 3 \times 16$	$26 \times 26 \times 16$
Conv 2	$26 \times 26 \times 16$	$3 \times 3 \times 16 \times 16$	$24 \times 24 \times 16$
Conv 1 (Loop)	$26 \times 26 \times 16$	$3 \times 3 \times 3 \times 16$	$24 \times 24 \times 16$
Conv 3	$24 \times 24 \times 32$	$3 \times 3 \times 32 \times 32$	$22 \times 22 \times 32$
Conv 1 (Loop)	$24 \times 24 \times 16$	$3 \times 3 \times 3 \times 16$	$22 \times 22 \times 16$
Conv 2 (Loop)	$24 \times 24 \times 16$	$3 \times 3 \times 16 \times 16$	$22 \times 22 \times 16$
Conv 4	$22 \times 22 \times 64$	$3 \times 3 \times 64 \times 64$	$20 \times 20 \times 64$
Conv 2 (Loop)	$22 \times 22 \times 16$	$3 \times 3 \times 16 \times 16$	$20 \times 20 \times 16$
Conv 3 (Loop)	$22 \times 22 \times 32$	$3 \times 3 \times 32 \times 32$	$20 \times 20 \times 32$
Conv 5	$20 \times 20 \times 112$	$3 \times 3 \times 112 \times 128$	$18 \times 18 \times 128$
Conv 3 (Loop)	$20 \times 20 \times 32$	$3 \times 3 \times 32 \times 32$	$18 \times 18 \times 32$
Conv 4 (Loop)	$20 \times 20 \times 64$	$3 \times 3 \times 64 \times 64$	$18 \times 18 \times 64$
Conv 6	$18 \times 18 \times 224$	$3 \times 3 \times 224 \times 128$	$16 \times 16 \times 128$
Conv 4 (Loop)	$18 \times 18 \times 64$	$3 \times 3 \times 64 \times 64$	$16 \times 16 \times 64$
Conv 5 (Loop)	$18 \times 18 \times 128$	$3 \times 3 \times 112 \times 128$	$16 \times 16 \times 128$
Conv 7	$16 \times 16 \times 320$	$3 \times 3 \times 320 \times 128$	$14 \times 14 \times 128$
Conv 5 (Loop)	$16 \times 16 \times 128$	$3 \times 3 \times 112 \times 128$	$14 \times 14 \times 128$
Conv 6 (Loop)	$16 \times 16 \times 128$	$3 \times 3 \times 224 \times 128$	$14 \times 14 \times 128$
Conv 8	$14 \times 14 \times 384$	$3 \times 3 \times 384 \times 128$	$12 \times 12 \times 128$

Table 3.5: Feature-map progression with looping limit of 2.

With τ of 2 the feature-map depth of the input to the eight layer of the LoopyDenseNet is just 384, which is 25% smaller then without a looping limit. The average feature-map depth is 146 per layer. The effect of a looping limit gets bigger with increasing network size.

As can be seen in both tables above there is the scenario where the filter depth is bigger than the depth of the looped feature-maps of the respective layer. This can be seen in the looped feature-maps which were generated by the sixth convolutional layer in the table 3.4 and the table 3.5. In the table with no looping limit the feature-maps have the dimensions $16 \times 16 \times 128$. When looping the sixth convolutional layer the filters have the dimensions $3 \times 3 \times 256 \times 128$. The filters are much deeper then the feature-maps. In this case a normal convolutional operation is applied, however, only half of the depth of the filter ($3 \times 3 \times 128$) gets used. This results in the desired output of $14 \times 14 \times 128$.

3.4.5 Bottleneck layer

Although using a looping limit can reduce the number of the feature-maps as the network gets deeper, it does not offer a lot of control over the feature-map depth. The second method to reduce the size and the computation time of the network would be to use

bottleneck layers in the form of a 1×1 convolution followed by a non-linearity before every convolutional layer. Since the feature-map depth of the input to a convolutional layer continuously increases with each layer, a 1×1 convolution presents a straightforward way to reduce the depth of the input feature-maps to a desired number. This can reduce computation without changing the network architecture. So before doing the normal convolution of a layer, a 1×1 convolution is processed. This way to reduce computation is also used in the original DenseNet, in the Inception module and many other network architectures [26] [70] [21] [69] [40] [10]. Another advantage of applying 1×1 convolutions is, that more non-linearities are added to the network, which might increase the expressive power of such network [69] [10]. In order to have more control over the depth of the feature-maps, bottleneck layers and a looping limit can be combined.

4. Datasets

In the following section all the datasets that were used for the experiments are described. Furthermore, in order to generate more images for training and to avoid overfitting data augmentation was used which gets adapted to each dataset so that the model is not overfitting on the evaluation data [23] [54]. The datasets were chosen, because they have relatively small dimensions and therefore can also be used when having limited computational resources.

4.1 MNIST

The original MNIST dataset consists of 60.000 training examples and 10.000 test examples. Each example has a format of 28 by 28 pixels and represents a grayscale image of a digit from 0 to 9. That means that each of the 784 pixels can have a value between 0 and 255. It is possible to download the MNIST dataset on [12] in a CSV format, as well as on [37] [39]. The MNIST dataset is a highly competitive dataset. The state of the art performance on MNIST at the date of publication of this work is 99.91% when using an ensemble of networks and 99.87% by a single network. According to [5] the highest reported accuracies on MNIST were the following:

Model and source	Accuracy	Parameters
Homogeneous ensemble with Simple CNN [7]	99.91%	-
Branching/Merging CNN + Homogeneous Filter Capsules [9]	99.87%	1,514,187
EnsNet [24]	99.84%	-
Efficient-CapsNet [43]	99.84%	161,824

4.1.1 Data Augmentation and preparation on MNIST

In order to evaluate the models during the experiments the training dataset was split up. That way the training dataset contains 50.000 images and the evaluation dataset 10.000 images. During training an example gets augmented with a probability of 67%. In the case of an augmentation the following operations were applied to modify the original instance and to get a new image:

- **Zoom:** With a gaussian distribution with mean 0 and variance 1 which gets scaled by a factor of 0.06 the image gets zoomed in or out. That way the size of the

digit inside the image can be changed. Because of the scaling of the image it can be expected that with a probability of about 63.5% the final example is up to 6% zoomed in or out, compared to the original image.

- **Shear:** Shearing gets applied with a gaussian distribution with mean 0 and variance 1 and a scaling factor of 0.03.
- **Rotation:** Next a rotation operation is used. This is also gaussian distributed with mean 0 and variance 1 and a scaling of 4. That way the image can be rotated about 4 degrees clockwise respectively counterclockwise.
- **Shift:** In the end the image gets shifted along the x and the y coordinates with a gaussian distribution with mean 0 and variance 1 and a scaling factor of 0.06.

By applying those operations a new image of a digit will be derived from the original image. After the augmentation the instance is made mean free, by subtracting the average pixel intensity from each pixel. Now the second augmentation step comes to place, which consists of Random Erasing [77]. In comparison to the first data augmentation step, Random Erasing is applied on every training example. Compared to the operations described above, Random Erasing does not change the position of pixels, however, randomly selects a rectangle area in the image and erases its pixels. In the paper [77] the authors recommend to randomly initialize the pixel values of the selected region, however, since the background of the MNIST digit is black, the pixels intensity are simply set to 0. The erasing of pixels is similar to dropout, where neurons are randomly dropped during training. While dropout drops neurons inside the network, Random Erasing drops information already at the input layer [66]. The height and the width of the rectangle area is randomly set via a uniform distribution that can be between 0 and 30% of the height or the width of the image. Since the images of the MNIST dataset have a format of 28 by 28, the maximum size of this region is 8 by 8. That way up to 64 pixels can be erased. This process of randomly selecting a rectangle and erasing its values can be repeated up to 5 times. The amount of data augmentation used and the scaling of the individual operations were adapted so that the base CNN model for the MNIST dataset, which is discussed in section 5.1, slightly underfits on the evaluation data. In general the following procedure was followed when adjusting the individual scaling factors of the augmentation operations. This does not only apply for this dataset, however, for all following datasets which use data augmentation.

1. Initialize the augmentation rate to 50% and the scaling factors of the zooming, shearing, shifting operations to 0.2 and the scaling factor of the rotation operation to 2.0.
2. Train the base model (further description in section 5.1) for 10 epochs on the training data and evaluate on the evaluation data. Use a learning rate decay of 40% to account for the small amount of epochs.

3. Check if the error and the loss on the training data is smaller or equal to the error and the loss on the evaluation data.

a.) If the loss and the error is smaller on the training data, increase the scaling factors of the operations and the augmentation rate. In general the scaling of the shifting and the zooming operations were increased more frequently, because too much shearing and rotation seem to hurt performance. Afterwards go back to step 2 and repeat.

b.) If the error and the loss on the the training data is higher (more then 5%) then on the evaluation data, decrease the scaling of operations. The scaling of the shearing and the rotation operation is more likely to be reduced. The augmentation rate should stay the same. Go back to step 2 and repeat.

c.) If the error and the loss on the training data is slightly bigger or equal to the error and the loss on the evaluation data stop and save the values.

4.2 Fashion-MNIST

Fashion-MNIST, which is a dataset from Zalando, is similar to the original MNIST dataset and consists of 60.000 training examples and 10.000 test examples. Similar to MNIST the images in Fashion-MNIST are grayscale images with a format of 28 by 28 pixels [63]. The dataset contains images of 10 different articles of clothing and it is possible to download the dataset on [58] as a CSV file and on [63]. Identical to the MNIST dataset, the training dataset of Fashion-MNIST gets split up as well, to have 50.000 images for training and the remaining 10.000 images for evaluation [75]. During training of the models the same data augmentation procedure were applied as for the MNIST dataset (4.1.1). The only difference is, that 50% of the images get augmented, since the base CNN model does not overfit as easily on the Fashion-MNIST dataset as compared to the MNIST dataset.
The state of the art performance according to [4] on Fashion-MNIST is 96.91%:

Model and source	**Accuracy**	**Parameters**
Fine-Tuning DARTS [73]	96.91%	3.2 M
Shake-Shake [17]	96.41%	-
Random Erasing [77]	96.36%	-
VGG8B (2x) [46]	95.86%	28 M

As can be seen in the state of the art performance on the Fashion-MNIST dataset, this dataset is much more challenging then the MNIST dataset. In comparison to MNIST, Fashion-MNIST represents clothing, which contains much more complex structures then digits. Furthermore, some clothing types have similar structures and because of the small resolution are difficult to distinguish.

4.3 Fruits-360

The Fruits-360 dataset contains over 90.000 images of 131 fruits and vegetables. 67.692 images are available for training and additional 22.688 for testing. Each image has a resolution of 100 by 100 and is a colored image. In the dataset different varieties of the same fruits were distinguished as separate classes. Therefore the dataset contains 13 different apple varieties and 18 different pear and tomato varieties. The dataset was created by filming each fruit or vegetable while rotating it. The fruit or vegetable was then extracted from the background. From these videos several 100 by 100 colored images were generated. The dataset is available at [49] and [50].
For the Fruits-360 dataset no data augmentation was used, since the CNN model was not overfitting on the evaluation data.
In the following table the best results based on [1] and own research were listed:

Model and source	**Accuracy**	**Parameters**
Wide ResNet-101 2 (Spinal FC) [31]	100.0%	125.5 M
VGG-19 bn (Spinal FC) [31]	99.96%	198.75 M

From each class about 80 images were used for evaluation. The rest was used for training. Since the computational resources in this work are limited, using the original 100 by 100 image was not sufficient. In order to reduce computational costs the format of the input image was shrunk to 32 by 32 pixels. First a margin of 2 pixels was cropped of the image, which in most cases was just white background, resulting in a 96×96 image. Next a 3×3 average-pooling operation was applied on the cropped image to get the final image of 32×32.

4.4 Hand gesture

On Kaggle you can download a popular hand gesture dataset [41]. It consists of 20.000 images of 10 different hand gestures. Those hand gestures were performed by 5 men and 5 women, where for each person and gesture 200 images were taken. The examples are grayscale images with a format of 240 by 640 pixels [20] [41].
The same data augmentation operations were applied on 75% of the hand gesture images during training as on the MNIST datasets, only the scaling factors differ: Zoom with 0.06, shear with 0.05, rotation with 6.0 and shifting with 0.06. Those values were the results from the exact same parameter search as described in section 4.1.1. With the hand gesture dataset overfitting was a much harder problem then with the other datasets described above, resulting in a higher augmentation rate and in general higher scaling factors. Especially the rotation and shearing operation were helpful to reduce the validation loss. Additionally Random Erasing is used in the same way as in the MNIST dataset.

4.4.1 Preprocessing of the images

While the original images of the hand gesture have a format of 240 by 640, the actual gesture covers a small area in the image. In most cases the hand gesture does not exceed an area of 240 by 320 pixels. Because of that the image gets cropped to this format, so that the hand gesture is roughly centered. To reduce computational costs this centering process is not very accurate, however, this is not very important considering the data augmentation and the following steps. In order to give it a square shape, it gets a black frame on the top and on the bottom to make it 320 to 320. After this the image gets scaled down to 32 by 32 pixels using average-pooling. That way it comes more in line with the other datasets used. Important to note is, that besides reducing the computational costs, reducing the resolution of the input image makes learning more difficult [72].

4.4.2 Splitting the data

On Kaggle there are a lot of submission where they merge all the available data and make random data splits with the merged data. However, since the images are generated as a video, there are not big differences between successive images. Therefore merging and shuffling the images makes for an easy classification problem. When the model is capable to learn the training dataset, it should not be difficult to get a good performance on the evaluation and the test dataset, since the images are so similar. That is probably the reason why there are a lot of submissions on Kaggle with up to 100% accuracy [20]. However, in the experiments performed for this work also the generalizability of the model should be important. To test this, cross-validation gets used. The testing procedure was done as follows. The data was split up in ten parts. Each part contains the images of one person (2.000 images). The images of the first person were used for evaluation. After defining the hyperparameters of the base model (a more detailed description will follow in section 5.2) cross-validation will be done on the data of the 9 remaining persons. The images from each of the nine persons were used once for testing, while the images of the remaining eight persons were used for training.

4.5 CIFAR-10

The CIFAR-10 dataset consists of 50.000 training images and 10.000 test images. Each example is a 32 by 32 colored image and belongs to one of 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck) [33]. From all the datasets used in this work the CIFAR-10 is the most complex dataset for a small network. Compared to all other datasets, the object to be classified in the CIFAR-10 dataset is not highlighted from the background. While in the MNIST, Fashion-MNIST, Fruits-360 and the hand gesture dataset the object and the background can be clearly distinguished from each other, this

is not true for instances of the CIFAR-10 dataset. Furthermore, the objects to be classified consist of more complex structures than a number or a single garment.
For the experiments the training dataset gets split up. In the end the model gets trained with 40.000 images and evaluated with 10.000 images. During training the same data augmentation operations were used as for the MNIST dataset (4.1.1). Additionally the image can be flipped vertically with a probability of 50%. The results of the parameter search for the CIFAR-10 dataset follows. In general the scaling factors of the augmentation operations were slightly smaller compared to the dataset before. The zoom operation gets scaled with a factor of 0.04, shearing with 0.02, rotation with 3.0 and shifting with 0.05. The scaling factors are smaller since the CIFAR-10 dataset is a far more complex dataset with more sophisticated shapes and structures. Because of the low resolution of 32×32 even small changes have a large effect on the augmented images. When those changes are to big classification might be to difficult. In order to compensate for the small scaling factors for the augmentation operations, the augmentation rate was increased to 80%. In the second data augmentation step Random Erasing is used as described in 4.1.1.
CIFAR-10 is highly competitive dataset for image classification tasks and according to [3] the state of the art performance at the date of publication of this work is $99.50 \pm 0.06\%$ and was achieved by a ViT-H/14 [14].

5. Experiment

In the following experiment the performance of the 5 different models, which are the ordinary CNN, the CNN with Flat-Skips, the modified DenseNet, the LoopyDenseNet and the LoopyDenseNet with Flat-Skips, are evaluated on the MNIST, the Fashion-MNIST, the Fruits-360, the hand gesture and the CIFAR-10 datasets. Since the examples of the datasets represent an image of a object the models have to solve a classification task. Those datasets were chosen because the images have small dimensions and therefore training is not very computational expensive. Furthermore, MNIST, Fashion-MNIST and CIFAR-10 are popular benchmarking datasets. The hand gesture and Fruits-360 are chosen in order to compare the generalizing abilities of the networks. By doing so the research question, whether additional connections, loops and a dense connectivity pattern increase the performance of the model, should be answered.

5.1 Base model

In the following section the different network settings that apply for all tests are described. Since the goal of this work is to show whether or not including loops, a dense connectivity pattern and additional connections within a convolutional neural network improve the accuracy of a neural network, a base model for each dataset was designed, which was used for all the following tests. This was done in order to not have to search for the optimal set of parameters for all these 5 different architectures. The base model was designed in order to maximize the performance of a 4 layered CNN with just one fully-connected layer on the MNIST dataset. This was also the only time when searching for hyperparameters. All following models share the exact same parameters in terms of learning rate, learning rate decay, batch size, optimizer, normalization, activation function, loss function and pooling operation. The only difference lies in their architecture, their number of layers and the number of filters per layer. Furthermore, the amount and the type of data augmentation, as well as the implementation of early stopping is adjusted for each dataset according to the base model. However, these settings apply for all models of the same dataset. No parameter search was done for other models or other architectures. Important to note is that the model size is small. This is because of computational restrictions.
The following parameters resulted from the parameter search of the 4-layered CNN on MNIST. First of which the mini-batch size was set to 256, which seems to make the performance of the network very consistent, since the gradient over the mini-batch is very close to the gradient of the whole training dataset. Usually a smaller mini-batch size is recommended, however, since data augmentation was heavily used the larger mini-batch

size works well [42] [57]. Furthermore, layer normalization gets used, which computes the mean and the variance for the input to a layer and makes the input feature-maps mean free [8]. It comes into use in the convolutional layers, as well as in the flatten layer. Layer normalization is very easy to implement and in comparison to the original batch normalization it works the same during training and testing. Similar to batch normalization layer normalization reduces the required training time and speeds up convergence [8] [28] [61].
For the optimizer of the network ADAM was chosen, with the standard settings $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-7}$. ADAM is used for stochastic gradient-based objective functions and computes individual learning rates for every parameter. It is doing this by estimating the first and second moments of the gradients [32]. Also tests with solely using momentum were made, however, when using ADAM the model converged much faster and higher accuracies could be achieved [55] [56]. ADAM was also tested with different parameter choices of β_1, β_2 and ϵ. For β_1 the following values were tested: $0.85, 0.9, 0.95, 0.975$, where 0.9 and 0.95 achieved similar results. In the end 0.9 was chosen, which was also suggested by the original paper [32]. Furthermore, β_2 was set to $0.99, 0.999$ and 0.9999, however, 0.999 was clearly better then all other settings. When selecting the value ϵ slightly better results could be achieved when setting ϵ to 10^{-7} instead of 10^{-8} as the author of the original paper suggest. Although combining ADAM and learning rate decay is uncommon, since the learning rates of the parameters get adapted individually [32], big performance improvements could be achieved when using a small amount of learning rate decay. The initial learning rate was set to 0.001 and gets decreased by 15% after each epoch. When selecting the decay rate the following properties could be observed. On the one hand when the decay rate is too small, the model tends to converge slowly and since early stopping is used training might stop very early because no new global optimum can be found in time. On the other hand when the decay rate is big and the learning rate gets reduced very quickly, the model converges very fast, however, later during training the parameter updates are very small due to the small learning rate and the model has difficulties escaping local minima. With a learning rate decay of 15% the overall best results could be achieved.
In order to prevent the CNN from overfitting data augmentation gets used. The magnitude and the type of data augmentation is dependent on the dataset used and was described in the section 4 in more detail. Besides data augmentation, early stopping was used, which stops training of the model after a specific amount of epochs without achieving a new global maximum in accuracy. The patience for the MNIST and the Fashion-MNIST dataset was set to 8 epochs and for the CIFAR-10 dataset to 10 epochs. Both these datasets seem to get stuck in local minima more frequently and therefore a bigger patience was required. For the Fruits-360 dataset and the hand gesture dataset the patience was set to 6 epochs. The base model had no problems to achieve 100% accuracy on the evaluation data of the Fruits-360 dataset and therefore a bigger patience

was not necessary. In the case of the hand gesture dataset the base model achieved its peak performance on the evaluation data in most cases within the first 10 epochs and therefore a high patience was not needed. Of course in general enlarging the number of epochs the model searches for a new global maximum in performance would be beneficial, however, this was not possible due to computational restrictions.

As mentioned before, the model only possesses one fully-connected layer, which connects the flatten layer with the output layer of the network. This was done in order to keep the number of parameters low, since fully-connected layers are often times parameter heavy. This is especially true when the flatten layer is large. Another reason for only using one fully-connected layer is to minimize the effects of the fully-connected part of the network on the performance of the model, as the goal of this work is to investigate if adding additional connections, applying a dense connectivity pattern and using loops within the convolutional part of the network increases the performance of a CNN. Additionally, only 3×3 convolutions were used, since they show superior results compared to larger filter sizes, like 5×5 during evaluation. Using 3×3 convolutions is also more parameter efficient then bigger filters. Two 3×3 filters cover the same receptive field as one 5×5 convolution, however, have less parameters. For example a single 5×5 filter consists of 5^2 parameters, while two 3×3 filters have $2 * 3^2$ parameters. Considering a feature-map of 5×5, using a 5×5 or two 3×3 convolutions have the same affect regarding the resulting feature-map format. When applying the first 3×3 convolution, the 5×5 feature-map becomes a 3×3 feature-map. Applying the second 3×3 convolution results in a 1×1 feature-map, which is also the result when applying a 5×5 convolution on a 5×5 feature-map. Furthermore, more non-linear rectification layers can be added this way [65]. When using convolutions no form of padding was applied, which means, that the output feature-maps have a different dimension than the input. ReLU was chosen to be the activation function, since it achieved far better accuracies compared to sigmoid or softsign [22] [53] [30] [59] [51] [45] [18] [19]. For the classification function softmax was chosen and cross-entropy is the loss function that gets used [47] [64]. Furthermore, for the pooling operation max-pooling showed better results than average-pooling.

The base model (4-layered CNN) on the MNIST dataset had the following architecture and consists of about 140k parameters and needs about 16.4M FLOPs for a single feed-forward pass:

Layer	**Input**	**Dimensions**	**Output**
Conv 1	$28 \times 28 \times 1$	$3 \times 3 \times 1 \times 16$	$26 \times 26 \times 16$
Conv 2	$26 \times 26 \times 16$	$3 \times 3 \times 16 \times 32$	$24 \times 24 \times 32$
Max-Pooling	$24 \times 24 \times 32$	2×2	$12 \times 12 \times 32$
Conv 3	$12 \times 12 \times 32$	$3 \times 3 \times 32 \times 64$	$10 \times 10 \times 64$
Conv 4	$10 \times 10 \times 64$	$3 \times 3 \times 64 \times 96$	$8 \times 8 \times 96$
Flatten	6144×1	-	-
Fully-connected	6144	6144×10	10
Output	-	-	10×1

Table 5.1: Base model: 4-layered CNN

5.2 Model setup and model selection

In section 5.1 a base model was designed, which maximizes the accuracy on the MNIST dataset. The parameters which were defined during the design phase of the base model apply for all following models and architectures. No further parameter search was done. Since the models should be tested on 5 different datasets, for each dataset a base CNN model with 4 convolutional layers was defined. For datasets where the input data has the same dimensions the exact same model gets used. This is the case for the MNIST and Fashion-MNIST, which both have a format of 28×28. The model can be seen in section 5.1 and consist of two convolutional layers with 16 and 32 filters, followed by a 2×2 max-pooling layer, followed by two more convolutions with 64 and 96 filters. The images of the CIFAR-10 and the processed images of the hand gesture dataset, as well as the Fruits-360 images, have the same format of 32×32. The base model for these datasets consists of two convolutional layers with 16 and 40 filters, followed by a max-pooling layer and two more convolutional layers with 64 and 96 layers. The number of filters per layer were chosen so that the computation time of the module does not get to long. This was necessary since the whole implementation of the architectures was done in Java without the use of public libraries and was not programmed for the GPU, but for the CPU, which is significantly slower. The models were tested on an Apple M1 chip [27] and the computation for an epoch of the base model on the MNIST dataset takes about 45 minutes. Because of these computational restrictions only small models could be tested.
All other models independent of the architecture were derived from these 4 layered CNNs. In order to guarantee, that the proposed methods do not have an unfair advantage over the base CNN models, the following aspect was considered. LoopyDenseNets, modified DenseNets and CNNs with Flat-Skips are more computational expensive, than a CNN, when the same number of filters and layers were used. To make valid comparisons between different architectures the training time for an epoch should be about equal for all models.

To accomplish a similar computation time for each model per epoch independent of their number of layers and their architecture, there is one parameter which can be adjusted. This is the number of filters per layer. All other parameters stay the same. In order to clarify this, the corresponding 7 layered LoopyDenseNet to the 4 layered CNN would have the following architecture:

Layer	**Input**	**Dimensions**	**Output**
Conv 1	$28 \times 28 \times 1$	$3 \times 3 \times 1 \times 6$	$26 \times 26 \times 6$
Conv 2	$26 \times 26 \times 6$	$3 \times 3 \times 6 \times 10$	$24 \times 24 \times 10$
Max-Pooling	$24 \times 24 \times 16$	2×2	$12 \times 12 \times 16$
Conv 3	$12 \times 12 \times 16$	$3 \times 3 \times 16 \times 16$	$10 \times 10 \times 16$
Conv 4	$10 \times 10 \times 32$	$3 \times 3 \times 32 \times 32$	$8 \times 8 \times 32$
Conv 5	$8 \times 8 \times 64$	$3 \times 3 \times 64 \times 64$	$6 \times 6 \times 64$
Conv 6	$6 \times 6 \times 128$	$3 \times 3 \times 128 \times 64$	$4 \times 4 \times 64$
Conv 7	$4 \times 4 \times 192$	$3 \times 3 \times 192 \times 96$	$2 \times 2 \times 96$
Flatten	384×1	-	-
Fully-connected	384	384×10	10
Output	-	-	10×1

Table 5.2: 7 layered LoopyDenseNet for the MNIST and Fashion-MNIST dataset.

Although the 7 layered LoopyDenseNet has three additional convolutional layers, it just needs about 11.7M FLOPs and consists of 158k parameters, compared to the base model, which needs 16.4M FLOPs and has 140k parameters. More detailed information of all models tested can be seen in the tables which come at the end of this section (table 5.4, table 5.5, table 5.6, table 5.7). Since the LoopyDenseNet is more computational expensive, the 7 layered CNN with Flat-Skips looks as follows:

Layer	**Input**	**Dimensions**	**Output**
Conv 1	$28 \times 28 \times 1$	$3 \times 3 \times 1 \times 10$	$26 \times 26 \times 10$
Conv 2	$26 \times 26 \times 10$	$3 \times 3 \times 10 \times 20$	$24 \times 24 \times 20$
Max-Pooling	$24 \times 24 \times 20$	2×2	$12 \times 12 \times 20$
Conv 3	$12 \times 12 \times 20$	$3 \times 3 \times 20 \times 30$	$10 \times 10 \times 30$
Conv 4	$10 \times 10 \times 30$	$3 \times 3 \times 30 \times 40$	$8 \times 8 \times 40$
Conv 5	$8 \times 8 \times 40$	$3 \times 3 \times 40 \times 40$	$6 \times 6 \times 40$
Conv 6	$6 \times 6 \times 40$	$3 \times 3 \times 40 \times 60$	$4 \times 4 \times 60$
Conv 7	$4 \times 4 \times 60$	$3 \times 3 \times 60 \times 100$	$2 \times 2 \times 100$
Flatten	10450×1	-	-
Fully-connected	10450	10450×10	10
Output	-	-	10×1

Table 5.3: 7 layered CNN with Flat-Skips

The 7 layered Flat-Skips consist of 213k parameters and requires 7.0M FLOPs and is therefore bigger then the 4 layered CNN, however, needs much fewer FLOPs per epoch. The flatten layer which consists of 10.450 neurons is the result of the concatenated feature-maps which come from the Flat-Skips. For the feature-maps which come from the first and second layer a 3×3 max-pooling operation was applied before passing them to the flatten layer. The computation time on the CPU of the M1 chip are about the same for both models, however, this is probably because of a bad implementation of the Flat-Skips and with better programming the computation of the 7 layered CNN with Flat-Skips can be significantly reduced, which is indicated by the smaller number of FLOPs. However, to get the same computation time on the CPU of the M1 chip the number of filters per layer is reduced.
The following table gives an overview over all the models which were used on the MNIST and the Fashion-MNIST dataset. The input images have a format of 28×28. All models use one 2×2 max-pooling layer after the second convolutional layer. The second column in the table displays the number of filters used per layer. Furthermore, in the third column the size of the flatten layer is shown. The following two columns show the number of parameters, as well as the FLOPs needed for a single feedforward pass of the respective model. In order to find the appropriate depth for each network architecture, each of these models were evaluated on the evaluation dataset. In the table the models which achieved the highest accuracy on the evaluation data of the MNIST and Fashion-MNIST dataset are highlighted. These models were then used to test them on the test set and make the final comparison between the network archietctures.

Model	Convolutional part	Flatten layer	Params.	FLOPs
CNN				
3-layered	24 - 46 - 64	6400	101k	17.1M
4-layered	16 - 32 - 64 - 96	6144	140k	16.4M
5-layered	12 - 24 - 40 - 64 - 128	4608	154k	13.2M
6-layered	12 - 24 - 32 - 64 - 64 - 128	2048	195k	11.9M
7-layered	12 - 24 - 32 - 32 - 48 - 64 - 128	512	140k	8.2M
CNN with Flat-Skips				
3-layered	22 - 44 - 60	10598	139k	15.3M
4-layered**	14 - 28 - 56 - 80	13646	195k	12.5M
5-layered	10 - 20 - 40 - 60 - 100	13530	220k	10.6M
6-layered*	10 - 20 - 30 - 50 - 60 - 100	12050	223k	8.9M
7-layered	10 - 20 - 30 - 40 - 40 - 60 - 100	10450	213k	7.0M
Modified DenseNet				
3-layered	20 - 40 - 64	10580	136k	13.4M
4-layered	14 - 26 - 48 - 80	12718	211k	15.8M
5-layered	10 - 16 - 26 - 54 - 96	11346	238k	13.1M
6-layered	8 - 12 - 24 - 48 - 64 - 96	10728	320k	12.7M
7-layered	6 - 10 - 16 - 32 - 64 - 64 - 96	8486	374k	8.8M
LDN				
3-layered**	20 - 40 - 64	6400	106k	15.8M
4-layered	14 - 28 - 54 - 72	4608	132k	17.2M
5-layered*	12 - 18 - 30 - 60 - 100	3600	187k	17.8M
6-layered	10 - 14 - 24 - 48 - 64 - 96	1536	156k	16.9M
7-layered	6 - 10 - 16 - 32 - 64 - 64 - 96	384	158k	11.7M
LDN with Flat-Skips				
3-layered**	20 - 40 - 64	10580	148k	15.9M
4-layered	12 - 24 - 48 - 72	11916	192k	14.0M
5-layered*	10 - 16 - 26 - 54 - 96	11346	238k	14.4M
6-layered	8 - 12 - 24 - 48 - 64 - 96	10728	320k	16.0M
7-layered	6 - 8 - 16 - 32 - 64 - 64 - 96	8358	367k	11.2M

* highest accuracy on the evaluation data of the MNIST dataset

** highest accuracy on the evaluation data of the Fashion-MNIST dataset

Table 5.4: Models for the MNIST and Fashion-MNIST dataset.

In general non-CNN models have slightly more parameters. This is because of the larger flatten layer which comes from the use of Flat-Skips. Nevertheless, non-CNN models tend to require fewer FLOPs for a feedforward pass. This comes from the reduction of the number of filters, which is needed to align the computation time on the hardware used.

For the hand gesture dataset and the CIFAR-10 dataset also 2×2 max-pooling was used after the second convolutional layer. The next two tables show the models tested for the hand gesture dataset which has an input format of $32 \times 32 \times 1$ and the CIFAR-10 dataset which has an input format of $32 \times 32 \times 3$. In general they are very similar, however, since the images in CIFAR-10 have 3 input channels the models are slightly more computational expensive.

Model	Convolutional part	Flatten layer	Params.	FLOPs
3-layered	24 - 48 - 96	13824	191k	29.6M
4-layered	16 - 40 - 64 - 96	9600	181k	27.7M
5-layered	12 - 32 - 48 - 64 - 128	8192	201k	25.1M
6-layered	12 - 24 - 32 - 64 - 96 - 128	4608	241k	25.5M
7-layered	12 - 24 - 32 - 48 - 64 - 96 - 128	2048	238k	20.5M
CNN with Flat-Skips				
3-layered	24 - 46 - 88	16462	212k	27.6M
4-layered	16 - 36 - 60 - 80	19428	263k	23.5M
5-layered	12 - 28 - 40 - 54 - 96	19444	274k	18.5M
6-layered	10 - 20 - 30 - 60 - 80 - 100	20660	346k	19.2M
7-layered	10 - 20 - 30 - 40 - 60 - 80 - 100	18260	338k	15.7M
Modified DenseNet				
3-layered	22 - 40 - 80	14888	202k	26.6M
4-layered	16 - 32 - 48 - 72	16704	255k	26.8M
5-layered	10 - 24 - 32 - 48 - 72	15832	273k	22.2M
6-layered	8 - 14 - 22 - 38 - 64 - 72	14854	353k	19.4M
7-layered	8 - 12 - 24 - 32 - 48 - 48 - 72	13708	198k	17.4M
LDN				
3-layered	20 - 40 - 80	11520	166k	25.2M
4-layered	18 - 36 - 54 - 72	7200	175k	33.2M
5-layered	12 - 24 - 36 - 48 - 72	4608	170k	28.1M
6-layered	8 - 12 - 20 - 40 - 54 - 72	2592	171k	20.7M
7-layered	3 - 6 - 9 - 18 - 36 - 72 - 72	1152	168k	10.9M
LDN with Flat-Skips				
3-layered	20 - 40 - 80	14760	199k	25.3M
4-layered	16 - 32 - 48 - 72	16704	255k	28.3M
5-layered	10 - 22 - 32 - 48 - 72	15734	269k	24.4M
6-layered	6 - 10 - 16 - 32 - 40 - 72	11530	218k	14.1M
7-layered	3 - 6 - 9 - 18 - 36 - 72 - 72	9630	252k	11.1M

Table 5.5: Models for the hand gesture dataset.

Model	Convolutional part	Flatten layer	Params.	FLOPs
CNN				
3-layered	24 - 48 - 96	13824	191k	29.6M
4-layered	16 - 40 - 64 - 96	9600	181k	27.7M
5-layered	12 - 32 - 48 - 64 - 128	8192	201k	25.1M
6-layered	12 - 24 - 32 - 64 - 96 - 128	4608	241k	25.5M
7-layered	12 - 24 - 32 - 48 - 64 - 96 - 128	2048	238k	20.5M
CNN with Flat-Skips				
3-layered	24 - 46 - 88	16462	212k	27.6M
4-layered	16 - 36 - 60 - 80	19428	263k	23.5M
5-layered	12 - 28 - 40 - 54 - 96	19444	274k	18.5M
6-layered	10 - 20 - 30 - 60 - 80 - 100	20660	346k	19.2M
7-layered	10 - 20 - 30 - 40 - 60 - 80 - 100	18260	338k	15.7M
Modified DenseNet				
3-layered	22 - 40 - 80	14888	202k	26.6M
4-layered	16 - 32 - 48 - 72	16704	255k	26.8M
5-layered	10 - 24 - 32 - 48 - 72	15832	273k	22.2M
6-layered	8 - 14 - 22 - 38 - 64 - 72	14854	353k	19.4M
7-layered	8 - 12 - 24 - 32 - 48 - 48 - 72	13708	198k	17.4M
LDN				
3-layered	20 - 40 - 80	11520	166k	25.8M
4-layered	18 - 36 - 54 - 72	7200	175k	33.8M
5-layered	12 - 24 - 36 - 48 - 72	4608	170k	28.5M
6-layered	8 - 12 - 20 - 40 - 54 - 72	2592	171k	21.0M
7-layered	3 - 6 - 9 - 18 - 36 - 72 - 72	1152	168k	11.1M
LDN with Flat-Skips				
3-layered	20 - 40 - 80	14760	199k	25.8M
4-layered	16 - 32 - 48 - 72	16704	255k	28.9M
5-layered	10 - 22 - 32 - 48 - 72	15734	269k	24.7M
6-layered	6 - 10 - 16 - 32 - 40 - 72	11530	218k	14.3M
7-layered	3 - 6 - 9 - 18 - 36 - 72 - 72	9630	252k	11.2M

Table 5.6: Models for the CIFAR-10 dataset.

With the Fruits-360 dataset the number of parameters increase significantly compared to the models used for the other datasets. This is mainly because of the 131 classes. Because of that the fully-connected layer which connects the flatten layer and the output layer is more parameter heavy and computational expensive compared to the other datasets, which only have 10 classes. Since most non-CNN models use Flat-Skips, which usually increases the size of the flatten layer, those models also have significantly more parameters, however, the FLOPs required are usually lower.

Model	**Convolutional part**	**Flatten layer**	**Params.**	**FLOPs**
CNN				
3-layered	24 - 48 - 96	13824	1863k	33.0M
4-layered	16 - 40 - 64 - 96	9600	1342k	30.0M
5-layered	12 - 32 - 48 - 64 - 128	8192	1193k	27.1M
6-layered	12 - 24 - 32 - 64 - 96 - 128	4608	798k	26.6M
7-layered	12 - 24 - 32 - 48 - 64 - 96 - 128	2048	486k	21.0M
CNN with Flat-Skips				
3-layered	24 - 46 - 88	16462	2204k	31.6M
4-layered	16 - 36 - 60 - 80	19428	2614k	28.2M
5-layered	12 - 28 - 40 - 54 - 96	19444	2627k	23.2M
6-layered	10 - 20 - 30 - 60 - 80 - 100	20660	2846k	24.2M
7-layered	10 - 20 - 30 - 40 - 60 - 80 - 100	18260	2548k	20.0M
Modified DenseNet				
3-layered	22 - 40 - 80	14888	2004k	30.2M
4-layered	16 - 32 - 48 - 72	16704	2277k	30.8M
5-layered	10 - 24 - 32 - 48 - 72	15832	2189k	26.0M
6-layered	8 - 14 - 22 - 38 - 64 - 72	14854	2109k	23.0M
7-layered	8 - 12 - 24 - 32 - 48 - 48 - 72	13708	2012k	20.7M
LDN				
3-layered	20 - 40 - 80	11520	1560k	28.0M
4-layered	18 - 36 - 54 - 72	7200	1046k	34.9M
5-layered	12 - 24 - 36 - 48 - 72	4608	727k	29.2M
6-layered	8 - 12 - 20 - 40 - 54 - 72	2592	485k	21.3M
7-layered	3 - 6 - 9 - 18 - 36 - 72 - 72	1152	307k	11.2M
LDN with Flat-Skips				
3-layered	20 - 40 - 80	14760	1985k	28.9M
4-layered	16 - 32 - 48 - 72	16704	2277k	32.4M
5-layered	10 - 22 - 32 - 48 - 72	15734	2173k	28.2M
6-layered	6 - 10 - 16 - 32 - 40 - 72	11530	1613k	16.9M
7-layered	3 - 6 - 9 - 18 - 36 - 72 - 72	9630	1418k	13.4M

Table 5.7: Models for the Fruits-360 dataset.

6. Results

In the following chapter the results of the experiments are discussed. Since all architectures were implemented in Java without the use of public libraries, the models are certainly not runtime optimal. This is especially true for the CNN with Flat-Skips, Modified DenseNet and LoopyDenseNet, as they require more data management than the normal CNN. To find the best model for the hardware at hand, the models are evaluated based on their accuracy according to the time required. Since the models were designed so that an epoch takes approximately the same amount of time, the different models can be compared based on their accuracy under the parameter settings described above. However, to account for runtime and non-optimal implementation of the networks, accuracy is also measured according to the FLOPs required. This means that only the accuracies of the non-CNN models achieved within the FLOPs required by the CNN to reach its maximum accuracy are considered. This can ensure that the non-CNN models do not get an advantage due to additional required computing power. In the following comparisons, the accuracy is given considering early-stopping, as well as after considering the required FLOPs. Since usually the non-CNN models do not need as many FLOPs for a single feedforward pass the accuracy is often the same for both observations.

6.1 Experiment on the MNIST dataset

The first dataset that will be looked at, is the MNIST dataset. The best accuracies on the evaluation data which consists of 10.000 images were achieved by the following models.

Model	Layers	Accuracy	Error Reduction	Parameters	FLOPs
CNN	4	99.53%	-	140k	16.4M
CNN (Flat-Skips)	6	99.33%	-42.55%	223k	8.9M
Modified DenseNet	4	99.39%	-29.79%	211k	15.8M
LDN	5	99.52%	-2.13%	187k	17.8M
LDN (Flat-Skips)	5	99.47%	-12.77%	238k	14.4M

Table 6.1: Performance of the best models of the respective architecture on the evaluation data of the MNIST dataset.

As can be seen the 4-layered CNN consisting of 140k parameters reached the overall best result with an accuracy of 99.53% on the evaluation dataset. The second best model was the LDN without Flat-Skips. All other models performed noticeably worse then those two models. Next the models were tested on the test data of the MNIST dataset. Almost all models achieved a higher accuracy on the test data compared to the evaluation data. The performance on the test data is shown in the following figures.

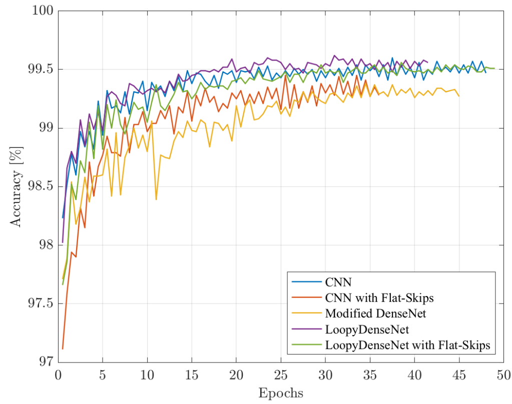


Figure 6.1: Accuracy on the test data of the MNIST dataset.

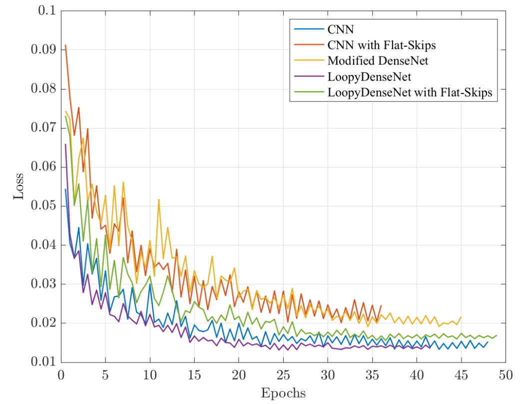


Figure 6.2: Loss on the test data of the MNIST dataset.

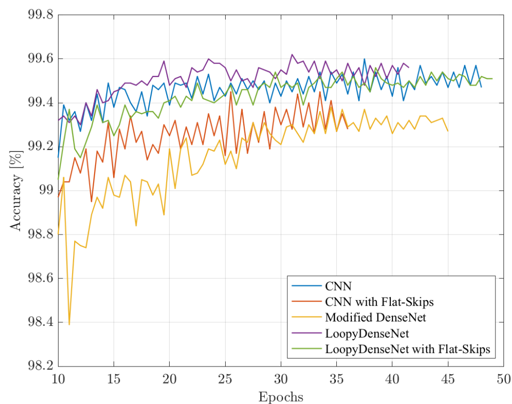


Figure 6.3: Accuracy on the test data of the MNIST dataset beginning from the 10-th epoch.

As can be seen the 5-layered LDN reaches the highest accuracy on the test data with an accuracy of 99.62%. The CNN reaches the second highest accuracy with 99.60%. Interestingly the CNN with Flat-Skips and the modified DenseNet perform significantly worse then the standard CNN, indicating that a more complex structure of the convolutional part does not always improve performance.

The following table summarizes the results on the test data. In addition to the accuracy, the error bounds are also given to assess the significance of the results. The error bounds were calculated with $\epsilon \pm (c_\delta \sqrt{\epsilon(1-\epsilon)/n} + c_\delta^2/n)$. In this expression ϵ stands for the error on the test data, while c_δ is set to 1.96 when having a confidence interval of 95%. Since the test dataset contains 10.000 examples n is 10.000. The error bounds will always be calculated with the formula above during this work.

Model	Layers	Accuracy	Error Reduction	Parameters	FLOPs
CNN	4	99.60 ± 0.162%	-	140k	16.4M
CNN (Flat-Skips)	6	99.45 ± 0.183%	-37.50%	223k	8.9M
Modified DenseNet	4	99.38 ± 0.192%	-55.00%	211k	15.8M
LDN	5	99.62 ± 0.160%	5.26%	187k	17.8M
LDN (Flat-Skips)	5	99.56 ± 0.168%	-10.00%	238k	14.4M

Table 6.2: Performance on the test data of the MNIST dataset.

As discussed before, comparing the achieved accuracies over time (over the epochs) does only make sense when looking for the best model on the given hardware. However, in

order to make a more general statement about the performance of the models, the achieved accuracy must be considered under the dependence of the required FLOPs. In order to take the FLOPs into account the number of FLOPs which the CNN model needs until it reaches its highest accuracy were counted. For all other models the accuracy will only be considered which was reached under the given number of FLOPs. For those models which stopped earlier because of early stopping the same accuracy as in the table above will be considered. However, accuracies which were achieved after the defined number of FLOPs will not be considered during the comparison.
The accuracy and the loss progression in respect to the FLOPs needed, looks as follows:

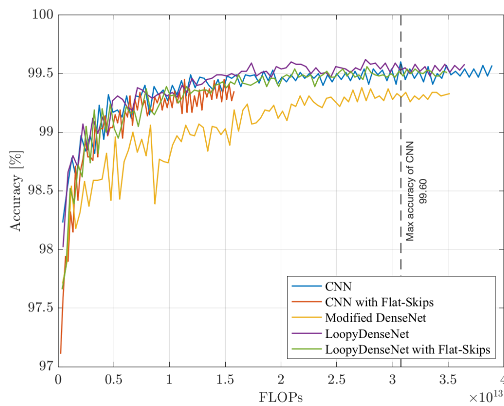


Figure 6.4: Accuracy on the test data of the MNIST dataset in respect to the FLOPs required.

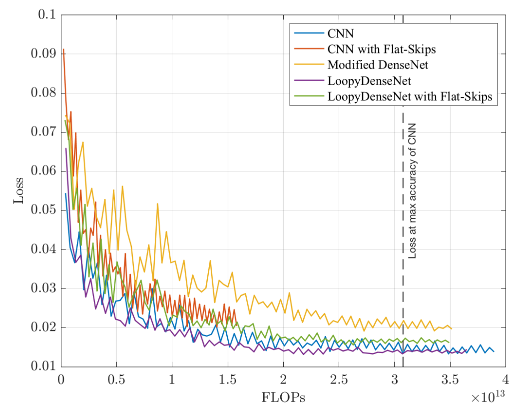


Figure 6.5: Loss on the test data of the MNIST dataset in respect to the FLOPs required.

As can be seen, the non-CNN models achieve their highest accuracies before the CNN model does. Due to this, the maximum achieved accuracy of the non-CNN models is not reduced.
All in all, the accuracy of the LDN is not significantly better then the accuracy of the CNN. For the MNIST dataset a more complex convolutional part does not always improve performance.

6.2 Experiment on the Fashion-MNIST dataset

The Fashion-MNIST dataset is very similar to the MNIST dataset, however, the objects that should be classified consist of more complex structures and shapes. Therefore a lower accuracy is to be expected. The results on the evaluation data are shown in the table below.

Model	Layers	Accuracy	Error Reduction	Parameters	FLOPs
CNN	4	92.64%	-	140k	16.4M
CNN (Flat-Skips)	4	92.02%	-8.42%	195k	12.5M
Modified DenseNet	4	91.68%	-13.04%	211k	15.8M
LDN	3	92.45%	-2.58%	106k	15.8M
LDN (Flat-Skips)	3	92.70%	0.82%	148k	15.9M

Table 6.3: Performance of the best models of the respective architecture on the evaluation data of the Fashion-MNIST dataset.

All in all, the LDN with Flat-Skip achieves about the highest accuracy on evaluation data. The best CNN was slightly better then the LDN. After finding the best model of the respective architecture, the models were then tested on the test dataset. The following figures show the accuracy and the loss on the test data.

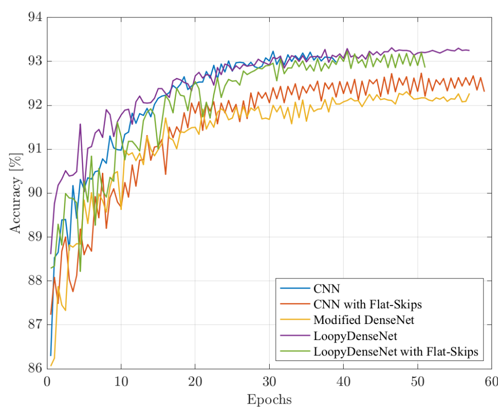


Figure 6.6: Accuracy on the test data of the Fashion-MNIST dataset.

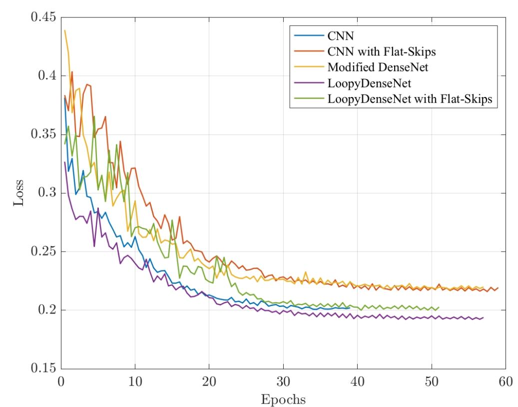


Figure 6.7: Loss on the test data of the Fashion-MNIST dataset.

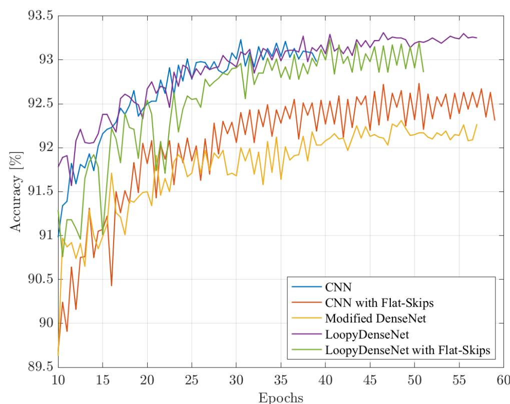


Figure 6.8: Accuracy on the test data of the Fashion-MNIST dataset beginning from the 10-th epoch.

All models were capable to achieve a higher accuracy on the test data compared to the evaluation data. The LDN without Flat-Skips manages to achieve the highest overall accuracy of 93.31%, which is, however, not a significant improvement over the standard CNN or the LDN with Flat-Skips, which both reached 93.23%.

Model	Layers	Accuracy	Error Reduction	Parameters	FLOPs
CNN	4	93.23 ± 0.531%	-	140k	16.4M
CNN (Flat-Skips)	4	92.73 ± 0.547%	-7.39%	195k	12.5M
Modified DenseNet	4	92.31 ± 0.561%	-13.59%	211k	15.8M
LDN	3	93.31 ± 0.528%	1.20%	106k	15.8M
LDN (Flat-Skips)	3	93.23 ± 0.531%	0.0%	148k	15.9M

Table 6.4: Performance on the test data of the Fashion-MNIST dataset.

In the following figures, the accuracy and the loss are shown in relation to the required FLOPs.

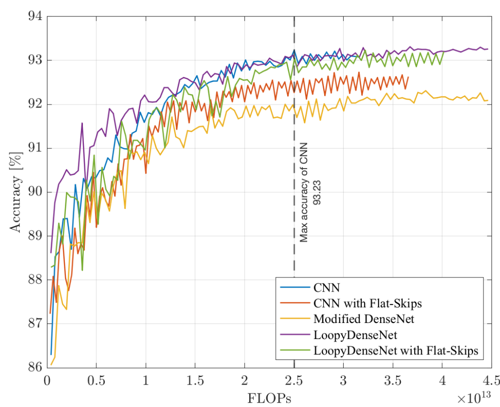


Figure 6.9: Accuracy on the test data of the Fashion-MNIST dataset in respect to the FLOPs required.

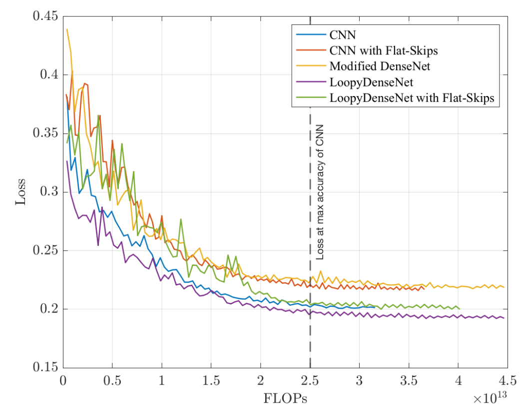


Figure 6.10: Loss on the test data of the Fashion-MNIST dataset in respect to the FLOPs required.

In the figure above it is clearly visible, that all non-CNN models need more FLOPs in order to achieve their maximum accuracy, then the CNN does. Because of that, the accuracies of the non-CNN models which accounts for the FLOPs required is smaller then from the table 6.5 above. When considering the FLOPs the CNN has the highest performance over all models, however, the difference to the LDN models is still not significant. The results were summarized in the following table.

Model	Layers	Accuracy	Error Reduction	Parameters	FLOPs
CNN	4	93.23 ± 0.531%	-	140k	16.4M
CNN (Flat-Skips)	4	92.59 ± 0.552%	-9.45%	195k	12.5M
Modified DenseNet	4	92.00 ± 0.570%	-18.17%	211k	15.8M
LDN	3	93.12 ± 0.535%	-1.62%	106k	15.8M
LDN (Flat-Skips)	3	93.01 ± 0.538%	-3.25%	148k	15.9M

Table 6.5: Performance on the test data of the Fashion-MNIST dataset considering the required FLOPs.

6.3 Experiment on the hand gesture dataset

In order to find the best models for each architecture the hand gesture dataset was split up in 10 parts. Each of the parts contains the images of one of the ten persons from which the images were taken (the dataset was described in more detail in section 4.4).

The images of the first person were used to evaluate the models. The following table shows the results of the best models of the respective architecture on the evaluation data.

Model	Layers	Accuracy	Error Reduction	Parameters	FLOPs
CNN	5	97.60%	-	201k	25.1M
CNN (Flat-Skips)	5	95.45%	-89.58%	274k	18.5M
Modified DenseNet	4	92.39%	-217.08%	255k	26.8M
LDN	5	97.60%	0.0%	170k	28.1M
LDN (Flat-Skips)	5	96.85%	-31.25%	269k	24.4M

Table 6.6: Performance of the best models of the respective architecture on the evaluation data of the hand gesture dataset.

The best CNN and the best LDN achieved the same accuracy on the evaluation data and both consist of 5 convolutional layers. The best CNN with Flat-Skips and the best modified DenseNet performed much worse then all other models on the evaluation data. Interestingly, they achieve slightly higher accuracies on the training data compared to the other models and therefore are heavily overfitting. This indicates, that they are very expressive, however, the models need more regularization. After finding the best model of each architecture, cross-validation was done, in order to compare the models properly. So every model was trained on the data of eight of the nine persons and evaluated on the data of the one remaining person. This was done nine times, so that the data of each person is used for evaluation once. The following figure shows the accuracies of each of the nine folds.

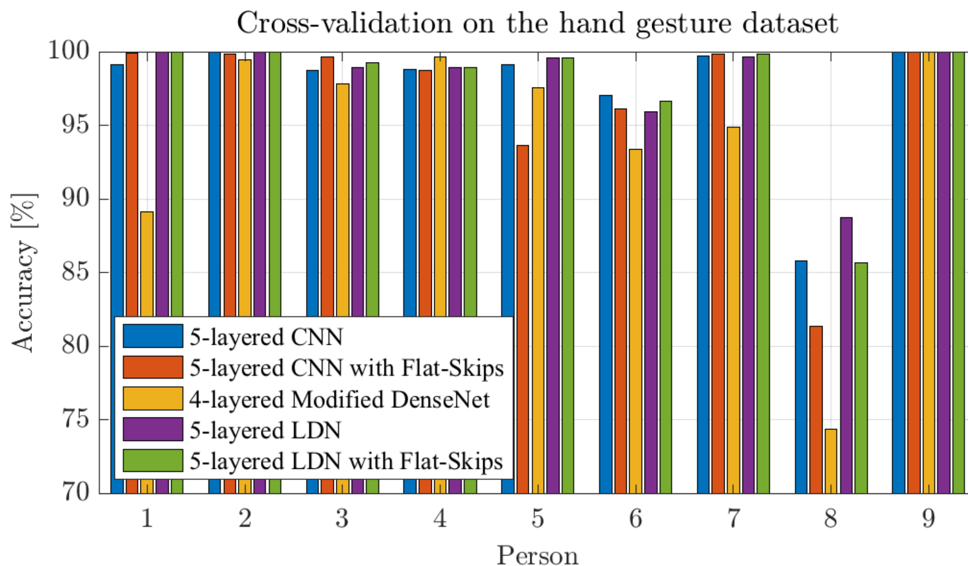


Figure 6.11: Results of the cross-validation on the hand gesture dataset.

In the figure above it is possible to see, that all models but the modified DenseNet tend to achieve similar results. Correctly classifying the images of person eight seems to be difficult for all of the models. The highest accuracy for this person was achieved by the LDN with 88.75%, which is significantly lower then the accuracies which were achieved on the images of the other persons. In the following table the average accuracy of all nine persons were given.

Model	Layers	Accuracy	Error Reduction	Parameters	FLOPs
CNN	5	97.59 ± 0.864%	-	201k	25.1M
CNN (Flat-Skips)	5	96.56 ± 0.991%	-42.74%	274k	18.5M
Modified DenseNet	4	94.03 ± 1.230%	-147.72%	255k	26.8M
LDN	5	97.97 ± 0.810%	18.72%	170k	28.1M
LDN (Flat-Skips)	5	97.77 ± 0.839%	8.07%	269k	24.4M

Table 6.7: Results of the cross-validation on the hand gesture dataset.

The standard 5-layered CNN managed to achieve a respectable result of 97.59%, which is higher then the accuracy of the CNN with Flat-Skips and much higher then the accuracy of the modified DenseNet. Both LDN models, the one with and the one without Flat-Skips achieved slightly higher accuracies then the CNN. The LDN manages to reduce the error of the CNN by about 18%. It is important to note, that because of the small size of the evaluation dataset, which only consist of 2.000 images per fold, the difference in accuracy is not significant.
Next the accuracies were looked at while considering the required FLOPs. Again, only the results of the non-CNN models obtained with a lower or equal number of FLOPs that the CNN model required to achieve its highest accuracy were considered. Because of that a slightly smaller accuracy was assumed in the first, second, fifth, sixth, seventh and eighth fold for the LDN and for the LDN with FLat-Skips in the first, second, fifth and eighth fold. The following figures should illustrate the accuracies in respect to the FLOPs needed for the sixth person.

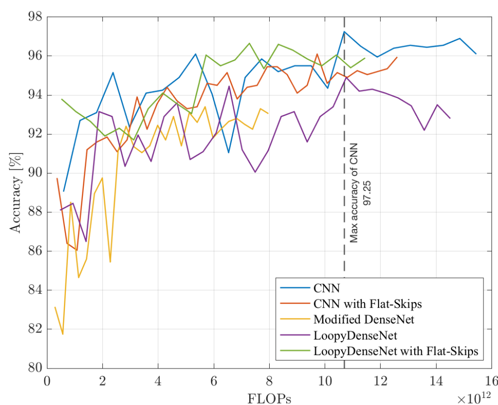


Figure 6.12: Accuracy on the test data of the hand gesture dataset in respect to the FLOPs required.

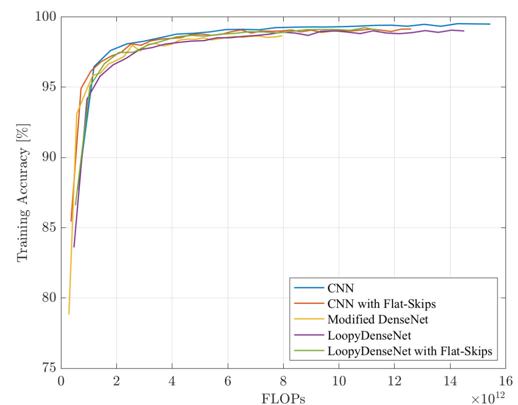


Figure 6.13: Accuracy on the training data of the hand gesture dataset in respect to the FLOPs required.

As can be seen, the LoopyDenseNet requires more FLOPs to achieve its highest accuracy then the CNN and therefore a smaller accuracy will be considered. The following figures shows the accuracies of the models after considering the FLOPs.

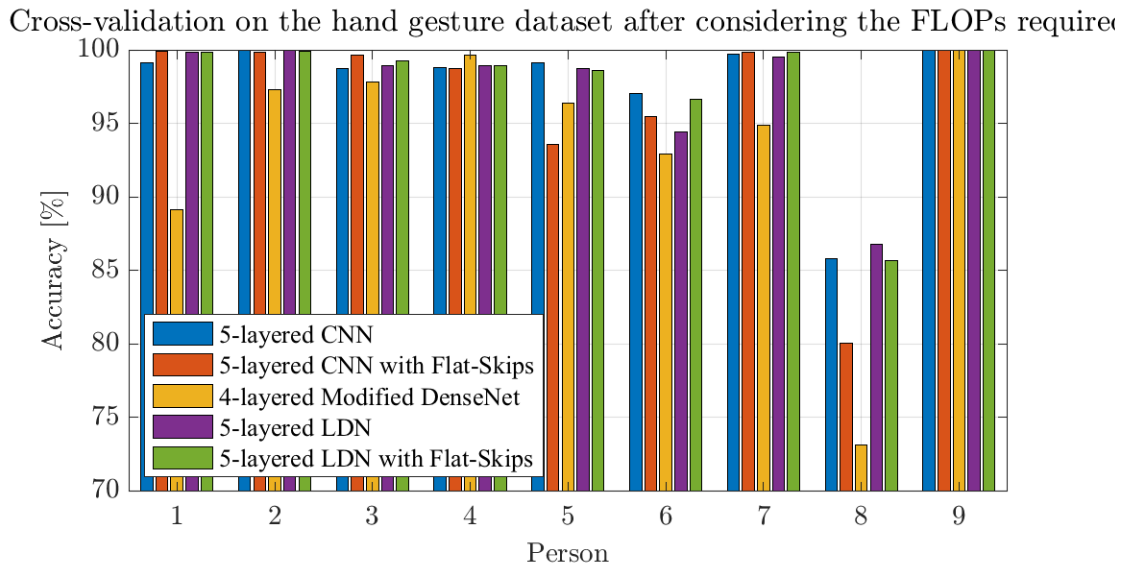


Figure 6.14: Results of the cross-validation on the hand gesture dataset in respect to the FLOPs required.

The results were summarized the following table:

Model	Layers	Accuracy	Error Reduction	Parameters	FLOPs
CNN	5	97.59 ± 0.864%	-	201k	25.1M
CNN (Flat-Skips)	5	96.33 ± 1.016%	-52.28%	274k	18.5M
Modified DenseNet	4	93.46 ± 1.276%	-171.37%	255k	26.8M
LDN	5	97.46 ± 0.882%	-5.39%	170k	28.1M
LDN (Flat-Skips)	5	97.63 ± 0.859%	1.69%	269k	24.4M

Table 6.8: Accuracies of the models after considering the required LOPs on the hand gesture dataset.

The accuracy of the LDN drops below the accuracy of the CNN, while the LDN with Flat-Skips still manages to achieve a slightly higher accuracy. All in all, the LDN with Flat-Skips does not achieve significantly better results compared to the CNN.

6.4 Experiment on the Fruits-360 dataset

The Fruits-360 dataset differentiates itself from all other datasets which were used in this work in the way that for each architecture at least one model achieved an accuracy of 100% on the evaluation dataset. In the case of the CNN and the LDN almost all models reached 100%. Because of that the model with the fewest amount of layers which achieved an accuracy of 100% were chosen for each architecture to be evaluated on the test data. Those models were the 3-layered CNN, the 7-layered CNN with Flat-Skips, the 5-layered modified DenseNet, the 3-layered LDN and the 3-layered LDN with Flat-Skips. Important to note is that for the Fruits-360 dataset no data augmentation was used. The results on the test data which consists of 22.688 images, were the following.

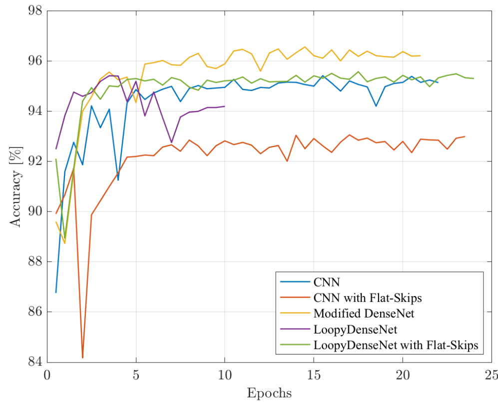


Figure 6.15: Accuracy on the test data of the Fruits-360 dataset.

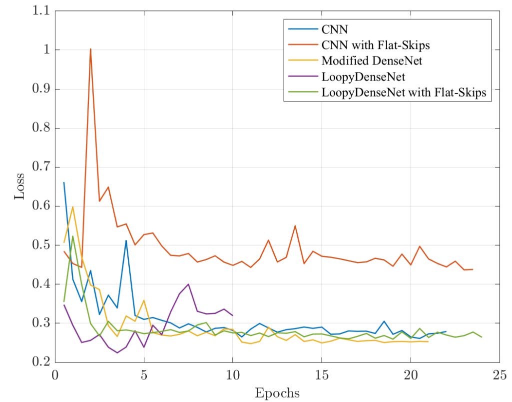


Figure 6.16: Loss on the test data of the Fruits-360 dataset.

The results were summarized in the following table. It is worth mentioning that all models achieved an accuracy of 100% on the training data.

Model	**Layers**	**Accuracy**	**Error Reduction**	**Parameters**	**FLOPs**
CNN	3	95.42 ± 0.289%	-	1863k	33.0M
CNN (Flat-Skips)	7	94.34 ± 0.318%	-23.58%	2548k	20.0M
Modified DenseNet	5	96.57 ± 0.254%	33.53%	2189k	26.0M
LDN	3	95.42 ±0.289%	-	1560k	28.0M
LDN (Flat-Skips)	3	95.58 ± 0.284%	3.62%	1985k	28.9M

Table 6.9: Performance of the best models of the respective architecture on the test data of the Fruits-360 dataset.

On the Fruits-360 dataset the modified DenseNet reached the highest accuracy and reduced the error compared to the CNN by about 33%, achieving a significantly better result. Furthermore, the LDN with Flat-Skips also reached a higher accuracy then the CNN. When considering the FLOPs the loss and the test accuracy look as follows:

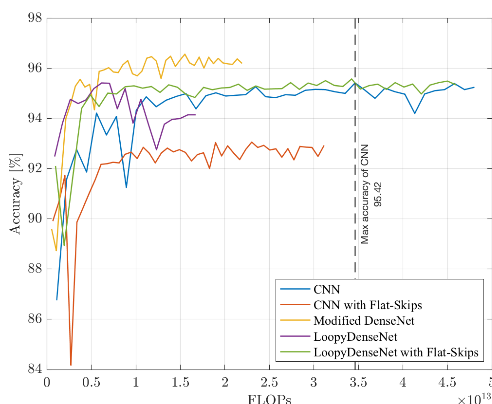


Figure 6.17: Accuracy on the test data of the Fruits-360 dataset in respect to the required FLOPs.

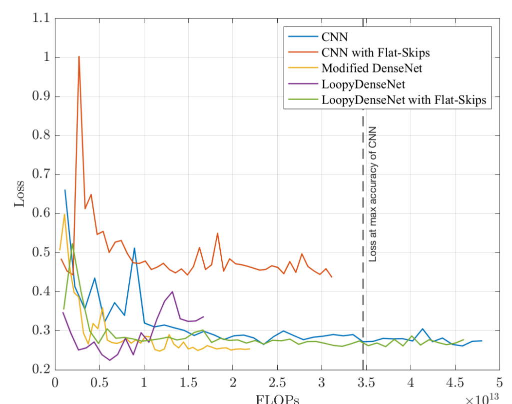


Figure 6.18: Loss on the test data of the Fruits-360 dataset in respect to the required FLOPs.

Similarly to the datasets before, all non-CNN models achieved its highest accuracy with fewer FLOPs then the CNN. In the case of the LDN, the CNN needs more then 3 times

as many FLOPs to reach the same maximum accuracy. The LDN with Flat-Skips reaches the maximum accuracy of the CNN with just a third of the FLOPs the CNN would need.

6.5 Experiment on the CIFAR-10 dataset

The CIFAR-10 dataset is the most difficult dataset to classify in this work. This dataset is a very popular benchmarking dataset and large networks were used to achieve high accuracies. Since in this work only small networks were used, no competitive results can be expected. Similarly to the other datasets the best model of each architecture is determined by evaluating the models on the evaluation data. The following table shows the best results on the evaluation data obtained by the respective architecture.

Model	Layers	Accuracy	Error Reduction	Params.	FLOPs
CNN	3	68.72%	-	191k	29.6M
CNN (Flat-Skips)	4	70.90%	6.97%	263k	23.5M
Modified DenseNet	4	68.82%	0.32%	255k	26.8M
LDN	3	70.65%	6.17%	166k	25.8M
LDN (Flat-Skips)	3	72.94%	13.49%	199k	25.8M

Table 6.10: Performance of the models of the respective architecture on the evaluation data of the CIFAR-10 dataset.

As can be seen, the LoopyDenseNet with Flat-Skips achieves the highest accuracy and reduces the error of the CNN by about 13.5%, while needing fewer FLOPs per epoch and only having 8 thousand more parameters. Furthermore, the standard LDN without Flat-Skips also achieves a higher accuracy then the CNN, indicating, that looping convolutions seem to improve performance. In order to make proper statements about the performance those models were then tested on the test data. The results were visualized in the following figures.

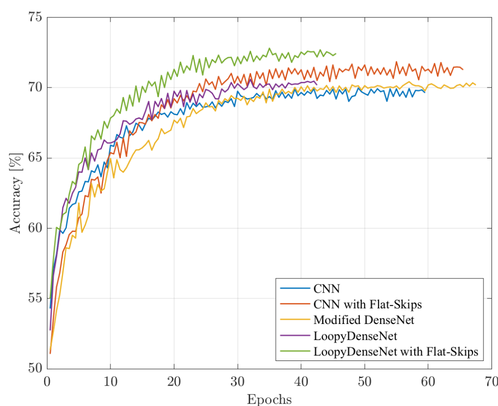


Figure 6.19: Accuracy on the test data of the CIFAR-10 dataset.

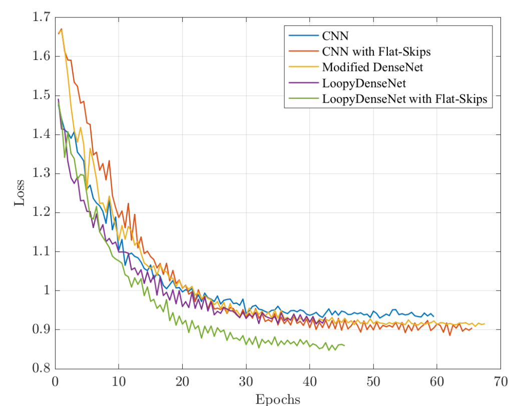


Figure 6.20: Loss on the test data of the CIFAR-10 dataset.

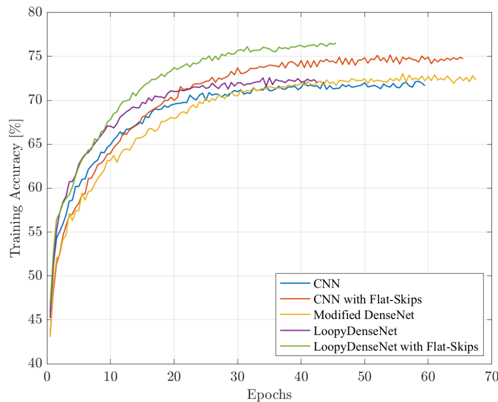


Figure 6.21: Accuracy of the different models on the training data of CIFAR-10.

Looking at the accuracy of the models on the training data a similar trend can be seen. The LDN with Flat-Skips achieves accuracies well above 75% and the CNN with Flat-Skips around 75% on the training data. Furthermore, it has to be said, that all models are overfitting. By using more data augmentation or a generalization method like dropout even higher accuracies can be expected [66].

The following table summarizes the results and also gives the error bounds (confidence interval of 95%).

Model	Layers	Accuracy	Error Reduction	Params.	FLOPs
CNN	3	69.98 ± 0.937%	-	191k	29.6M
CNN (Flat-Skips)	4	71.85 ± 0.920%	6.64%	263k	23.5M
Modified DenseNet	4	70.43 ± 0.933%	1.52%	255k	26.8M
LDN	3	70.60 ± 0.931%	2.11%	166k	25.8M
LDN (Flat-Skips)	3	72.81 ± 0.910%	10.41%	199k	25.8M

Table 6.11: Performance of the models of the respective architecture on the evaluation data of the CIFAR-10 dataset.

In the table above it is possible to see, that the LDN with Flat-Skips and the CNN with Flat-Skips significantly outperform the standard CNN model, which indicates, that Flat-Skips might be helpful for the CIFAR-10 dataset. Furthermore, also the modified DenseNet shows better results then the CNN. Compared to the previous datasets the CIFAR-10 is a much harder classification problem especially when using a small network. The results on this dataset indicate, that for harder classification problem more complex structures in the convolutional part of the network seem to be beneficial, since all non-CNN models achieve higher accuracies then the standard CNN.
In order to make a more general statement about the performance of the models, the achieved accuracy must be considered under the dependence of the required FLOPs. This can be seen in the following figures.

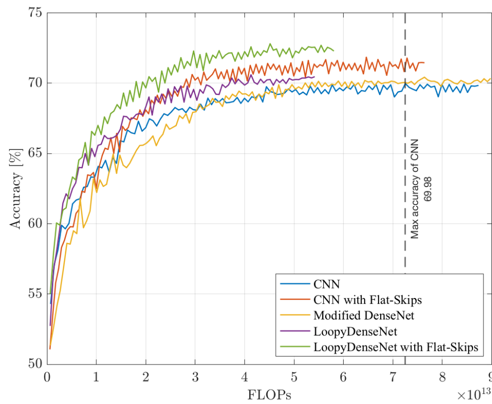


Figure 6.22: Accuracy on the test data of the CIFAR-10 dataset in respect to the FLOPs required.

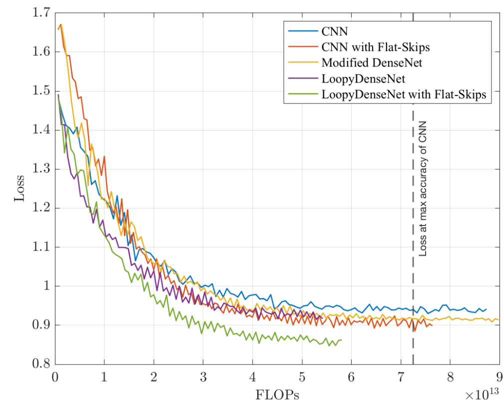


Figure 6.23: Loss on the test data of the CIFAR-10 dataset in respect to the FLOPs required.

The LDN, the LDN with Flat-Skips and the CNN with Flat-Skips learn faster then the CNN when considering the number of FLOPs. The LDN with Flat-Skips requires about 35% less FLOPs to achieve its highest accuracy in comparison to the CNN. Only the modified DenseNet does reach its highest accuracy after the CNN does. However, its performance is still above 70% and therefore higher then the accuracy of the CNN.
All in all, incorporating convolutional loops seem to be beneficial for the performance of the network. However, Flat-Skips improve the accuracy of the models more significantly. When combining both methods the overall best result can be achieved. The 3-layered LDN with Flat-Skips is capable of correctly classifying about 72.81% of the images of the test data of CIFAR-10, despite consisting of only 199k parameters. Considering, that the state of the art performance on graph classification on CIFAR-10 for models with fewer then 100k parameters is 72.84% according to [2] this result is considerable [13]. In order to compare the LDN with other models of this benchmark the 3-layered network was modified by incorporating an additional max-pooling layer and reducing the number of filters per layer. By doing so the overall parameters were reduced to 94.3k. The model consists of three convolutional layers with 20 filters in the first, 40 filters in the second and 60 filters in the third layer. After the second layer a 2×2 max-pooling operation is done, as well as, after the third layer. Furthermore, for the feature-maps of the first and the second layer a 4×4 max-pooling operation was done, before passing the feature-maps to the flatten layer. Because of that the flatten layer only consists of 5400 neurons and the model has less then 100k trainable parameters. After the convolutional part there is only a single fully-connected layer.

	Main	**Conv 1 (Loops)**	**Max-Pooling**	**Flat-Skips**
Input	$32 \times 32 \times 3$	-	-	-
Conv 1	$3 \times 3 \times 3 \times 20$	-	-	-
Output	$30 \times 30 \times 20$	-	4×4	$8 \times 8 \times 20$
Input	$30 \times 30 \times 20$	$30 \times 30 \times 20$	-	-
Conv 2	$3 \times 3 \times 20 \times 40$	$3 \times 3 \times 3 \times 20$	-	-
Output	$28 \times 28 \times 40$	$28 \times 28 \times 20$	4×4	$7 \times 7 \times 40$
Max-Pooling	$14 \times 14 \times 40$	$14 \times 14 \times 20$	-	-
Input	$14 \times 14 \times 60$	-	-	-
Conv 3	$3 \times 3 \times 60 \times 60$	-	-	-
Output	$12 \times 12 \times 60$	-	-	-
Max-Pooling	$6 \times 6 \times 60$	-	-	$6 \times 6 \times 60$
Flatten	-	-	-	5400×1
FC	-	-	-	5400×10
Prediction	-	-	-	10×1

Table 6.12: LoopyDenseNet with Flat-Skips with 94.3k parameters.

The model described above achieves an accuracy of 69.44%, which is comparable to the best submits on the CIFAR-10 100k benchmark according to [2].

6.6 Summary of the results

In the following section the results of the different network architectures on all the datasets get summarized. In order to answer the research question whether skip connections, a dense connectivity pattern and the use of loops improve the performance of a convolutional neural network the performance were compared to the results of the ordinary CNN. In the following table the results on the test data in respect to the required FLOPs were given for each dataset. Additionally, the number of layers and the number of parameters of the models were given and the FLOPs which were required to do a feedforward pass. The only exception here will be the hand gesture dataset. Here the results of the cross-validation will be given.

Data	Model	Layers	Accuracy	Params.	FLOPs
MNIST	CNN	4	99.60 ± 0.162%	140k	16.4M
	CNN (Flat-Skips)	6	99.45 ± 0.183%	223k	8.9M
	Modified DenseNet	4	99.38 ± 0.192%	211k	15.8M
	LDN	5	**99.62** ± 0.160%	187k	17.8M
	LDN (Flat-Skips)	5	99.56 ± 0.168%	238k	14.4M
Fashion-MNIST	**CNN**	4	**93.23** ± 0.531%	140k	16.4M
	CNN (Flat-Skips)	4	92.59 ± 0.552%	195k	12.5M
	Modified DenseNet	4	92.00 ± 0.570%	211k	15.8M
	LDN	3	93.12 ± 0.535%	106k	15.8M
	LDN (Flat-Skips)	3	93.01 ± 0.538%	148k	15.9M
Hand gesture	CNN	5	97.59 ± 0.864%	201k	25.1M
	CNN (Flat-Skips)	5	96.33 ± 1.016%	274k	18.5M
	Modified DenseNet	4	93.46 ± 1.276%	255k	26.8M
	LDN	5	97.46 ± 0.882%	170k	28.1M
	LDN (Flat-Skips)	5	**97.63** ± 0.859%	269k	24.4M
Fruits-360	CNN	3	95.42 ± 0.289%	1863k	33.0M
	CNN (Flat-Skips)	7	94.34 ± 0.318%	2548k	20.0M
	Modified DenseNet	5	**96.57** ± 0.254%	2189k	26.0M
	LDN	3	95.42 ±0.289%	1560k	28.0M
	LDN (Flat-Skips)	3	95.58 ± 0.284%	1985k	28.9M
CIFAR-10	CNN	3	69.98 ± 0.937%	191k	29.6M
	CNN (Flat-Skips)	4	71.85 ± 0.920%	263k	23.5M
	Modified DenseNet	4	70.43 ± 0.933%	255k	26.8M
	LDN	3	70.60 ± 0.931%	166k	25.8M
	LDN (Flat-Skips)	3	**72.81** ± 0.910%	199k	25.8M

Table 6.13: Performance of the models on the different classification datasets considering the required FLOPs.

As can be seen in the table 6.13, extending the standard CNN architecture with Flat-Skips does not improve the performance of the network in most cases. Only on the CIFAR-10 dataset Flat-Skips significantly improved the performance of the network. This indicates, that Flat-Skips on their own only improve performance on more complex datasets. For simpler datasets with easier structures Flat-Skips on their own do not add any additional value, however, increase the number of parameters and increases the chances of overfitting. The combination of Flat-Skips and a dense connectivity pattern such it is realized in the modified DenseNet also do not boost the performance compared to the CNN. Only on the Fruits-360 dataset the modified DenseNet manages to outperform all other models. Nevertheless, on other datasets it often times performed significantly worse.

The LDN and the LDN with Flat-Skips both achieved comparable results to the CNN on all datasets. On the MNIST dataset the LDN manages to achieve the overall highest performance. Also on the Fashion-MNIST dataset, when the required FLOPs were not considered. Since the LDN does not use Flat-Skips and relays on the use of convolutional loops, this network architecture is extremely parameter efficient. The LDN with Flat-Skips on the other hand achieved the highest accuracy on the hand gesture and the CIFAR-10 dataset. It manages to achieve a significantly better accuracy then the CNN model while requiring a third less FLOPs. Considering, that the CIFAR-10 dataset is the most complex dataset, this indicates, that convolutional loops and Flat-Skips might be especially useful on more complex datasets compared to the CNN.

7. Prospects of LoopyDenseNets

LoopyDenseNets are the attempt to utilize convolutional layers multiple times in a single forward run. In comparison to other neural network architectures, which also use some sort of convolutional loops within the network, like the Loopy Neural Network [29] or the Recurrent Convolutional Neural Network [44], the LoopyDenseNet does not need an additional parameter to define the number of loops. The number of loops of each layer are defined by the number of layers of the convolutional part. Furthermore, convolutional loops are used to adapt the feature-map size of feature-maps from different origins in order to be able to concatenate them. However, there are many more rooms to explore with looping convolutions. In this section some ideas were given which would be interesting to make further investigations.

7.1 Deep LoopyDenseNets

Because of computational restrictions the models tested in this work were relatively small. The deepest networks that were tested only had 7 convolutional layers. Furthermore, the convolutional part of the networks only consist of few 10k parameters. However, it would be very interesting to use the LoopyDenseNet architecture in bigger models, consisting of more layers and more parameters. Beside the performance comparison described in section 5.1, bigger models were tested on the MNIST dataset, as well as on the CIFAR-10 dataset. The goal was to see if making the model bigger results in an increase of performance. Indeed higher accuracies could be obtained by simply increasing the number of filters per layer. For the CIFAR-10 dataset a 3-layered model was defined, which has 32 filters in the first layer, 64 in the second and 96 filters in the third layer. Again max-pooling was used after the second convolutional layer. This model contains about 293k parameters and requires 35.5M FLOPs. Despite only having 100k parameters more then the best model in section 6.5, which achieves an accuracy of 72.81% on the test data and 76.5% on the training data, this model achieves an accuracy of 74.69% on the test data and an accuracy of 79.3% on the training data, which is significantly better. It is worth mentioning, that all the other parameters were the same as in the experiment above. A 5 layered LoopyDenseNet with 20 filters in the first, 40 filters in the second, 60 in the third, 100 in the fourth and 120 filters in the fifth layer achieved an accuracy of 99.69% on the test data of the MNIST dataset, which is a slight improvement over the 99.62% achieved with the smaller model. Again the exact same hyperparameters as in section 5.1 were used. The exact model and the feature-map generation looks as follows:

	Main	**Conv 1 (Loops)**	**Conv 2 (Loops)**	**Conv 3 (Loops)**
Input	$28 \times 28 \times 1$	-	-	-
Conv 1	$3 \times 3 \times 1 \times 20$	-	-	-
Output	$26 \times 26 \times 20$	-	-	-
Input	$26 \times 26 \times 20$	$26 \times 26 \times 20$	-	-
Conv 2	$3 \times 3 \times 20 \times 40$	$3 \times 3 \times 1 \times 20$	-	-
Output	$24 \times 24 \times 40$	$24 \times 24 \times 20$	-	-
Pooling	$12 \times 12 \times 40$	$12 \times 12 \times 20$	-	-
Input	$12 \times 12 \times 60$	$12 \times 12 \times 20$	$12 \times 12 \times 40$	-
Conv 3	$3 \times 3 \times 60 \times 60$	$3 \times 3 \times 1 \times 20$	$3 \times 3 \times 20 \times 40$	-
Output	$10 \times 10 \times 60$	$10 \times 10 \times 20$	$10 \times 10 \times 40$	-
Input	$10 \times 10 \times 120$	$10 \times 10 \times 20$	$10 \times 10 \times 40$	$10 \times 10 \times 60$
Conv 4	$3 \times 3 \times 120 \times 100$	$3 \times 3 \times 1 \times 20$	$3 \times 3 \times 20 \times 40$	$3 \times 3 \times 60 \times 60$
Output	$8 \times 8 \times 100$	$8 \times 8 \times 20$	$8 \times 8 \times 40$	$8 \times 8 \times 60$
Input	$8 \times 8 \times 220$	-	-	-
Conv 5	$3 \times 3 \times 220 \times 120$	-	-	-
Output	$6 \times 6 \times 120$			
Flatten	4320×1	-	-	-
FC	4320×10	-	-	-
Predicition	10×1	-	-	-

Table 7.1: Bigger 5 layered LoopyDenseNet for MNIST.

In the sections 3.4.4 and 3.4.5 concepts are presented, which can be used to design deeper LDNs and still be able to reduce the computational costs. The effectiveness of bottleneck layers were also tested on the model described above (20 - 40 - 60 - 100 - 120). When placing a 1×1 convolution layer with a depth of 60 before the last layer, which consists of 120 filters and a 1×1 convolutional layer with a depth of 40 in front of the fourth layer, which consists of 100 filters, the computation time could be reduced by about 20% while reaching almost the same accuracy. When using 1×1 convolutions the model looks like this:

	Main	**Conv 1 (Loops)**	**Conv 2 (Loops)**	**Conv 3 (Loops)**
Input	$28 \times 28 \times 1$	-	-	-
Conv 1	$3 \times 3 \times 1 \times 20$	-	-	-
Output	$26 \times 26 \times 20$	-	-	-
Input	$26 \times 26 \times 20$	$26 \times 26 \times 20$	-	-
Conv 2	$3 \times 3 \times 20 \times 40$	$3 \times 3 \times 1 \times 20$	-	-
Output	$24 \times 24 \times 40$	$24 \times 24 \times 20$	-	-
Pooling	$12 \times 12 \times 40$	$12 \times 12 \times 20$	-	-
Input	$12 \times 12 \times 60$	$12 \times 12 \times 20$	$12 \times 12 \times 40$	-
Conv 3	$3 \times 3 \times 60 \times 60$	$3 \times 3 \times 1 \times 20$	$3 \times 3 \times 20 \times 40$	-
Output	$10 \times 10 \times 60$	$10 \times 10 \times 20$	$10 \times 10 \times 40$	-
Input	$10 \times 10 \times 120$	-	-	-
Bottleneck	$1 \times 1 \times 120 \times 40$	-	-	-
Output	$10 \times 10 \times \mathbf{40}$	-	-	-
Input	$10 \times 10 \times 40$	$10 \times 10 \times 20$	$10 \times 10 \times 40$	$10 \times 10 \times 60$
Conv 4	$3 \times 3 \times 40 \times 100$	$3 \times 3 \times 1 \times 20$	$3 \times 3 \times 20 \times 40$	$3 \times 3 \times 60 \times 60$
Output	$8 \times 8 \times 100$	$8 \times 8 \times 20$	$8 \times 8 \times 40$	$8 \times 8 \times 60$
Input	$8 \times 8 \times 220$	-	-	-
Bottleneck	$1 \times 1 \times 220 \times 60$	-	-	-
Output	$8 \times 8 \times \mathbf{60}$	-	-	-
Input	$8 \times 8 \times 60$	-	-	-
Conv 5	$3 \times 3 \times 60 \times 120$	-	-	-
Output	$6 \times 6 \times 120$			
Flatten	4320×1	-	-	-
FC	4320×10	-	-	-
Predicition	10×1	-	-	-

Table 7.2: Bigger 5 layered LoopyDenseNet with 1×1 convolutions for MNIST.

As can be seen in the table the bottleneck layer reduce the depth of the feature-maps which were the input to the fourth and the fifth convolutional layer. While the fourth layer previously received a feature-map of depth 120, the feature-map depth can be reduced to an arbitrary size by using the 1×1 convolution. The same thing applies for the fifth layer. The bottleneck layers are so computational inexpensive that they reduce the computation time, since the feature-map depth of the input to the next 3×3 convolution is smaller. When using bottleneck layers one might experiment with the use of Flat-Skips. There are several possibilities on how to incorporate Flat-Skips in LDN with bottleneck layers. Fist of which, it is possible to use Flat-Skips on all convolutional layers, including the bottleneck layers. The second possibility is to only use Flat-Skips on the normal convolutions and lastly it is possible to use Flat-Skips only on the output of the bottleneck layers when it is used. This would be a very parameter efficient implementation of Flat-Skips. In the example above Flat-Skips are only applied on the normal convolutional layers. Because of that the flatten layer has the same size as the network without bottleneck layers.
Another approach would be to use the equivalent of a dense block, which are used in the

original DenseNet and are also mentioned in section 2.1 [26]. When adding more layers to the network the number of loops grow as well, which leads to an massive increase in the feature-map depth and an increase in computation costs. When splitting up the network in blocks, the feature-map depth can be controlled. Only feature-maps of layers which belong to the same block will be concatenated. In a deep LDN with a block size of 5 layers, the first convolutional layer in this block gets looped 3 more times. The second layer gets looped two times and the third layer only once. The output of the last convolutional layer of the block goes to the first convolutional layer of the next block. Another way to utilize blocks in LDNs and to make the network more parameter efficient is to only use Flat-Skips at the end of a block. That means, that not every convolutional layer is directly connected to the flatten layer, however, just to the last convolutional layer of a block.

7.2 LDNs without pooling

As mentioned several times in this work no padding was used during a convolutional operation in a LDN, resulting in a reduction of the dimensions. This was done in order to minimize the use of pooling operations, since pooling is primarily used for down-sampling. When using pooling, information about the exact position of a feature gets lost. This is problematic, since the relative position between features can be very important to correctly detect and classify objects and therefore pooling layers should be avoided [43] [9] [60]. The reason why pooling was used during this work was mainly to reduce the computational costs. However, it would be interesting to build LDNs which only relay on convolutional operations for down-sampling. Using an ensemble of CNNs which do not use pooling layers achieved state of the art performance on the MNIST dataset, reaching an accuracy of 99.91% [7]. By doing so the effects of the loops might be reinforced. When considering, that loops can have a shifting effect, which gets stronger with each loop, more detailed features could be detected.
In LDNs only a single convolutional layer gets looped. In Loopy Neural Networks the whole convolutional part of the network gets looped [29]. However, what happens when any number of layers gets looped? When using a neural network which is not using any pooling layers, looping multiple layers might be easier to implement. Considering a LDN which consist of 13 layers and is not using any pooling layers, it is possible to loop the first and the second convolutional layer five times, for example, and still be able to concatenate all the generated feature-maps with the corresponding layer. This method could be extended so that any combination of loops are used in a single LDN. The number of possible loops will be limited to the effective depth of the network. It would be particularly interesting what effect this approach would have on the performance of the network and what effects looping multiple convolutional layers have on the generated

feature-maps. This approach would be extremely parameter efficient, while still being able to detect complex features. However, the backpropagation of such network might be more difficult to implement, since the standard backpropagation through time cannot be applied here without doing some adaptations.
Since a ordinary convolutional operation reduces the dimensions of feature-maps just slightly (in the case of a 3×3 convolution, the dimensions get reduced by two), there is the possibility to use a stride greater then one. This can replace the ordinary pooling operation. When the stride is selected properly information of the whole feature-map is considered. Considering a 3×3 convolution with stride two (depending on the dimensions of the input feature-maps), the dimensions of the resulting feature-maps could be up to 80% smaller. Using a stride of three up to 89% [48] [30].

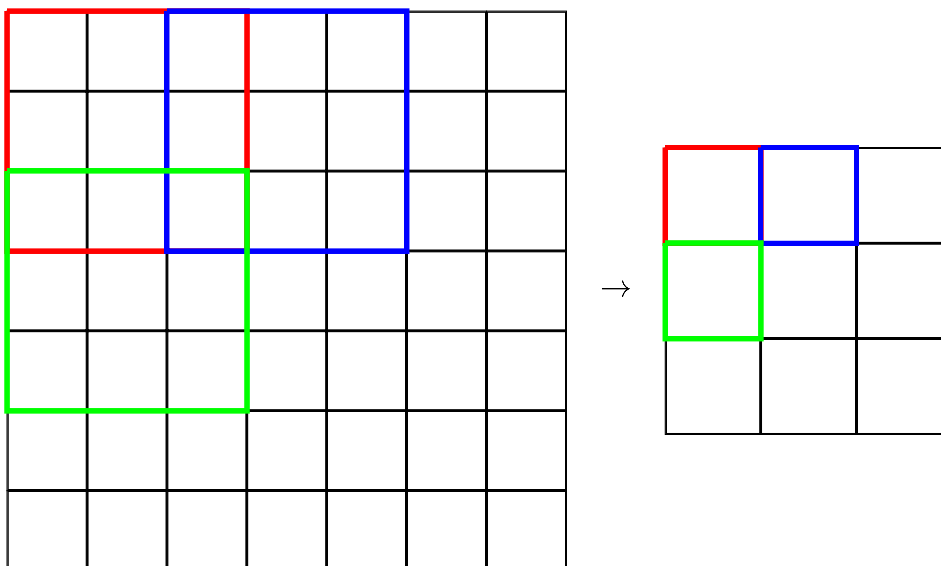


Figure 7.1: 3×3 convolution with stride 2 on an input with the dimensions 7×7; resulting in a 3×3 matrix

In the figure above an example of a 3×3 convolution is given, which uses a stride of 2. That means, that the filter is moved two steps instead of just one, which results in a bigger reduction of dimensionality and therefore could replace the pooling operation.

7.3 LDNs with bigger filter sizes

Since the AlexNet it is standard to use 3×3 convolutions [34]. Because of that in the experiments in section 5 only 3×3 convolutions were used. However, further testing was done afterwards, where the 3×3 convolutions got replaced with 5×5 convolutions. This was done for the 3-layered LDN with Flat-Skips, which with 3×3 convolutions achieved an accuracy of 72.81% on the test data and an accuracy of 76.5% on the training data of the CIFAR-10 dataset. When using 5×5 convolutions an accuracy of 74.64% could be achieved on the test data and an accuracy 76.7% on the training data using the exact

same model an the same number of filters per layer (20 - 40 - 80). Again no padding was used, resulting in a dimension reduction of the feature-maps when applying convolutions. The exact model looks as follows:

	Main	**Conv 1 (Loops)**
Input	$32 \times 32 \times 3$	-
Conv 1	$5 \times 5 \times 3 \times 20$	-
Output	$28 \times 28 \times 20$	-
Input	$28 \times 28 \times 20$	$12 \times 12 \times 24$
Conv 2	$5 \times 5 \times 20 \times 40$	$5 \times 5 \times 3 \times 20$
Output	$24 \times 24 \times 40$	$24 \times 24 \times 20$
Pooling	$12 \times 12 \times 40$	$12 \times 12 \times 20$
Input	$12 \times 12 \times 60$	-
Conv 3	$5 \times 5 \times 60 \times 80$	-
Output	$8 \times 8 \times 80$	-
Flatten	7540×1	-
FC	7540×10	-
Predicition	10×1	-

Table 7.3: 3-layered LoopyDenseNet with Flat-Skips and 5×5 convolutions for the CIFAR-10 dataset.

Interestingly the loss is even lower with the 5×5 LDN with Flat-Skip, then the loss of the bigger LDN with Flat-Skips (32 - 64 - 96), which is used in the section 7.1 and uses 3×3 convolutions. For completion also the bigger model, which was used in section 7.1, with 32 filters in the first, 64 in the second and 96 in the third were tested with 5×5 convolutions, combining bigger filter sizes with more filters per layer. This results in the overall best accuracy of 76.62% on the test data and 80.57% on the training data.

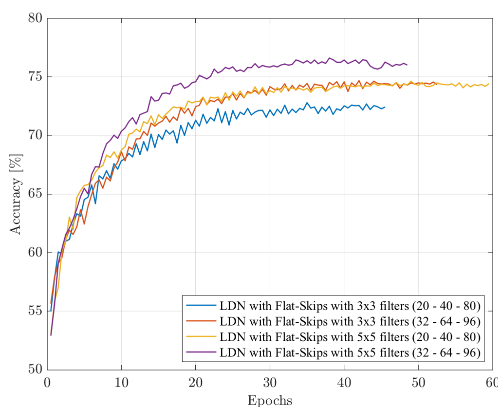


Figure 7.2: Accuracy on the test data of the CIFAR-10 dataset using different filter sizes and different number of filters per layer.

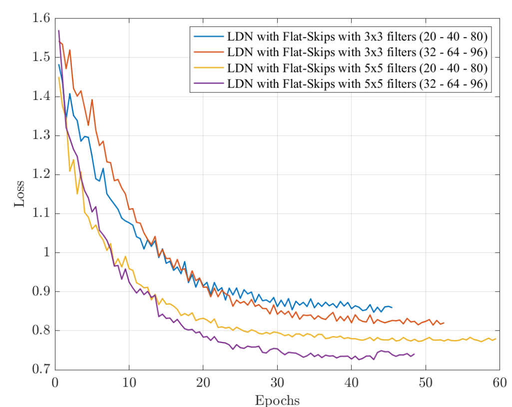


Figure 7.3: Loss on the test data of the CIFAR-10 dataset using different filter sizes and different number of filters per layer.

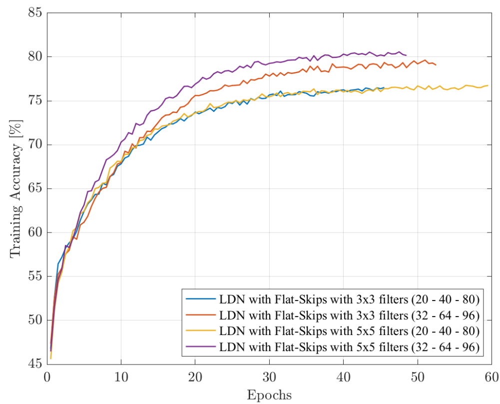


Figure 7.4: Accuracy on the training data of the CIFAR-10 dataset using different filter sizes and different number of filters per layer.

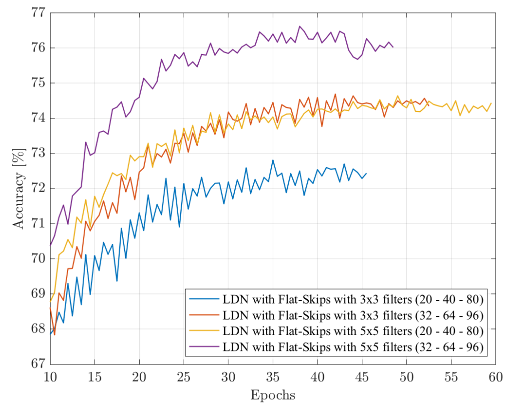


Figure 7.5: Accuracy on the test data of the CIFAR-10 dataset using different filter sizes and different number of filters per layer beginning from the 10-th epoch.

Because of this high performance of LDNs with 5×5 convolutions it would be interesting to test models with even larger filter sizes. By using bigger filter sizes pooling layers could be avoided, since they reduce the dimensionality of the feature-maps even more then the standard 3×3 convolution.

8. Conclusion

In this section the research questions of this thesis will be answered and the most interesting findings will be summarized. All in all, it can be said that simply adding additional connections does not necessarily make the network better. Depending on the dataset, more complex structures and operations in the convolutional part of the network can reduce the accuracy, because the model overfits. This can be seen especially in simpler classification problems. However, for more complex datasets, additional connections, the looping of layers and feature-map concatenation seem to increase accuracy and are indeed beneficial for the performance of the network.

8.1 Research Question 1 - Flat-Skips

Can skip connections, which directly connect each convolutional layer with the flatten layer improve the performance of CNNs?

Flat-Skips are an easy method to improve the gradient flow of the network. By adding an additional connection from every convolutional layer to the flatten layer each layer has a much closer path to the loss. Furthermore, it strengthens feature propagation and it provides all the feature-maps that get created throughout the convolutional part of the network to the flatten layer, making more information available for classification. These properties should be helpful for deeper models which consist of many more layers. However, since the models tested in this work were all relatively shallow and consisted of few parameters, Flat-Skips didn't increase the performance. Only on the CIFAR-10 dataset it manages to surpass the results of the standard CNN significantly, which indicates, that additional connections within the network can indeed increase the performance of the network, however, this depends on the dataset.

8.2 Research Question 2 - Modified DenseNet

Can a dense connectivity pattern combined with Flat-Skips be successfully applied on small convolutional neural networks and outperform the ordinary CNN while having comparable computational costs?

In order to be able to create a dense connectivity pattern in a small neural network similarly to a DenseNet, no padding gets used during convolution, which eliminates the

need for dense blocks and transition layers [26]. However, in order to still be able to concatenate feature-maps of different layers an adaptive pooling algorithm was developed. The adaptive pooling operation can reduce the dimensions of any feature-map to the desired format. This operation creates a unique solution so that backpropagation can be applied as with an ordinary pooling operation. Furthermore, adaptive pooling allows the use of pooling layers within the densely connected convolutional layers, which in the original DenseNet was not possible [26]. Furthermore, this modified version of the DenseNet also incorporates Flat-Skips, which should increase the gradient flow.
Despite the success of the DenseNet, applying a dense connectivity pattern on shallow models does not improve the performance in most cases compared to the standard CNN. The Fruits-360 dataset is the only dataset, where the modified DenseNet achieved to significantly outperform the CNN.

8.3 Research Question 3 - LoopyDenseNet

How can convolutional layers be looped in a convolutional neural network and how does such network perform in comparison to the ordinary CNN?

In this work a new convolutional neural network architecture is proposed which is looping single convolutional layers in order to create a dense connectivity pattern by concatenating feature-maps from different layers. The LoopyDenseNet architecture can directly be applied on any ordinary CNN and no additional parameters were needed. In comparison to the modified DenseNet which uses adaptive pooling to adjust the feature-map dimensions in order to be able to concatenate them, LDNs achieve the desired feature-map size by looping convolutions. Because of that LDNs do not have skip connections within the convolutional part. In order to improve the gradient flow within the LDN Flat-Skips can be included, which seem to pair well with the loopy architecture. By looping convolutions LDNs are capable of detecting more complex features, while still being extremely parameter efficient.
In the empirical part of this work it is shown, that the LDN and the LDN with Flat-Skips are both capable of achieving higher or similar results as the CNN, while having comparable computational costs. Especially on the CIFAR-10 dataset the LDN with Flat-Skips outperforms the CNN model significantly, while needing fewer FLOPs in order to achieve its highest accuracy. Since the classification problem of the CIFAR-10 dataset is much more difficult than the other datasets used in this work, this result suggests that convolutional loops could be useful for more complex classification tasks. However, LDNs and LDNs with Flat-Skips are still capable to achieve similar results as the ordinary CNN on easier classification task. Furthermore, LDNs can be extremely parameter efficient.
The LoopyDenseNet architecture was only tested for small networks, due to computa-

tional constraints. When increasing the number of filters per layer it is shown, that the LDN also achieves higher performance. In future work it would be interesting to build deep LDNs consisting of many more layers and parameters and test them on more complex datasets. Especially deep LDNs with a looping limit and bottleneck layers could be a promising network to make further investigations on. Looping limits and 1×1 convolutions can both be used to decrease the feature-map depth and make the network more efficient. Deep LDNs are particularly interesting since with increasing number of layers the effects of loops should be bigger as well.
With LDNs, Loopy neural networks and Recurrent Convolutional Neural Networks only a small room of neural networks which utilize convolutional loops in computer vision were explored. There is still a huge area for further investigations regarding convolutional loops and feedback mechanics in convolutional neural networks for visual object classification.

References

[1] Meta AI. *Fine-Grained Image Classification on Fruits-360*. https://paperswithcode.com/sota/fine-grained-image-classification-on-fruits. Online; accessed 8 February 2022.

[2] Meta AI. *Graph Classification on CIFAR-10 100k*. https://paperswithcode.com/sota/graph-classification-on-cifar10-100k. Online; accessed 21 April 2022.

[3] Meta AI. *Image Classification on CIFAR-10*. https://paperswithcode.com/sota/image-classification-on-cifar-10. Online; accessed 8 February 2022.

[4] Meta AI. *Image Classification on Fashion-MNIST*. https://paperswithcode.com/sota/image-classification-on-fashion-mnist. Online; accessed 8 February 2022.

[5] Meta AI. *Image Classification on MNIST*. https://paperswithcode.com/sota/image-classification-on-mnist. Online; accessed 8 February 2022.

[6] Md Zahangir Alom et al. "The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches". In: *arXiv:1803.01164 [cs]* (Sept. 2018). arXiv: 1803.01164.

[7] Sanghyeon An et al. "An Ensemble of Simple Convolutional Neural Network Models for MNIST Digit Recognition". In: *arXiv:2008.10400 [cs]* (Oct. 2020). arXiv: 2008.10400.

[8] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. "Layer Normalization". In: *arXiv:1607.06450 [cs, stat]* (July 2016). arXiv: 1607.06450.

[9] Adam Byerly, Tatiana Kalganova, and Ian Dear. "No routing needed between capsules". en. In: *Neurocomputing* 463 (Nov. 2021), pp. 545–553. ISSN: 09252312. DOI: 10.1016/j.neucom.2021.08.064.

[10] Francois Chollet. "Xception: Deep Learning with Depthwise Separable Convolutions". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition*

(CVPR). Honolulu, HI: IEEE, July 2017, pp. 1800–1807. ISBN: 978-1-5386-0457-1. DOI: 10.1109/CVPR.2017.195.

[11] Junyoung Chung et al. “Gated Feedback Recurrent Neural Networks”. In: Proceedings of the 32Nd International Conference on International Conference on Machine Learning.Volume 37 (June 2015). arXiv: 1502.02367, pp. 2067–2075.

[12] Dariel Dato-on. *MNIST in CSV*. http://yann.lecun.com/exdb/mnist/. Online; accessed 8 February 2022.

[13] Dominique Beaini et al. “Directional Graph Networks”. In: *Proceedings of the 38 th International Conference on Machine Learning*. Apr. 2021.

[14] Alexey Dosovitskiy et al. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: ICLR 2021 (2020). Publisher: arXiv Version Number: 2. DOI: 10.48550/ARXIV.2010.11929.

[15] Michal Drozdzal et al. “The Importance of Skip Connections in Biomedical Image Segmentation”. In: *Deep Learning and Data Labeling for Medical Applications*. Ed. by Gustavo Carneiro et al. Vol. 10008. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 179–187. ISBN: 978-3-319-46975-1 978-3-319-46976-8. DOI: 10.1007/978-3-319-46976-8_19.

[16] B. Fernandez, A.G. Parlos, and W.K. Tsai. “Nonlinear dynamic system identification using artificial neural networks (ANNs)”. In: *1990 IJCNN International Joint Conference on Neural Networks*. San Diego, CA, USA: IEEE, 1990, 133–141 vol.2. DOI: 10.1109/IJCNN.1990.137706.

[17] Pierre Foret et al. “Sharpness-Aware Minimization for Efficiently Improving Generalization”. In: *ICLR 2021* (Apr. 2021). arXiv: 2010.01412.

[18] Glorot, Xavier and Bengio, Y. “Understanding the difficulty of training deep feedforward neural networks”. In: Journal of Machine Learning Research - Proceedings Track.9 (Jan. 2010), pp. 249–256.

[19] Glorot, Xavier, Bengio, Yoshua, and Bordes, Antoine. “Deep Sparse Rectifier Neural Networks”. In: Journal of Machine Learning Research.15 (Jan. 2010).

[20] GTI. *Hand Gesture Recognition Database*. https://www.kaggle.com/gti-upm/leapgestrecog. Online; accessed 8 February 2022. Sept. 2018.

[21] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, June 2016, pp. 770–778. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.90.

[22] Geoffrey E. Hinton and Vinod Nair. “Rectified Linear Units Improve Restricted Boltzmann Machines”. In: *In Proc. 27th International Conference on Machine Learning*. 2010.

[23] Geoffrey E. Hinton et al. “Improving neural networks by preventing co-adaptation of feature detectors”. In: *arXiv:1207.0580 [cs]* (July 2012). arXiv: 1207.0580.

[24] Daiki Hirata and Norikazu Takahashi. “Ensemble learning in CNN augmented with fully connected subnetworks”. In: *arXiv:2003.08562 [cs, eess]* (Mar. 2020). arXiv: 2003.08562.

[25] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. en. In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/neco.1997.9.8.1735.

[26] Gao Huang et al. “Densely Connected Convolutional Networks”. In: *CVPR 2017* (Aug. 2016). arXiv: 1608.06993.

[27] Apple Inc. *Apple stellt den M1 vor*. https://www.apple.com/at/newsroom/2020/11/apple-unleashes-m1/. Online; accessed 3 May 2022. Nov. 2020.

[28] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Proceedings of the 32nd International Conference on Machine Learning*. arXiv: 1502.03167. Lille, Mar. 2015.

[29] Isaac Caswell, Chuanqi Shen, and Lisa Wang. “Loopy neural nets: Imitating feedback loops in the human brain”. In: *CS231n Report, Stanford* (2016).

[30] Wu Jianxin. “Introduction to Convolutional Neural Networks”. National Key Lab for Novel Software Technology. Nanjing University, May 2017.

[31] H. M. Dipu Kabir et al. “SpinalNet: Deep Neural Network with Gradual Input”. In: *arXiv:2007.03347 [cs, eess]* (Jan. 2022). arXiv: 2007.03347.

[32] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *ICLR 2015* (Jan. 2017). arXiv: 1412.6980.

[33] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. *The CIFAR-10 dataset*. `https://www.cs.toronto.edu/~kriz/cifar.html`. Online; accessed 8 February 2022.

[34] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25*. 2012.

[35] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. “FractalNet: Ultra-Deep Neural Networks without Residuals”. In: *ICLR 2017* (May 2017). arXiv: 1605.07648.

[36] Y. Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324. ISSN: 00189219. DOI: `10.1109/5.726791`.

[37] Yann LeCun and Corinna Cortes. *THE MNIST DATABASE of handwritten digits*. `http://yann.lecun.com/exdb/mnist/`. Online; accessed 8 February 2022.

[38] Yann A. LeCun et al. “Efficient BackProp”. en. In: *Neural Networks: Tricks of the Trade*. Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. Vol. 7700. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 9–48. ISBN: 978-3-642-35288-1 978-3-642-35289-8. DOI: `10.1007/978-3-642-35289-8_3`.

[39] Li Deng. “The MNIST Database of Handwritten Digit Images for Machine Learning Research”. In: *IEEE Signal Processing Magazine* 29.6 (Nov. 2012), pp. 141–142. ISSN: 1053-5888. DOI: `10.1109/MSP.2012.2211477`.

[40] Min Lin, Qiang Chen, and Shuicheng Yan. “Network In Network”. In: *CoRR* (2013). Publisher: arXiv Version Number: 3. DOI: `10.48550/ARXIV.1312.4400`.

[41] Tomás Mantecón et al. “Hand Gesture Recognition Using Infrared Imagery Provided by Leap Motion Controller”. In: *Advanced Concepts for Intelligent Vision Systems*. Ed. by Jacques Blanc-Talon et al. Vol. 10016. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 47–57. ISBN: 978-3-319-48679-6 978-3-319-48680-2. DOI: `10.1007/978-3-319-48680-2_5`.

[42] Dominic Masters and Carlo Luschi. “Revisiting Small Batch Training for Deep Neural Networks”. In: *arXiv:1804.07612 [cs, stat]* (Apr. 2018). arXiv: 1804.07612.

[43] Vittorio Mazzia, Francesco Salvetti, and Marcello Chiaberge. “Efficient-CapsNet: capsule network with self-attention routing”. en. In: *Scientific Reports* 11.1 (Dec. 2021), p. 14634. ISSN: 2045-2322. DOI: 10.1038/s41598-021-93977-0.

[44] Ming Liang and Xiaolin Hu. “Recurrent convolutional neural network for object recognition”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Boston, MA, USA: IEEE, June 2015, pp. 3367–3375. ISBN: 978-1-4673-6964-0. DOI: 10.1109/CVPR.2015.7298958.

[45] Sridhar Narayan. “The generalized sigmoid activation function: Competitive supervised learning”. en. In: *Information Sciences* 99.1-2 (June 1997), pp. 69–82. ISSN: 00200255. DOI: 10.1016/S0020-0255(96)00200-9.

[46] Arild Nøkland and Lars Hiller Eidnes. “Training Neural Networks with Local Error Signals”. In: *ICML* 2019 (May 2019). arXiv: 1901.06656.

[47] Chigozie Nwankpa et al. “Activation Functions: Comparison of trends in Practice and Research for Deep Learning”. In: *arXiv:1811.03378 [cs]* (Nov. 2018). arXiv: 1811.03378.

[48] Keiron O’Shea and Ryan Nash. “An Introduction to Convolutional Neural Networks”. In: *arXiv:1511.08458 [cs]* (Dec. 2015). arXiv: 1511.08458.

[49] Mihai Oltean. *Fruits 360*. https://www.kaggle.com/moltean/fruits. Online; accessed 8 February 2022. Sept. 2021.

[50] Mihai Oltean. *Fruits-360: A dataset of images containing fruits and vegetables*. https://github.com/Horea94/Fruit-Images-Dataset. Online; accessed 8 February 2022. May 2020.

[51] Manish Panicker. “Efficient FPGA Implementation of Sigmoid and Bipolar Sigmoid Activation Functions for Multilayer Perceptrons”. In: *IOSR Journal of Engineering* 02.06 (June 2012), pp. 1352–1356. ISSN: 22788719, 22503021. DOI: 10.9790/3021-026113521356.

[52] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “On the difficulty of training Recurrent Neural Networks”. In: *Proceedings of the 30th International Conference on Ma- chine Learning* Volume 28 (Feb. 2013). arXiv: 1211.5063.

[53] Dabal Pedamonti. “Comparison of non-linear activation functions for deep neural networks on MNIST classification task”. In: *arXiv:1804.02763 [cs, stat]* (Apr. 2018). arXiv: 1804.02763.

[54] Luis Perez and Jason Wang. “The Effectiveness of Data Augmentation in Image Classification using Deep Learning”. In: *arXiv:1712.04621 [cs]* (Dec. 2017). arXiv: 1712.04621.

[55] B.T. Polyak. “Some methods of speeding up the convergence of iteration methods”. en. In: *USSR Computational Mathematics and Mathematical Physics* 4.5 (Jan. 1964), pp. 1–17. ISSN: 00415553. DOI: 10.1016/0041-5553(64)90137-5.

[56] Ning Qian. “On the momentum term in gradient descent learning algorithms”. en. In: *Neural Networks* 12.1 (Jan. 1999), pp. 145–151. ISSN: 08936080. DOI: 10.1016/S0893-6080(98)00116-6.

[57] Pavlo M. Radiuk. “Impact of Training Set Batch Size on the Performance of Convolutional Neural Networks for Diverse Datasets”. In: *Information Technology and Management Science* 20.1 (Jan. 2017). ISSN: 2255-9094. DOI: 10.1515/itms-2017-0003.

[58] Zalando Research. *Fashion MNIST*. https://www.kaggle.com/zalando-research/fashionmnist. Online; accessed 8 February 2022. Dec. 2017.

[59] David E Rumelhart et al. *Parallel distributed processing. explorations in the microstructure of cognition*. English. OCLC: 990582444. 1987. ISBN: 978-0-262-18120-4 978-0-262-29140-8.

[60] Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. “Dynamic Routing Between Capsules”. In: *NIPS 2017* (Nov. 2017). arXiv: 1710.09829.

[61] Shibani Santurkar et al. “How Does Batch Normalization Help Optimization?” In: *NeurIPS 2018* arXiv:1805.11604 [cs, stat]. Corpus ID: 53104146 (Apr. 2019). arXiv: 1805.11604.

[62] Robin M. Schmidt. "Recurrent Neural Networks (RNNs): A gentle Introduction and Overview". In: *arXiv:1912.05911 [cs, stat]* (Nov. 2019). arXiv: 1912.05911.

[63] Zalando SE. *fashion-mnist*. `https://github.com/zalandoresearch/fashion-mnist`. Online; accessed 8 February 2022. Aug. 2017.

[64] Sebastian Bruch et al. "An Analysis of the Softmax Cross Entropy Loss for Learning-to-Rank with Binary Relevance". In: *Proceedings of the 2019 ACM SIGIR International Conference on the Theory of Information Retrieval (ICTIR 2019)*. 2019, pp. 75–78.

[65] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *ICLR 2015* (Apr. 2015). arXiv: 1409.1556.

[66] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: The Journal of Machine Learning Research (Jan. 2014).

[67] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. "Highway Networks". In: *ICML 2015 Deep learning workshop* (Nov. 2015). arXiv: 1505.00387.

[68] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. "Training Very Deep Networks". In: *Advances in Neural Information Processing Systems 2015* (Nov. 2015). arXiv: 1507.06228.

[69] Christian Szegedy et al. "Going Deeper with Convolutions". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015.

[70] Christian Szegedy et al. "Rethinking the Inception Architecture for Computer Vision". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, June 2016, pp. 2818–2826. ISBN: 978-1-4673-8851-1. DOI: `10.1109/CVPR.2016.308`.

[71] Hong Hui Tan and King Hann Lim. "Vanishing Gradient Mitigation with Deep Learning Neural Network Optimization". In: *2019 7th International Conference on Smart Computing & Communications (ICSCC)*. Sarawak, Malaysia, Malaysia: IEEE, June 2019, pp. 1–4. ISBN: 978-1-72811-557-3. DOI: `10.1109/ICSCC.2019.8843652`.

[72] Mingxing Tan and Quoc V. Le. “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”. In: *ICML 2019* (2019). Publisher: arXiv Version Number: 5. DOI: 10.48550/ARXIV.1905.11946.

[73] Muhammad Suhaib Tanveer, Muhammad Umar Karim Khan, and Chong-Min Kyung. “Fine-Tuning DARTS for Image Classification”. In: *2020 25th International Conference on Pattern Recognition (ICPR)*. Milan, Italy: IEEE, Jan. 2021, pp. 4789–4796. ISBN: 978-1-72818-808-9. DOI: 10.1109/ICPR48806.2021.9412221.

[74] Andreas Veit, Michael Wilber, and Serge Belongie. “Residual Networks Behave Like Ensembles of Relatively Shallow Networks”. In: *NIPS 2016* (2016). Publisher: arXiv Version Number: 2. DOI: 10.48550/ARXIV.1605.06431.

[75] Han Xiao, Kashif Rasul, and Roland Vollgraf. “Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms”. In: *arXiv:1708.07747 [cs, stat]* (Sept. 2017). arXiv: 1708.07747.

[76] Matthew D. Zeiler and Rob Fergus. “Stochastic Pooling for Regularization of Deep Convolutional Neural Networks”. In: *ICLR 2013* 1st International Conference on Learning Representations (2013). Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.1301.3557.

[77] Zhun Zhong et al. “Random Erasing Data Augmentation”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.07 (Apr. 2020), pp. 13001–13008. ISSN: 2374-3468, 2159-5399. DOI: 10.1609/aaai.v34i07.7000.

List of Tables

3.1 Input of the l-th layer in relation to the input I_1 of a LoopyDenseNet . . . 21
3.2 Input of the l-th layer in relation to the input I_1 of a CNN 21
3.3 Feature-map generation of a 5 layered LoopyDenseNet. 23
3.4 Feature-map progression without looping limit. 31
3.5 Feature-map progression with looping limit of 2. 32

5.1 Base model: 4-layered CNN 43
5.2 7 layered LoopyDenseNet for the MNIST and Fashion-MNIST dataset. . . 44
5.3 7 layered CNN with Flat-Skips 44
5.4 Models for the MNIST and Fashion-MNIST dataset. 46
5.5 Models for the hand gesture dataset. 47
5.6 Models for the CIFAR-10 dataset. 48
5.7 Models for the Fruits-360 dataset. 49

6.1 Performance of the best models of the respective architecture on the evaluation data of the MNIST dataset. 50
6.2 Performance on the test data of the MNIST dataset. 51
6.3 Performance of the best models of the respective architecture on the evaluation data of the Fashion-MNIST dataset. 53
6.4 Performance on the test data of the Fashion-MNIST dataset. 54
6.5 Performance on the test data of the Fashion-MNIST dataset considering the required FLOPs. 54
6.6 Performance of the best models of the respective architecture on the evaluation data of the hand gesture dataset. 55
6.7 Results of the cross-validation on the hand gesture dataset. 56
6.8 Accuracies of the models after considering the required LOPs on the hand gesture dataset. 57
6.9 Performance of the best models of the respective architecture on the test data of the Fruits-360 dataset. 58
6.10 Performance of the models of the respective architecture on the evaluation data of the CIFAR-10 dataset. 59
6.11 Performance of the models of the respective architecture on the evaluation data of the CIFAR-10 dataset. 60

6.12 LoopyDenseNet with Flat-Skips with 94.3k parameters. 62
6.13 Performance of the models on the different classification datasets considering the required FLOPs. 63

7.1 Bigger 5 layered LoopyDenseNet for MNIST. 65
7.2 Bigger 5 layered LoopyDenseNet with 1×1 convolutions for MNIST. . . . 66
7.3 3-layered LoopyDenseNet with Flat-Skips and 5×5 convolutions for the CIFAR-10 dataset. 69

List of Figures

2.1 DenseNet of paper 5
2.2 Loopy neural net of paper 7
2.3 Comparison between a Feedforward Neural Network and a Recurrent Neural Network [62]. 9
2.4 Recurrent Neural Network unrolled [62]. 9
2.5 Comparison between the Convolutional Neural Network, the Recurrent Multi-Layered Perceptron and the Recurrent Convolutional Neural Network [44]. 11

3.1 Convolutional part of an ordinary CNN: The flatten layer just consists of the feature maps of the last convolutional layer. 14
3.2 Convolutional part with Flat-Skip connections: The flatten layer consists of the concatenated feature maps of all convolutional layers. 14
3.3 8×8 input matrix for adaptive pooling 17
3.4 Convolutional part of a LoopyDenseNet with Flat-Skips 24
3.5 Feature-maps of the looped filter 3.8. 27
3.6 Feature-maps of the looped filter 3.9. 27
3.7 Feature-maps of the looped filter 3.10. 28
3.8 Feature-maps of the looped filter 3.11. 28
3.9 Feature-maps of the randomly initialized filter applied on the image of the digit five which was shifted 6 pixels to the left. 29
3.10 Feature-maps of the randomly initialized filter applied on the image of the digit five which was shifted 8 pixels to the left. 29

6.1 Accuracy on the test data of the MNIST dataset. 51
6.2 Loss on the test data of the MNIST dataset. 51
6.3 Accuracy on the test data of the MNIST dataset beginning from the 10-th epoch. 51
6.4 Accuracy on the test data of the MNIST dataset in respect to the FLOPs required. 52
6.5 Loss on the test data of the MNIST dataset in respect to the FLOPs required. 52
6.6 Accuracy on the test data of the Fashion-MNIST dataset. 53
6.7 Loss on the test data of the Fashion-MNIST dataset. 53

6.8 Accuracy on the test data of the Fashion-MNIST dataset beginning from the 10-th epoch. 53
6.9 Accuracy on the test data of the Fashion-MNIST dataset in respect to the FLOPs required. 54
6.10 Loss on the test data of the Fashion-MNIST dataset in respect to the FLOPs required. 54
6.11 Results of the cross-validation on the hand gesture dataset. 55
6.12 Accuracy on the test data of the hand gesture dataset in respect to the FLOPs required. 56
6.13 Accuracy on the training data of the hand gesture dataset in respect to the FLOPs required. 56
6.14 Results of the cross-validation on the hand gesture dataset in respect to the FLOPs required. 57
6.15 Accuracy on the test data of the Fruits-360 dataset. 58
6.16 Loss on the test data of the Fruits-360 dataset. 58
6.17 Accuracy on the test data of the Fruits-360 dataset in respect to the required FLOPs. 58
6.18 Loss on the test data of the Fruits-360 dataset in respect to the required FLOPs. 58
6.19 Accuracy on the test data of the CIFAR-10 dataset. 59
6.20 Loss on the test data of the CIFAR-10 dataset. 59
6.21 Accuracy of the different models on the training data of CIFAR-10. 60
6.22 Accuracy on the test data of the CIFAR-10 dataset in respect to the FLOPs required. 61
6.23 Loss on the test data of the CIFAR-10 dataset in respect to the FLOPs required. 61

7.1 3×3 convolution with stride 2 on an input with the dimensions 7×7; resulting in a 3×3 matrix 68
7.2 Accuracy on the test data of the CIFAR-10 dataset using different filter sizes and different number of filters per layer. 69
7.3 Loss on the test data of the CIFAR-10 dataset using different filter sizes and different number of filters per layer. 69
7.4 Accuracy on the training data of the CIFAR-10 dataset using different filter sizes and different number of filters per layer. 70
7.5 Accuracy on the test data of the CIFAR-10 dataset using different filter sizes and different number of filters per layer beginning from the 10-th epoch. 70