

BACHELOR'S THESIS
ARTIFICIAL INTELLIGENCE

Radboud University



**Self-Attention in Convolutional
Neural Networks: a Potentially
Fundamental Improvement for
Image Classification**

Author:
Pelle Kools
S1010033

First supervisor:
dr. R.S. van Bergen¹
ruben.vanbergen@ru.nl

Second supervisor:
dr. T.C. Kietzmann¹
t.kietzmann@donders.ru.nl

¹Donders Institute for Brain,
Cognition and Behaviour

June 18, 2021



Faculty of Social Sciences
Radboud University
The Netherlands

Abstract

Conventional (feedforward) Deep Neural Networks (DNNs) have some inherent limitations and weaknesses that humans do not have. In the domain of computer vision, this becomes clear when a CNN is confronted with adversarial attacks. An explanation for this difference in performance and robustness between a CNN and the human brain could be the usage of recurrence. The visual cortex of the human brain amply uses feedback connections when processing visual input. Most conventional CNNs, however, do not make use of this type of connections. Therefore, we believe that recurrence might help improve performance of CNNs, at least in certain challenging tasks. In this paper, we propose a kind of recurrence that is based on top-down attention. For our experiments, we used a feedforward CNN as a benchmark network and compared it to an equivalent network but with top-down attention. We compared the performance of both models in a digit classification task that was complicated by partial occlusion. In this task, the recurrent network significantly outperformed the benchmark network. Although further research is needed to make definitive conclusions, the results provide promising initial evidence that the use of recurrence in a CNN can lead to fundamental improvements in image classification.

Contents

1	Introduction	3
1.0.1	Top-down attention	4
1.0.2	Recurrence	4
1.0.3	Research question	4
1.0.4	Related Work	5
1.0.5	Overview	6
2	Methods	7
2.1	The Data and the Task	7
2.1.1	Importance of choosing the right data	7
2.1.2	Requirements	7
2.1.3	The dataset	8
2.1.4	The task	8
2.2	The models	9
2.2.1	Benchmark model: LeNet-5	9
2.2.2	The altered model: Attentional LeNet	10
2.3	Experiments	11
2.3.1	Model settings	11
2.3.2	Optimizing the training process	11
3	Results	14
3.0.1	Model comparison	14
4	Discussion	17
4.1	Interpreting results	17
4.2	Future research	17
4.2.1	Complexity	17
4.2.2	Position of attentional layer	17
4.2.3	Number of attention iterations	18
4.2.4	Amount of attentional layers	18
4.2.5	Experimenting with different data	18
4.2.6	Conclusion	19

A Appendix	21
A.1 Attentional Convolutional Layer	21

Chapter 1

Introduction

Computer vision systems are often not as robust as we want them to be. Often they rely upon CNNs that are trained on a certain dataset and seem to perform well on a corresponding test set. However, it can be surprisingly easy to deceive such a CNN with an adversarial attack. In such an attack, an input is modified intentionally to deceive the network, i.e. to let it make a false prediction.



Figure 1.1: Stop sign modified with black and white stickers. Used for adversarial attacks by researchers from Washington University [1].

See Figure 1.1, for example. This image shows a stop sign that is slightly modified by putting black and white stickers on top of it. Researchers from Washington University [1] showed this modified sign to a computer vision system that had been trained on traffic signs. They found that these small modifications let the system think it was seeing a 45mph speed limit sign instead of a stop sign. A human, on the other hand, would effortlessly recognize this as a stop sign, despite the modifications. One reason that computer vision systems are vulnerable in such situations may be that their way of processing visual information differs fundamentally from that of the

human brain. In the visual cortex, the human brain not only makes use of feedforward (bottom-up) connections, like most traditional neural networks do, but also of feedback (top-down) connections [2, 3, 4]. Therefore, the use of feedback connections, and thereby recurrence, might also be beneficial to the performance of a CNN. One way to implement such recurrence in neural networks is to make use of top-down attention.

1.0.1 Top-down attention

Top-down attention is the basis of recurrence in our network. A convolutional layer with top-down attention still makes use of kernels, in the same way a typical convolutional layer would. However, on top of this, another filter is applied, containing the attention weights. The attention filter is as large as the size of the resulting feature map multiplied by the kernel size. For every window of the input image on which the kernel is applied, the attention filter represents how strongly that window correlates with all other windows in the image. As a result, the spatial relationship between different parts of the image, outside of the kernel scope, can be represented. This is the basis of the routing mechanism of the layer. Consider, for instance, the digit 8. An 8 mainly consists of two loops on top of each other. A loop can therefore be seen as a feature that increases the probability of the digit 8 being present. However, if such a loop is present in the top half of the image, while it is absent in the lower half, it is likely that no 8 is present in the image. This spatial relationship between different parts of the image can not be represented in a traditional CNN. The attention weights make this possible.

1.0.2 Recurrence

In the previous paragraph the mechanisms of attention and routing in a convolutional layer are explained. However, these mechanisms do not automatically make use of feedback connections, which is something we do want to incorporate. In order for this kind of feedback to be present, a kind of reverse convolution had to be performed. The top-down weights associated with this computation should predict the original image from the computed feature map. The difference between the inferred image and the original image is used to adjust the attention weights. After training the network, the bottom-up weights (the normal kernels) and the top-down weights are fixed. However, the layer is still run for a fixed number of iterations, allowing the attention weights to adjust to the input.

1.0.3 Research question

Because of the possibility to represent spatial relationships across an entire input image, a network that implements this kind of behaviour might have

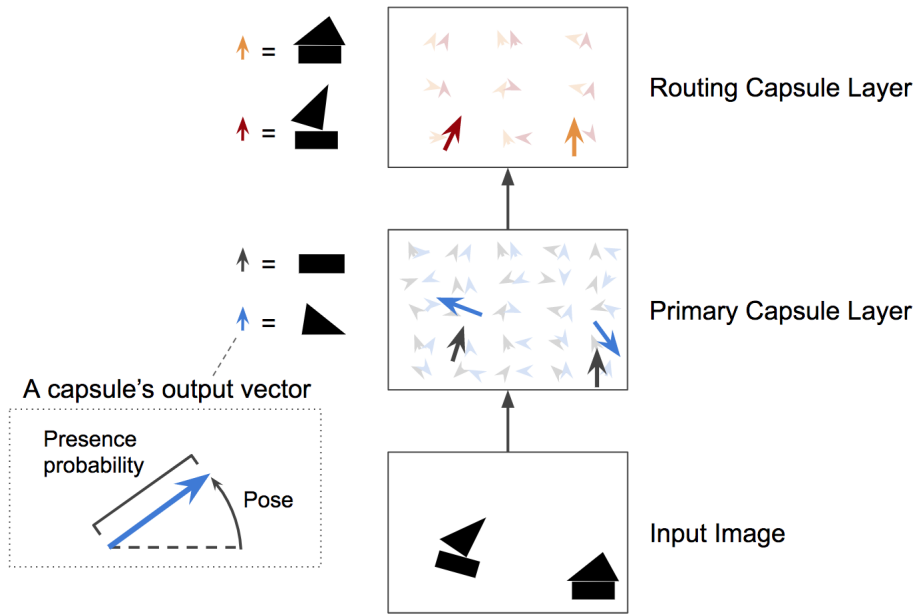


Figure 1.2: The input image contains two objects: a boat and a house. Both can be represented by a combination of features such as a triangle and a rectangle. The first layer in the network contains capsules that represent rectangles and triangles of different orientations and sizes, and how likely they are to be present in the input. The second layer contains higher level capsules for houses and boats. According to routing by agreement, capsules representing the house and boat will only be activated if there is a combination of lower level capsules that *together* are likely to form a house or boat.

more capacity to capture the underlying properties that comprise an object. Accordingly, the research question is: “Can a Convolutional Neural Network (CNN) with top-down attention outperform an equivalent feedforward CNN in image classification?”. The goal of this thesis is not to implement a new state of the art architecture in the field of image classification. Rather, the goal is to prove that if a CNN is augmented with top-down attention, the performance increases significantly. If this thesis can prove that, it would be a great starting point for overcoming some of the fundamental limitations of conventional CNNs.

1.0.4 Related Work

The kind of top-down attention and resulting recurrence we propose are closely related to and partially inspired by Capsule Neural Networks (CapsNets), which were introduced by Geoffrey Hinton et al. in 2017 [5]. Hinton proposed a this new kind of architecture, that, as the name suggests, was built upon capsules. A CapsNet can contain multiple capsules, and each

capsule is a small neural network in itself. Each capsule represents the probability of a specific feature being present in the input. The output of all capsules is then combined, and fed forward to a new group of capsules, this time representing higher level features that are built up on (some of) the lower level capsules in the previous layer. This is where the network makes use of a new kind of routing mechanism; routing by agreement. A traditional CNN would base the presence of an object on the presence of the features it is built from, without taking into account what the spatial relationship between those features is. In a CapsNet, the *agreement* of capsule activations must be high too in order for the higher level feature or object to be activated in a capsule. Figure 1.2, which was made by Aurélien Géron [6], illustrates the mechanism of capsules with a visual example.

1.0.5 Overview

In Chapter 2, the methods and experiments we used to answer our research question will be explained. Thereafter, in Chapter 3, the results will be reported. Finally, in Chapter 4, the results will be interpreted and possibilities for future research will be discussed.

Chapter 2

Methods

In this chapter the research methods and results will be shown and explained. First, the dataset and its properties will be explained. Then, a detailed explanation will follow on the task that was used for the experiments and why that task is relevant in this scenario. Thereafter, the two models that were used for the experiments will be described in detail. Finally, the experiments and their results will be reported.

2.1 The Data and the Task

2.1.1 Importance of choosing the right data

To find out whether attention really improves image classification performance, a dataset and a corresponding task must be used that is particularly challenging for current CNNs to perform. However, this thesis only strives to create a proof of concept, as attention and its usages in image classification are still somewhat unknown. The task must therefore not be overly complicated yet. Furthermore, feedforward CNNs, despite their limitations, still perform remarkably well on simple datasets like MNIST, so adding attention in such a scenario would be an unnecessary complication.

2.1.2 Requirements

One element of computer vision that often proves to be difficult for image classification is the concept of *occlusion*. Occlusion is when in the input image one object (partially) covers another object. Humans can easily segment the input into different parts, and in a way fill in the missing parts of an object. For a neural network, however, it can be hard to tell whether the input consists of two or more separate objects that are in front of each other, or just one object. Intuitively, because our implementation of top-down attention more closely resembles the visual cortex of the human brain,

its addition to a neural network should improve performance in such situations. For the aforementioned reasons, a task involving occlusion of targets could really distinguish a network with attention from a network without attention.

2.1.3 The dataset

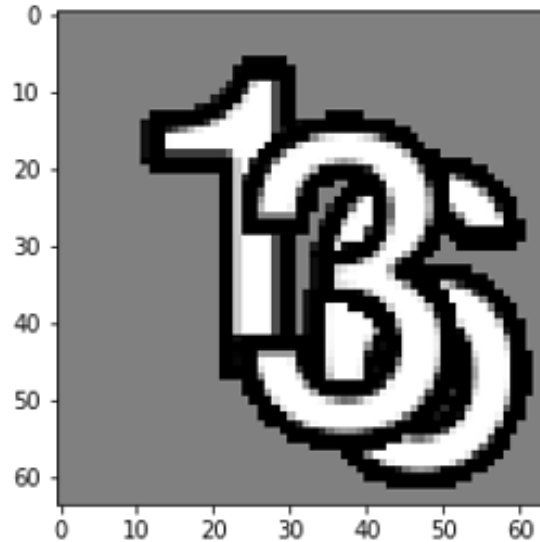


Figure 2.1: A typical sample from the dataset

The dataset that was used for the experiments consists of 86,000 grayscale images, each image containing three distinct digits. For every image, the label is simply a list with a 1 at the index of a present digit and similarly a 0 for every absent digit. A typical input image can be seen in Figure 2.1. The digits themselves are randomly positioned across the image, with the limitation that a digit must always be fully contained in the image frame, to not make the task too difficult. Furthermore, the depth-order of the digits is also determined randomly per image. This dataset is relatively simple, because the input is unambiguous and easy to interpret. However, although the images are simple, they are not easy to process for a network, because the digits may occlude each other. The dataset was generated using the code of

2.1.4 The task

Given the generated data, the task of the models was to correctly classify all three digits in a given image. When the input is as in Figure 2.1, the model should return that a 1, a 3 and a 6 are present, while no other digits are there.

If attention actually improves a network’s capacity to capture the properties of the targets, it should become clear when processing images like these. Hence, performing this task on the generated dataset is particularly well-suited to assess the influence of attention on image classification. Finally, in order to assess the performance of a model in this task, we had to choose an appropriate loss function. The Binary Cross-Entropy Loss (BCELoss) is certainly a fitting loss function. For every possible class, the BCELoss calculates the probability of it being present or not. The equation for the BCELoss is as follows:

$$BCELoss(\mathbf{y}) = \frac{1}{N} \sum_{i=0}^N y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$

In this equation, N stands for the amount of classes. For our dataset, $N = 10$, because there are 10 digits. \mathbf{y} is the target vector of one sample. In other words, it is an array of size N , telling which classes are present in the input (1 if a class is present, 0 if not). y_i stands for the i th element of \mathbf{y} , i.e. the presence of a single digit i . $p(y_i)$ represents the *probability* of y_i . So $p(y_3)$ with $y_3 = 0$ represents the probability that 3 is **not** present in the image.

2.2 The models

In order to make a good comparison, a suitable benchmark model had to be chosen first. Then, this benchmark model had to be modified to make use of top-down attention.

2.2.1 Benchmark model: LeNet-5

motivation

The benchmark model that was chosen for this experiment was LeNet-5. There are two main reasons why this is a suitable benchmark model:

1. First of all, LeNet-5 is a feedforward model that makes use of standard convolutional layers. Therefore, it is the perfect example of a ‘traditional’ CNN that is limited because of its lack of recurrence and attention.
2. This thesis is aimed at making a proof of concept, rather than creating a new state of the art image classification model. Therefore, there is no need to make the model overly complicated. LeNet-5 is a relatively simple model, that is easy to train, which is useful when running experiments.

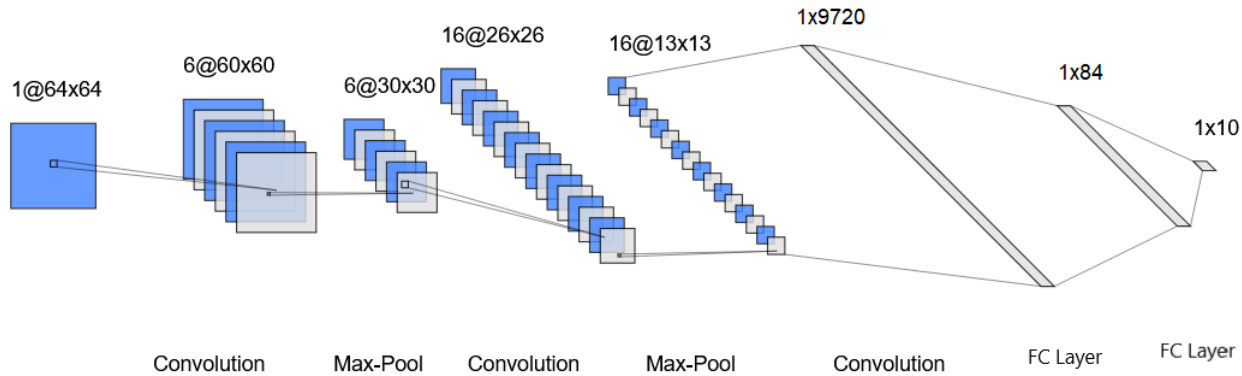


Figure 2.2: LeNet-5 architecture

The architecture

The architecture of the LeNet-5 model is as shown in Figure 2.2. Note that the average-pooling that is normally used in Lenet-5 has been replaced by max-pooling. Furthermore, instead of the TanH activation function, a ReLU activation has been used. In the end, the network gives ten outputs, one for each digit. The logits are normalized by a sigmoid function to ensure that a probability is returned per digit. In other words, for every digit, the probability that the digit is in the input image according to the network is returned.

2.2.2 The altered model: Attentional LeNet

The altered model is exactly the same as the LeNet-5 model, except for the first layer of the network. Instead of a regular convolutional layer, an attentional convolutional layer is used (see appendix A for the code). This layer was developed by dr. R.S. van Bergen.

The attentional convolutional layer

The attentional layer, next to doing a normal convolution, applies attention to the input it receives. Using the bottom-up weights and the attention weights, a feature map of the input is computed. This feature map, using the top-down weights, is then used to make a prediction of what the input was that resulted in that particular feature map. The difference between that prediction and the actual input is then used to adjust the weights. This process of computing a feature map and recurrently adjusting the weights is repeated for a limited number of iterations (default is 4). After the recurrent application of attention, the attentional layer gives its output to the next

layer. The rest of the processing is exactly the same as in the benchmark network.

2.3 Experiments

2.3.1 Model settings

Splitting the data

The dataset, consisting of 86,000 images and corresponding labels, was split up into train, validation and test sets with a ratio of 8:1:1. In other words, the train set consists of $0.8 * 86,000 = 68,800$ images, while the validation and test set both consist of $0.1 * 86,000 = 8,600$ images.

Hyperparameters and optimization

For the first experiment, the following hyperparameters were used:

- **number of epochs = 20**: this number was chosen because the networks both seemed to get close to convergence with this number of epochs.
- **batch size = 32**: this is a batch size that contains a reasonable variety of images in one batch, while staying within the memory limits.
- **learning rate = 3e-4**

The criterion used to evaluate the performance of both models is the BCELoss, as explained before. Furthermore, the optimizer that was used to train the networks was Adam. The criterion and the optimizer were kept the same across all experiments.

2.3.2 Optimizing the training process

In the first experiment, the benchmark LeNet-5 clearly outperformed the altered model. Not only did the model reach a lower test loss (0.1167 compared to 0.1837), it did so in a smaller number of epochs, as well as taking less time per epoch. The benchmark model outperformed the model with attention in every aspect.

Adding batch normalization

According to [7], *"BatchNorm impacts network training in a fundamental way: it makes the landscape of the corresponding optimization problem be significantly more smooth. This ensures, in particular, that the gradients are more predictive and thus allow for use of larger range of learning rates and faster network convergence."* Batch normalization could be a great addition

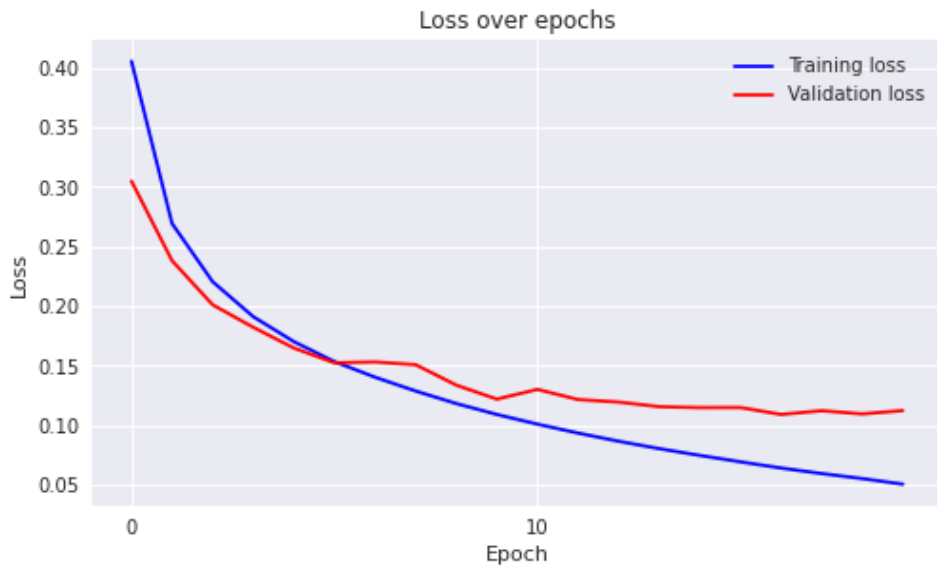


Figure 2.3: BCELoss over epochs for LeNet-5, first experiment

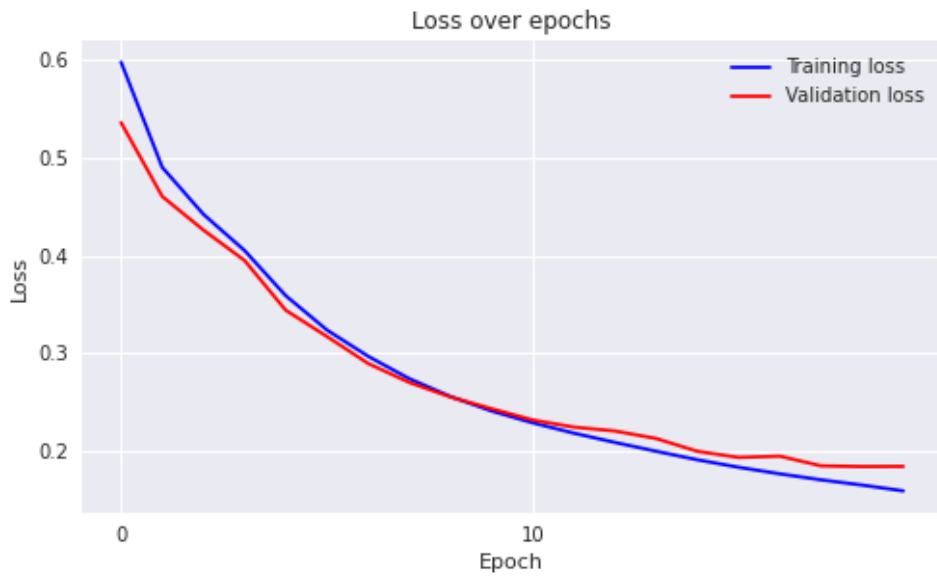


Figure 2.4: BCELoss over epochs for Attentional LeNet, first experiment

to improve performance, especially for the altered model, because it is more complex. Therefore, batch normalization was added to both models (any modification should be applied to both models, to keep the comparison fair). Batch normalization indeed vastly improved performance, especially for the model with attention. Both models trained faster in terms of epochs, but the LeNet-5 model did not achieve higher performance in terms of loss, while the attentional model did.

Removing ReLU from the attention layer

While looking for more ways to optimize the training process, it was discovered that the Attentional LeNet uses ReLU inside of its Attention layer as well as after this layer. This made one of the ReLUs redundant, making it a logical step to remove one of the two. To make the Attentional LeNet most similar to the benchmark model, the ReLU was placed outside of the attention layer. This decision did not change the performance of the network significantly.

Storing best model

Now that the training process seemed to be optimized, the most noteworthy comparisons could be done. During training, the training loss as well as the validation loss were measured. In order to make sure the best version of the model was stored instead of a possibly over-fitting model, the model with the best validation loss was stored for evaluation.

Chapter 3

Results

3.0.1 Model comparison

To ensure that both models converged, they were each run for 30 epochs. From previous results, 30 epochs appeared to be an amount of epochs in which the optimal validation loss was always reached. After training, both models were evaluated on the test set. The benchmark model achieved a test loss of approximately 0.1130. The Attentional LeNet, though, achieved a lower loss of approximately 0.1029. To check whether this difference is significant, statistical tests have to be performed.

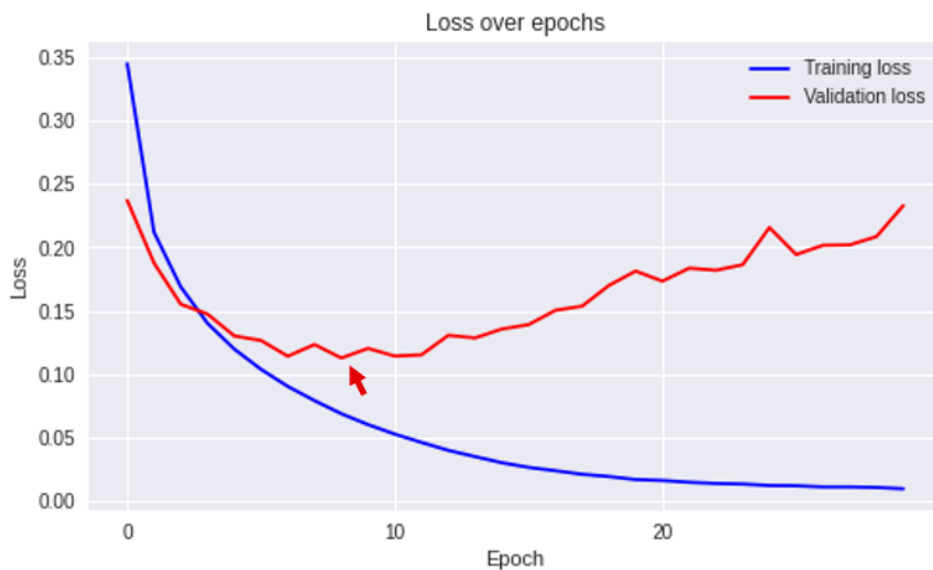


Figure 3.1: BCELoss over epochs for LeNet-5, final comparison

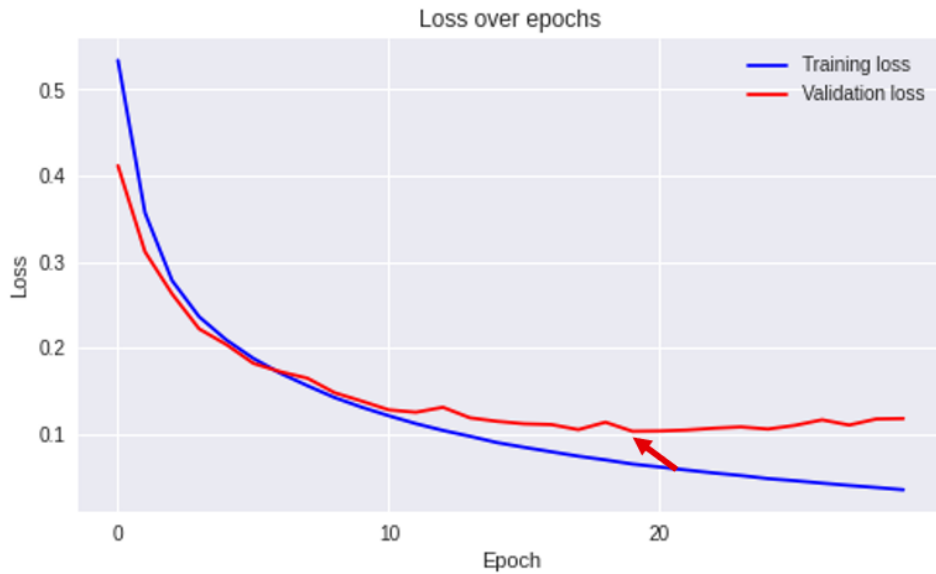


Figure 3.2: BCELoss over epochs for Attentional LeNet, final comparison

Statistical analysis

The first statistical test that comes to mind when comparing to populations that describe the results of the same independent variables (the input) in different circumstances (model parameters) is the paired t-test. However, when doing a paired t-test, it is assumed that the population differences are normally distributed. A Q-Q plot is a robust way to check for normality.

As can be seen in Figure 3.3, the differences in loss distributions are not normally distributed. Rather, the distribution of loss differences is right-skewed. Therefore, a paired t-test is not suited for this situation. Instead a more robust analysis should be used, that does not assume normality. One such analysis is the Wilcoxon signed-rank test. This test compares the differences between the results of all paired samples. The absolute values of those differences are taken and ranked in ascending order. Finally, the ranks corresponding to an originally negative difference are added up, while the same is done for the positive differences. The smallest of these numbers indicates the test statistic. Furthermore, this analysis returns a p-value, indicating whether the found difference is actually significant. In this experiment, the results were as follows: the test statistic had a value of 16434458 with a p-value of approximately $4e-19$. This means that the difference in performance is indeed significant, meaning that the model with attention performed significantly better than the benchmark model on the image classification task.

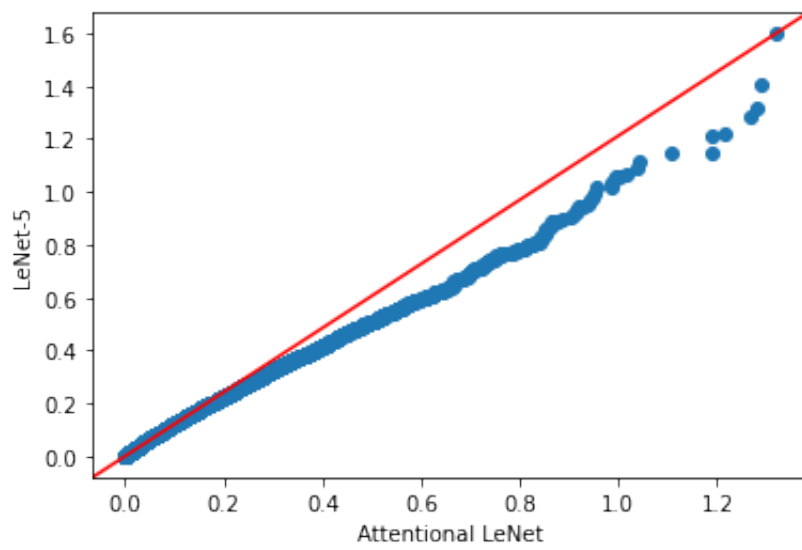


Figure 3.3: QQplot, checking for normality in the differences between the loss distributions

Chapter 4

Discussion

4.1 Interpreting results

As can be seen in the results section of the previous chapter, the model with attention significantly outperformed the benchmark model. This means that the addition of top-down attention and thereby recurrence have a positive effect on the performance of CNNs in image classification. Therefore, . In this section, the possible topics for future research and their relevance will be discussed.

4.2 Future research

4.2.1 Complexity

The influence of the increase in complexity is one of the first topics that should be investigated in further research. This can for example be done by increasing the number of parameters in the benchmark model, until the its complexity is comparable to that of the recurrent model. Then, the same experiments should be run to see if there is still a significant difference in performance.

4.2.2 Position of attentional layer

Another factor that should be considered is that the attentional layer has so far only replaced the first layer of the benchmark model. The attentional layer probably has different effects on the output depending on its position in the network. Changing the position of the attentional layer could therefore affect performance too and might provide other interesting insights.

4.2.3 Number of attention iterations

Not only the relation between the attentional layer and the entire network is of importance. Increasing the number of iterations in such an attentional layer might give the layer more opportunity to really learn the underlying patterns of the input. Hence, experimenting with the number of iterations might give even more insight into how much influence the attentional part of the network really has.

4.2.4 Amount of attentional layers

Apart from adjusting the amount of iterations and the position of a recurrent layer in the network, it could be an option to use multiple recurrent layers. This might improve performance even more, because the human brain also makes use of feedback connections between multiple visual areas [2, 3, 4], not just at one point in the process. However, because this would increase the complexity even more, this is an experiment that should be done after other experiments have been performed and it still seems to be worth pursuing.

4.2.5 Experimenting with different data

Finally, the data itself are also crucial for the way a network trains, and thus for its performance. There are many ways in which the data could be altered to both possibly improve the training process, as well as give a deeper understanding of the way the networks learn:

1. **Digit size:** when the size of the digits is always the same, like in the dataset we used, there is a possibility that the network does not perform well on digits of different sizes. To check this, it would be useful to test the model on digits that are of a different size than the digits in the training data. If the model appears to not perform well on different digit sizes, that would mean that the model learned that size is an inherent property of a digit, even though it is not. A possible solution would be to train the model on images containing digits of various sizes.
2. **Digit orientation:** similarly, the model is trained on digits that are all in the same orientation, it might learn that that specific orientation (upright, in our dataset), is inherent to the digit it is labelling. Again, it would be useful to first test the model on digits with different orientations, to check whether this is the case. And if it indeed appears to be so that the model has learned that orientation is an inherent property, digits of different orientations should be included in the training set to see if it will improve model performance.

3. **Withholding certain combinations:** currently, no combinations of digits were withheld in the training data. The result of this could be that the model has learned all (or many) combinations of digits overlaying each other, thus being able to correctly classify every combination it comes across. In that case, the model has only learned the specific visual pattern that corresponds to a combination of digits. What we want the model to learn, however, is what the properties are that comprise a digit. This would make the model much more robust, and less vulnerable to the weaknesses and limitations that conventional CNNs come across. Hence, training the model on images in which certain combinations of digits do not occur would be useful for checking whether the model is as robust as we want it to be.
4. **Digit color:** the dataset contains only grayscale images, the digits always being white. This might cause the model to learn that the colour white is an inherent property of a digit, even though it is not. A 9 is a 9, whether it is green, red, or any other colour. If the model has identified colour as an inherent property, it has not learned the right properties of a digit and it is not as robust as we want it to be. In order to check whether the model has indeed learned to classify digits independent of their colour, we could give each digit a random colour. Furthermore, comparable to the withholding combinations of digits, we could withhold combinations of colours. This way we can check with the test data whether the model indeed does not depend on colour when classifying digits.

4.2.6 Conclusion

In short, our model that was augmented with top-down attention significantly outperformed the benchmark model, giving promising initial evidence that top-down attention can be a long term fundamental improvement for CNNs in image classification. It is definitely worth it to pursue further research in this area, advancing on the findings from this paper.

Bibliography

- [1] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, “Robust physical-world attacks on deep learning models,” 2018.
- [2] A. Angelucci and P. C. Bressloff, “Contribution of feedforward, lateral and feedback connections to the classical receptive field center and extra-classical receptive field surround of primate v1 neurons,” in *Visual Perception* (S. Martinez-Conde, S. Macknik, L. Martinez, J.-M. Alonso, and P. Tse, eds.), vol. 154 of *Progress in Brain Research*, pp. 93–120, Elsevier, 2006.
- [3] G. Kreiman and T. Serre, “Beyond the feedforward sweep: feedback computations in the visual cortex,” *Annals of the New York Academy of Sciences*, vol. 1464, no. 1, pp. 222–241, 2020.
- [4] V. A. Lamme and P. R. Roelfsema, “The distinct modes of vision offered by feedforward and recurrent processing,” *Trends in Neurosciences*, vol. 23, no. 11, pp. 571–579, 2000.
- [5] S. Sabour, N. Frosst, and G. E. Hinton, “Dynamic routing between capsules,” *arXiv preprint arXiv:1710.09829*, 2017.
- [6] A. Géron, “Introducing capsule networks,” Feb 2018.
- [7] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, “How does batch normalization help optimization?,” 2019.

Appendix A

Appendix

A.1 Attentional Convolutional Layer

```
class AttConv(nn.Module):
def __init__(self, num_in_chan=1, num_out_chan=6, kernel_size=5, stride=1):
    super(AttConv, self).__init__()
    self.kernel_size=kernel_size
    self.num_in_chan=num_in_chan
    self.num_out_chan=num_out_chan
    self.stride=stride
    self.BU_weights = nn.Parameter(torch.Tensor(1,num_in_chan*kernel_size**2,
                                                1, num_out_chan))
    init.kaiming_uniform_(self.BU_weights, a=np.sqrt(5))
    self.TD_weights = nn.Parameter(self.BU_weights.data.detach().clone())

    self.BU_bias = nn.Parameter(torch.randn(1,1,1,num_out_chan)*0.1)
    self.TD_bias = nn.Parameter(torch.randn(1,num_in_chan,1,1)*0.1)

def normalize_att_weights(self, in_spat_dim):
    batch_size = self.att_weights.shape[0]
    num_wins = self.att_weights.shape[-1]
    aw_sum = F.unfold(F.fold(self.att_weights, in_spat_dim, self.kernel_size),
                    self.att_weights.shape[-1], self.kernel_size)
    self.att_weights = self.att_weights/aw_sum
    self.att_weights = self.att_weights.view(batch_size, 1, self.kernel_size**2)

def forward(self, x, num_iter=4):
    batch_size = x.shape[0]
    device = x.device
    in_spat_dim = list(x.shape[-2:])
    assert in_spat_dim[0]==in_spat_dim[1], 'Only square images are supported'
```

```

x_wins = F.unfold(x.view(batch_size,self.num_in_chan,*in_spat_dim), self.kernel_size, self.stride)
out_spat_dim = np.int(np.sqrt(x_wins.shape[-1]))
self.att_weights = torch.ones([batch_size, self.kernel_size**2, x_wins.shape[-1]])
self.normalize_att_weights(in_spat_dim)

for i in range(num_iter):
    y = (x_wins.unsqueeze(-1)*self.att_weights.unsqueeze(-1)*self.BU_weights).sum(-1)
    pred = (y*self.TD_weights).sum(-1).view(batch_size,self.num_in_chan,self.num_out_chan)
    self.att_weights = ((pred*x_wins.view(batch_size,self.num_in_chan,self.num_out_chan)).sum(-1)).unsqueeze(-1)
    self.normalize_att_weights(in_spat_dim)

y = y.view(batch_size, out_spat_dim, out_spat_dim, self.num_out_chan).permute(0,3,1,2)

return y

```