UNIVERSITÄT OSNABRÜCK

INSTITUTE OF COGNITIVE SCIENCE
COMPUTER VISION

Bachelor's Thesis

ADVERSARIAL EXAMPLES

ANALYZING ATTACKS AND DEFENSES

Laura Goerke

February 4, 2019

First supervisor: Dr. Ulf Krumnack
Second supervisor: Prof. Dr. Gunther Heidemann

Adversarial Examples
Analyzing Attacks and Defenses

During the quick rise of neural networks a phenomenon called *Adversarial Examples* became known. Adversarial examples are images derived from a dataset which fool a neural network into making incorrect classifications. Now, attacks using adversarial examples and defenses trying to protect neural networks against them are being developed. This bachelor thesis will analyze such attacks and defenses.

Contents

Chapter 1

Introduction

The German postal office processes ~20.000 million letters every year (*Deutsche Post - Beförderte Briefe | 2017*, 2018). This would not be possible with human resources alone. Machine learning and computer vision are used to read the postal address and automatically sort them accordingly (*Briefverteilanlage - Wikipedia*, 2019). This is but one example how machine learning pervades our everyday lives, it plays a similarly important role in various industrial settings such as healthcare, retail industry, and financial sector (*7 Industries Leveraging Machine Learning*, 2019). However, there are also negative aspects to this automization. Knowing how computer vision systems are comprised and where their weaknesses lie, it is possible to abuse them systematically. Malicious parties might be interested in fooling computer vision systems into making wrong predictions. One way of compromising these targets that gained particular attention are so called adversarial examples (Szegedy et al., 2013). They purposefully lead the algorithm to make wrong decisions. A small perturbation is added to the original image, so small that the resulting image will still seem natural to the human eye. However, this perturbation is enough to push the algorithm to make an incorrect classification. If the perturbation was bigger, a human might detect that there was something wrong with the input and might mistrust the neural network. But with perturbations so small they are unperceivable for a human, there is no reason to not believe the wrong predictions of the neural network. This makes adversarial examples particularly dangerous.

Relevance Aside from the postal service, there are more cases for which adversarial examples could lead to security issues. Another example is facial recognition. This is a broadly used machine learning application, especially in security critical fields. Sharif et al. exploited the possibilities of adversarial glasses (Sharif, Bhagavatula, Bauer, & Reiter, 2016). Printed frames were used

to either evade recognition or to impersonate somebody else. Such glasses would make it possible for burglars to impersonate a person with access to protected areas and therefore gain access as well. Another important issue are self-driving cars. Already today, many cars have assistance features relying on machine learning such as traffic sign recognition (Moutarde, Bargeton, Herbin, & Chanussot, 2009) and park assistance (*Audi uses machine learning to refine self-parking technology*, 2019). When switching to fully autonomous cars the security issues become even more significant. Sitawarin, Bhagoji, Mosenia, Chiang, and Mittal (2018) used adversarial examples to fool a traffic sign detection. They used signs looking like advertisement to deceive the machine learning algorithm to make wrong classifications. Using such signs, one could purposefully prevent autonomous cars from stopping before intersection or make them accelerate on small and dangerous streets, which could both lead to fatal accidents. Because of the severe security concerns defenses against adversarial examples are already being developed. They aim at making machine learning systems more robust against such attacks (Goodfellow, 2018). In the light of these studies and their relevance for applications in which security is imperative, it is important to research the mechanisms underlying adversarial examples and potential protections against them.

Thesis Outline There is a multitude of different adversarial attacks and defenses for neural networks (Yuan, He, Zhu, Bhat, & Li, 2017). In order to gain a better understanding of the interplay of adversarial attacks and defenses, we investigated one attack named *Fast Gradient Sign Method (FGSM)* (Goodfellow, Shlens, & Szegedy, 2014) and one building upon FGSM named *Momentum Iterative Fast Gradient Sign Method (MI-FGSM)* (Dong, Liao, Pang, Hu, & Zhu, 2017). Both were used on an unprotected network and on a network defended with *Defensive Distillation* (Papernot, McDaniel, Wu, Jha, & Swami, 2015). Firstly, adversarial examples will be explained including their history of research. Secondly, we will describe the defense and attacks which were employed. Then, implementation details are described as well as the datasets that were used. Lastly, the results are presented and discussed.

Chapter 2

Background

2.1 Neural Networks

Artificial neural networks are inspired loosely by the neuronal networks in the human brain (Jain, Mao, & Mohiuddin, 1996). To transfer electrical signals between the synapses of the brain, a certain threshold has to be reached. By overshooting the threshold an action potential will be created and the signal will be transmitted.

Neural networks use a similar approach. An input (signal) is passed into the neural network. When it reaches a neuron, an activation function determines whether the signal will be transmitted. In order to make the neural network process more information, complex models comprising a multitude of interwoven neurons are subject to intensive research. The input of the neural network is passed to several units simultaneously which in turn transmit their output to multiple neurons along a cascade of activations. Neurons which receive the input simultaneously are grouped into a so-called layer. A neural network can consist of an input layer, several hidden layers, and an output layer. Units across layers are connected by transmission weights which determine how strongly a neuron is affected by its input. This mirrors the different strengths of synaptic connections within the brain. Some synaptic connections pass higher electrical signals than others and are therefore more important for passing the threshold. Moreover, different activation functions can be used. The step function is the activation function which is equivalent to the behaviour in the human brain. It is 0 for all values below a certain threshold and 1 for all values after. This mimics the synapses in the brain, which only transmit a signal after the threshold is reached. But, this activation function is limited because of its discrete values. The inbetween values if a continuous function, however, can propagate more granular information to the rest of the network. One intuitive choice

would be the identity function. However, as we will later see in Figure 2.1, the derivative of the activation dependent on the input is needed for training a neural network. The derivative of the identity function is always a constant and is therefore not depended on its input. Thus it is unusable for the training algorithm of neural networks. Hence, we need a non-linear activation function.

One example of a non-linear activation function is the sigmoid function.

$$A(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

An advantage of this activation function is that it is bounded. The function can only take on values between 0 and 1. This way the activation cannot "explode" resulting in ever increasing values. Another positive aspect is its step-like behaviour. Even though it is a continuous function, for most values the sigmoid function returns values close to either 0 or 1.

Another smooth activation function is the softmax function.

$$A(x_i) = \frac{exp(x_i)}{\sum_j exp(x_j)} \tag{2.2}$$

It is similar to the sigmoid function. However, the resulting activations for each neuron i add up to 1. So when using softmax in the last layer, the produced activation is a probability distribution over all neurons i.e. classification targets.

An activation function broadly used in modern neural networks is ReLU or a modification of it.

$$A(x) = max(0, x) \tag{2.3}$$

It is a special activation function because for all values above 0 it is linear while for all other values it is 0. For the values below 0 the derivation is also 0, which can — as mentioned above — become an issue when using the backpropagation algorithm.

Neural networks using non-linear activation functions can approximate any continuous real-valued function (Hornik, Stinchcombe, & White, 1989). In particular, they can be used for classification. In order to use a neural network as a classifier it has to be trained. The most relevant way of adjusting the network's weights in order to learn a desired function is the so called backpropagation algorithm (*cf. Figure 2.1)*. It is a form of gradient descent which aims at calculating the gradient in order to minimize the loss.

After the network weights are initialized with random numbers, the algorithm consists of a loop of alternating forward steps and backward steps.

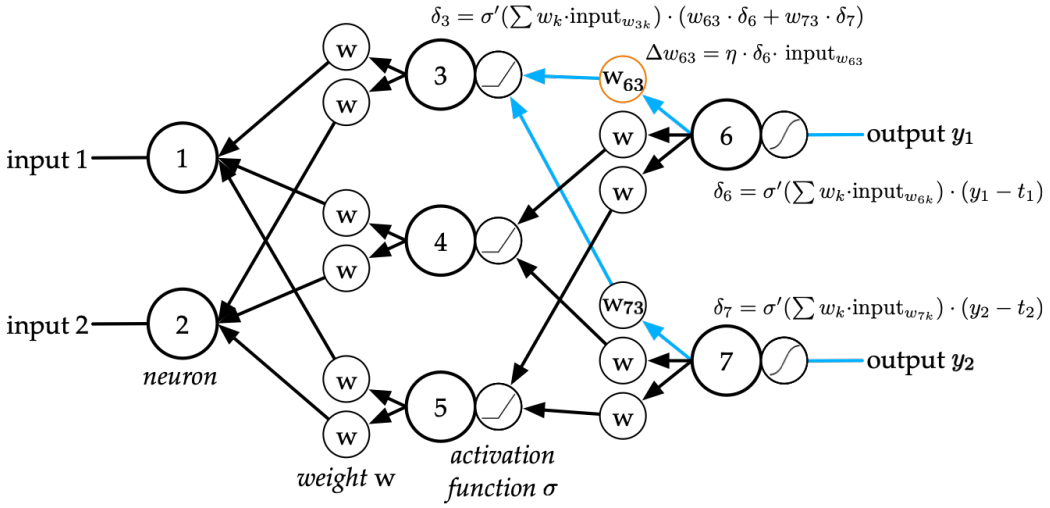


Figure 2.1: Visualization of the backpropagation algorithm

In the forward step, the input to be classified is fed into the neural network. During this process, the predictions of the last layer y and the derivative of the activation function of each neuron depending on the current input $\sigma'(\sum w_k \cdot \text{input}_k)$ are saved. In the backward step, the weights of the network are adjusted starting from the last layer through the hidden layers. For this, an error signal δ_n is calculated for each neuron n, which is used to adjust its weights.

In the output layer, the error δ_n is calculated using the the difference between the prediction y of the network and the actual target label t multiplied with the saved derivative of the activation function $\sigma'(\sum w_k \cdot \text{input}_k)$.

$$\delta_n = \sigma'(\sum w_k \cdot \text{input}_k) \cdot (y_n - t_n) \tag{2.4}$$

In the hidden layer, since no target label is available, the error has to be calculated in a different way. Here, the error δ_n is calculated using the sum of the errors of the neurons of the next layer multiplied with the weight connecting neuron n with them. This sum $\sum_{m \in \text{next layer}} w_{mn} \cdot \delta_m$ is multiplied with the derivative of the activation function $\sigma'(\sum w_k \cdot \text{input}_k)$.

$$\delta_n = \sigma'(\sum w_k \cdot \text{input}_k) \cdot (\sum_{m \in \text{next layer}} w_{mn} \cdot \delta_m) \tag{2.5}$$

The update rule for the weights is the same for output and hidden layers. To calculate the weight update $\triangle w_{nm}$ the learning rate η of the network, the

error δ_n of the neuron which is modified by weight w_{nm} and the input to the weight are multiplied.

$$\triangle w_{nm} = \eta \cdot \delta_n \cdot \mathrm{input}_{nm} \tag{2.6}$$

The weight update is then added to the original weight to get the new weight. This is done iteratively until a local optimum is found or a maximum number of iterations is reached.

2.2 Adversarial Examples

Adversarial examples are a phenomenom of machine learning classification problems (Szegedy et al., 2013). They gained notoriety with the quick rise of neural networks but are not limited to this type of machine learning system. Linear classifiers, decision trees, and support vector machines are also susceptible to adversarial examples (Yuan et al., 2017). Furthermore, adversarial examples are not limited to computer vision problems, but can be found in speech recognition (Jia & Liang, 2017), maleware detection (Grosse, Papernot, Manoharan, Backes, & McDaniel, 2017), and other fields. However, the computer vision field is the most investigated one, therefore we decided to focus this thesis on adversarial examples for image classification problems.

Formally, an adversarial example x^* for an image x and a classifier f can be defined by

$$x^* = x + \delta \approx x \qquad \text{such that } f(x) \neq f(x^*). \tag{2.7}$$

This means that an adversarial example is the sum of the original image and a perturbation just small enough so that the two images seem to be the same, but also big enough for the classifier to misclassify the adversarial example.

They generalize between different neural network models as well as different machine learning techniques (Yuan et al., 2017). This is an important feature when attacking a classifier without exact knowledge of the underlying algorithm. In contrast to other algorithms that often require intricate knowledge of the algorithm that is attacked, adverserial examples can be created for a substitute network/classifier that was trained on the same dataset as the attacked algorithm (Papernot et al., 2016). Due to their good transferability, they pose a considerable threat for the original network.

2.2.1 History

Adversarial examples were first investigated by Szegedy et al. (2013). There are various hypotheses why adversarial examples are possible and as re-

search progresses these hypotheses change.

In their first paper, Szegedy et al. (2013) stated that the high non-linearity of neural networks is the reason for adversarial examples. They state that the space on which adversarial examples lie are *pockets* of low probability in the higher dimensional manifold of the input space. This means that even though the network has not seen an input from this *pocket* in the input space, it generalizes to make an assumption. This generalization is most of the time very helpful and the reason why neural networks are so successful. However, when strategically exploiting the existence of such *pockets*, adversarial examples can be created.

One aspect favoring this explanation was the transferability of adversarial examples. Assuming that adversarial examples are a phenomena arising because of a certain structure common to neural networks, having several networks which are vulnerable to the same attack is reasonable.

In a later paper the group changed their position and stated that it is sufficient to have linear behavior in high-dimensional spaces in order for adversarial examples to work (Goodfellow et al., 2014). Because of this linear behavior, a small perturbation on the input of a neural network will make the activation grow linearly while it progresses through the network. This is due to the basic structure of a neural network. Apart from the non-linear activation function, the neural network consists of a linear concatenation of the sums of the dot products of neuron weights and inputs. However, because most modern neural networks use the ReLU activation function in hidden layers, the activation is linear for most parts as well. This results in the network being linear in most regions. Because of this linear growth, a small perturbation in many input dimensions suffices to change the classification of the neural network. However, if there are less dimensions which can be perturbated, the changes have to be considerably higher in order to shift the classification outcome.

2.2.2 Examples of adversarial attacks

There are many different attack methods for image classification problems. In the literature, most of them are divided into two categories. White-box attacks rely on knowledge about the machine learning algorithm they want to attack. Black-box attacks, by contrast, do not rely on this knowledge and thus can be used if the underlying algorithm is unknown. Additionally, attacks can be classified as targeted or non-targeted depending on whether they want to push the classification into a certain class or they just want to produce wrong classifications in general.

Here, we consider two non-targeted white-box attacks. The second attack

we investigate is built upon the first attack. Therefore it is interesting to see if the second attack can achieve better results and is indeed an improvement over the first one.

Fast Gradient Sign Method The fast gradient sign method (FGSM) proposed by Goodfellow et al. (2014) is one of the earliest adversarial attacks. As perturbation, the following term is used:

$$\epsilon \cdot sign(\nabla_X J(\theta, x, y)) \tag{2.8}$$

To calculate the complete adversarial image x^* the perturbation is added to the original image x :

$$x^* = x + \epsilon \cdot sign(\nabla_X J(\theta, x, y)) \tag{2.9}$$

The factor ϵ regulates how intense the perturbation should be. With a smaller ϵ-value the adversarial example seems more natural, however, the classification error might be smaller. With a bigger ϵ-value the adversarial example seems more artificial, but the classification error might also be bigger. The gradient ∇_x of the cost function $J(\cdot)$ — which is dependent on the model's parameters θ — the input image x and the target labels y illustrate how a change in the input changes the cost function.

When adding a positive value to the input, the cost function will increase in case of a positive gradient, whereas a negative gradient will result in a decrease of the cost function. The cost function is a measure to show how high the error of a neural network is, so it is our goal to increase the cost function in order to change the classification of the input. For this, we either have to perturb the input into the direction of the positive gradient or away from the direction of the negative gradient. This is achieved by using the sign function. For a positive input, in this case a positive gradient $\nabla_x J(\cdot)$, the function returns 1. This means that the small ϵ-value will be added to the original input. For a negative input, in our case a negative gradient, the function returns -1. This means that the small ϵ-value will be subtracted from the original input. With this, perturbation is added or subtracted to or from each pixel so that the classification is gradually pushed away from the true label. Originally this attack was done for one step only. However, Kurakin, Goodfellow, and Bengio (2016) found that a one-step FGSM attack causes label leaking. This means that the true labels are still embedded in the adversarial images. Because the crafted adversarial perturbation is easily predictable for such a simple attack, defenses can learn to withstand adversarial images more effortlessly. This phenomenon can increase the accuracy of the model for the created adversarial examples.

They recommended to use iterative attack methods to avoid the leaking label problem. Another way to withstand the problem is the usage of the predicted labels $\hat{y}$ instead of the true labels y in the cost function for the calculation of the gradient $\nabla_x J(\cdot)$, as found empirically by Kurakin et al. (2016). In our implementation of the attack we realized both recommendations to improve our results.

The momentum iterative fast gradient sign method FGSM was developed further by Dong et al. (2017). Their motivation was the shortcoming of iterative FGSM that the adversarial examples are greedily pushed in the direction of the signed gradient. As a result, the method can get stuck in local optima. To overcome this issue, the authors combined momentum with the original approach and established the momentum iterative fast gradient sign method (MI-FGSM). This method balances the current gradient with the sum of the gradients from the previous epochs.

$$g_{t+1} = \mu \cdot g_t + \frac{\nabla_x J(x_t^*, y)}{\|\nabla_x J(x_t^*, y)\|_1} \qquad \text{with} \quad g_0 = 0 \tag{2.10}$$

$$x_{t+1}^* = x_t^* + \epsilon \cdot sign(g_{t+1}) \tag{2.11}$$

A decay factor μ determines how much the gradients of previous epochs are taken into account. When using a value of $\mu = 0$, only the current gradient is taken into consideration, resulting in an equivalent method to the normal iterative FGSM. In order to prevent exploding gradients, the authors suggest to divide the current gradient by its L_1-norm for normalization. This term is added to the sum of the preceding gradients g_t (velocity vector). The velocity vector is then used as input to the sign function for the normal update step of the adversarial example.

With the help of the momentum this method can overcome local optima and thus decrease the accuracy of the targeted neural network.

2.2.3 Example of an adversarial defense

Since the topic of adversarial examples arose, research was also aimed at creating defenses to stop adversarial attacks from affecting machine learning systems. There are two approaches to defending a system. Proactive defenses try to make a classifier more robust against adversaries. Reactive defenses try to detect adversarial examples only after they have been classified.

Defensive Distillation The defense investigated in this bachelor thesis is a proactive defense. When first introduced, network distillation was not meant as a defense mechanism but as a mean to make neural networks more accessible to people with no access to high-computing computers (Papernot et al., 2015). To make this possible, training data with hard class labels (labels only consisting of 1 if it is the right class or 0 if it is not) is fed to a neural network. Once the network is trained and all weights are set, a new data set is created by feeding the data to the neural network once again and saving the probabilistic output as new soft class labels. This new data set is now used to train a second neural network of smaller size. With the original test data we can empirically verify that the accuracy of both networks does not differ much. However, such smaller neural networks need less computations and thus can be used on devices such as smartphones.

The idea behind network distillation as defense is quite similar. Instead of using a smaller second neural network with a marginal loss in accuracy, a second neural network of the same size is used in hope of robustifying the network against adversarial attacks. Using the soft labels to train the second neural network gives additional knowledge to the training process. The soft labels implement information not only about the true class but about the relation to other classes as well. Using this relative information helps the network to generalize well between the training points and to not fit too tightly to the data. This way, the *pockets* in which the adversarial examples lie, are covered by the neural network.

Chapter 3

Methods

Many structures and parameters were chosen following the examples of the original authors of each method. However, some parameters were not exactly determined in the original papers and therefore we tried to find the best values. All implementation choices together with explanations are described in this section.

3.1 Approach

In this bachelor thesis we compare two adversarial attacks trying to overcome one adversarial defense. This is done by first training a baseline neural network, then training a defended neural network and afterwards creating adversarial examples with the two chosen attacks to be used on both networks. Then, the adversarial examples are analyzed qualitatively and in terms of accuracy.

3.1.1 Datasets

In our experiments we used the datasets MNIST (MNIST, 2019) and CIFAR-10 (CIFAR-10, 2019). Both sets are renowned in the field of image classification. Nonetheless, they are relatively small compared to other datasets such as SVHN, CIFAR-100 and ImageNet (Yuan et al., 2017). This allows for training without a high computational power device.

MNIST MNIST is a dataset consisting of handwritten digits from 0 to 9 with according labels. Images are of size 28 x 28 and limited to grayscale. In total there are 70.000 images divided into two parts. The first part contains 60,000 images to be used as training data. These images are scanned

handwriting samples from 250 people, half of whom were US Census Bureau employees, and half of whom were high school students (MNIST, 2019). The second part of the MNIST dataset has 10,000 images to be used as test data. To guarantee an independent test of performance, the test data was taken from a different set of 250 people than the original training data (albeit still a group split between Census Bureau employees and high school students). This serves as an assessment of whether the machine learning system can recognize digits written by people whose handwriting it did not see during training.

CIFAR-10 CIFAR-10 is a dataset consisting of 60.000 colour images of ten mutually exclusive classes.[1] Each image is of dimension 32 x 32 x 3, representing height, width and RGB colour-channel ranging from 0 to 255, respectively. The dataset is split into 50.000 training images and 10.000 test images.

The dataset is a subset of the 80 million tiny images dataset (CIFAR-10, 2019). The original dataset was collected over a period of 8 month by downloading images from 7 independent search engines for ~75.000 non-abstract nouns. In order to make it useable for devices with less computational power, the resolution was downgraded to 32 x 32 pixels. Nonetheless, the complete dataset is still 760 GB in size (Torralba, Fergus, & Freeman, 2008). This is why in 2009 Alex Krizhevsky reduced the set to 10 classes for CIFAR-10 and 100 classes for CIFAR-100 with each class having an equal number of images (CIFAR-10, 2019).

3.2 Implementation

For the implementation we used the Keras (Chollet et al., 2015) extension of Tensorflow (Abadi et al., 2015).

3.2.1 Defensive Distillation

To implement defensive distillation we first trained an undistilled neural network. We used the predicted probabilities of the original dataset as soft labels for a second dataset. Then we used this second dataset to train the distilled neural network. Next to the usual adaptable parameters for this approach such as learning rate and momentum, we also used a parameter T for the temperature of the softmax layer. By changing this temperature T,

[1] Classes are *airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck*

the network's predictions can be made less or more discrete. T is used in the softmax probability computation of class i as follows:

$$q_i = \frac{exp(\frac{z_i}{T})}{\sum_j exp(\frac{z_j}{T})} \tag{3.1}$$

The probability of class i is now dependent on both, the logit z_i and the temperature T. High temperature values decrease the exponent $\frac{z_i}{T}$, resulting in $exp(\frac{z_i}{T})$ being smaller. When all terms are very small, differences between the activation for different classes are less severe, making the output of the whole softmax layer more ambiguous. Reducing the temperature increases the exponent and results in a bigger $exp(\frac{z_i}{T})$. Accordingly, the impact of differences in the activation is increased and the output of the whole softmax layer is less ambiguous. However, the biggest effect is present for temperatures below 1. For these small temperatures the term $exp(\frac{z_i}{T})$ becomes infinitely large as the temperature approaches 0. For such big values the impact of the differences between each value becomes very significant, making the output of the softmax layer more discrete. Papernot et al. (2015) suggested to only use temperatures above 1 for defensive distillation. We followed this suggestion in our implementation.

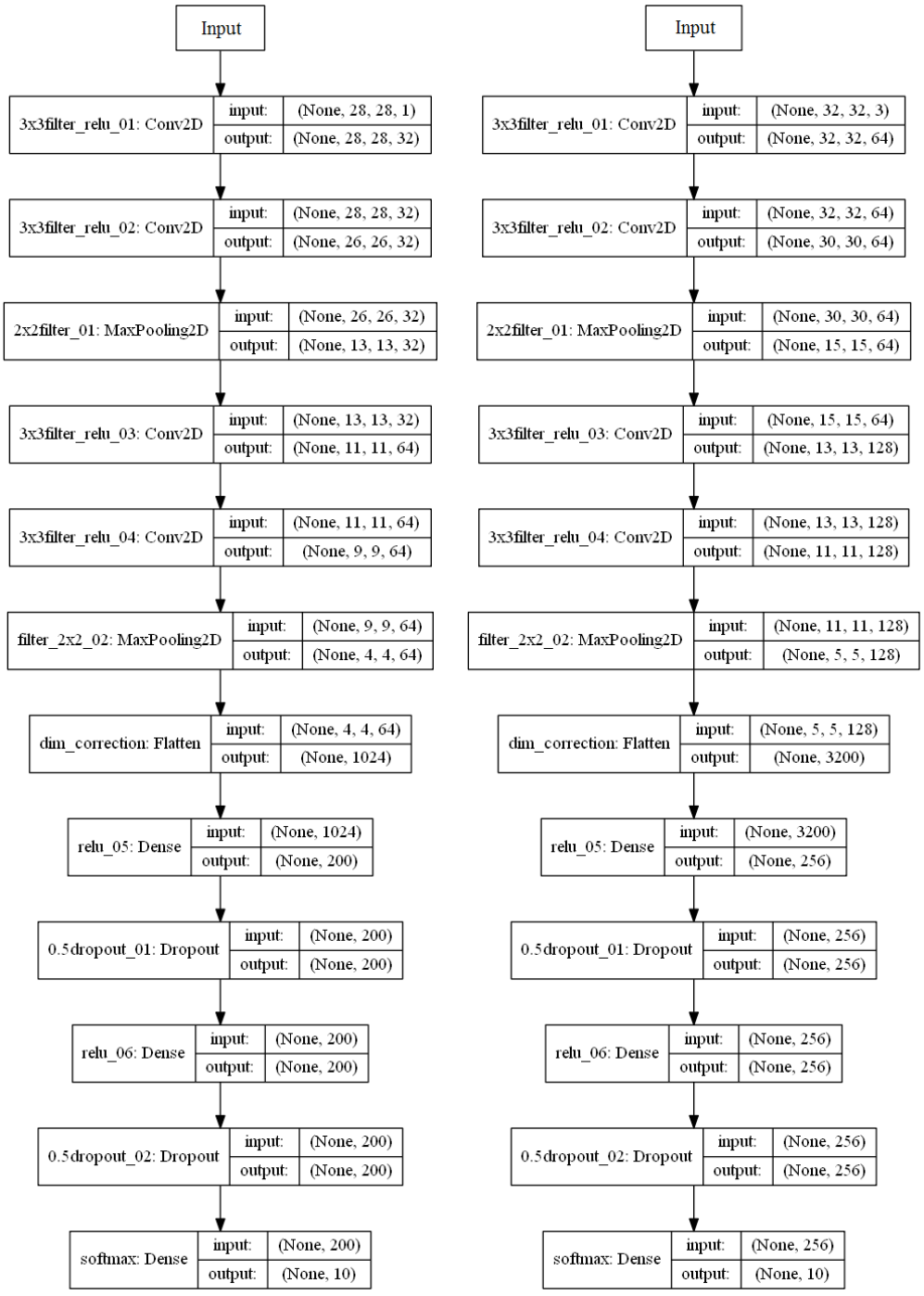
3.2.1.1 Architecture

For the MNIST dataset as well as the CIFAR-10 dataset we trained our networks similar to the paper by Papernot et al. (2015).

For both datasets the original and the distilled network are of the same architecture (*cf. Figure* 3.1).

The network trained on the MNIST dataset is using categorical crossentropy as loss function and a stochastic gradient descent optimizer with a learning rate of 0.1 and momentum of 0.5. In the original paper, 50 epochs with a batch size of 128 were used. However, during our experiments we found that such long training led to overfitting. This is why we made a validation split into 10.000 validation images and 50.000 training images and used a model checkpoint callback to save the model weights with best validation set accuracy. This is an approach called early stopping (Prechelt, 1998). With this, the undistilled model found optimal weights after 18 epochs on average and the distilled model after 25 epochs on average.

The network trained on the CIFAR-10 dataset is also using categorical crossentropy as loss function and a stochastic gradient descent optimizer with an initial learning rate of 0.01 and momentum of 0.9. Following the description of the authors, we used a learning rate decay of 0.95 every 10



(a) MNIST dataset **(b)** CIFAR-10 dataset

Figure 3.1: Network Architectures

epochs. Again, we used a model checkpoint callback to save only the best results of the 50 trained epochs. For CIFAR-10 we had 40.000 training images and 10.000 validation images. For high temperatures (45-95) the undistilled model found optimal weights only on the fiftieth epoch, for low temperatures it found optimal weights after 37 epochs on average. The distilled model also found optimal weights for high temperatures (45-95) only on the fiftieth epoch, for low temperatures it found optimal weights after 47 epochs on average.

3.2.2 Fast Gradient Sign Method

For the implementation of FGSM several parameters had to be fixed. One significant parameter is the ϵ-value. It determines the magnitude of the added perturbation. In literature ϵ is always set to a small value, commonly below 0.01. During our experiments we tried several ϵ-values and examined the resulting adversarial images *(cf. Figure 3.2 and 3.3)*. Weighing the natural appearance of the adversarial images against the decrease in accuracy we chose a value of 0.01 for MNIST and a value of 0.005 for CIFAR-10. For 9 iterations we calculated the adversarial perturbation, using softmax cross entropy as loss function. At the end of every iteration we clipped the adversarial examples to values between 0 and 1.

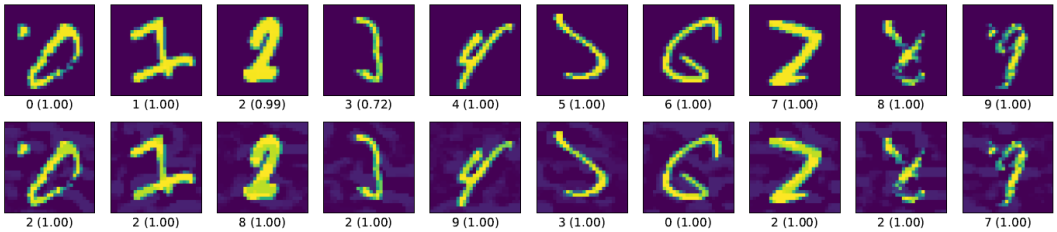
3.2.3 Momentum Iterative Fast Gradient Sign Method

For the implementation of MI-FGSM we used the same parameters as in FGSM. For MNIST 0.01 as ϵ-value and for CIFAR-10 0.005 as ϵ-value. For the additional decay vector μ we used a value of 1. This was suggested by Dong et al. (2017). We also used the built-in softmax cross entropy function of tensorflow as loss. Additionaly, we normalized the gradient by dividing it by its L_1-norm.

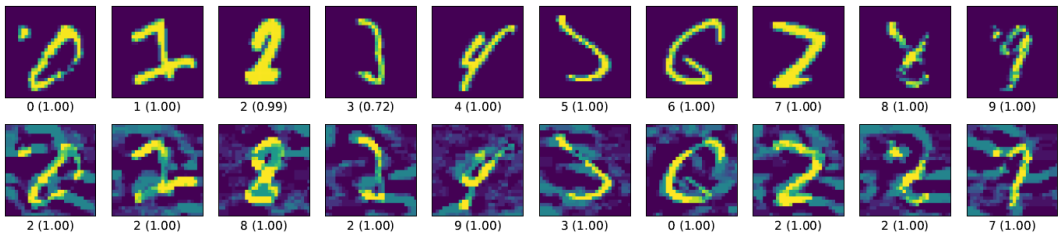
3.3 Experimentation

3.3.1 Setup

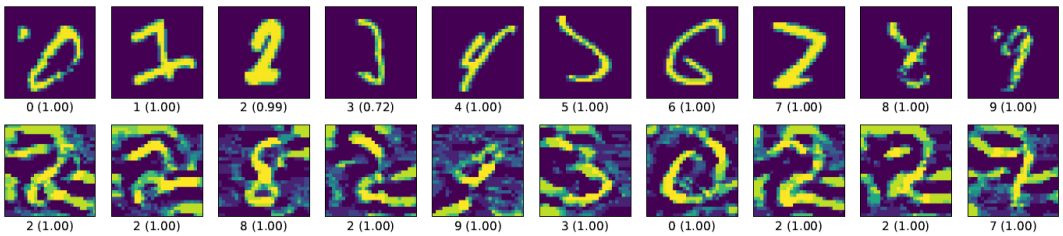
Initially we train the first undistilled model according to the network architecture mentioned above. We use the first network to generate soft labels for all training images and create a modified data set. On this new data set we train the second distilled model in the same manner as the first model. Then we use FGSM and MI-FGSM on both models to create adversarial images



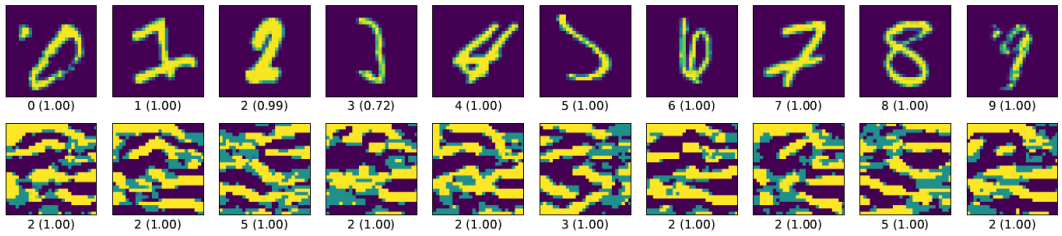
(a) Adversarial examples on MNIST with $\epsilon = 0.01$



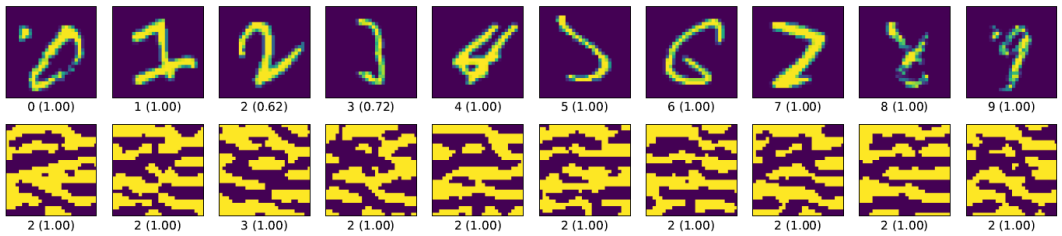
(b) Adversarial examples on MNIST with $\epsilon = 0.05$



(c) Adversarial examples on MNIST with $\epsilon = 0.1$

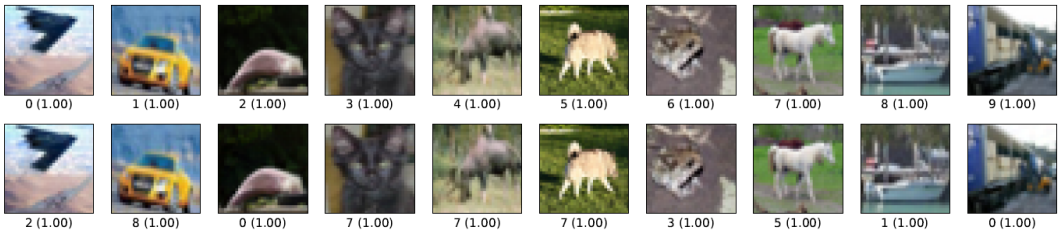


(d) Adversarial examples on MNIST with $\epsilon = 0.5$

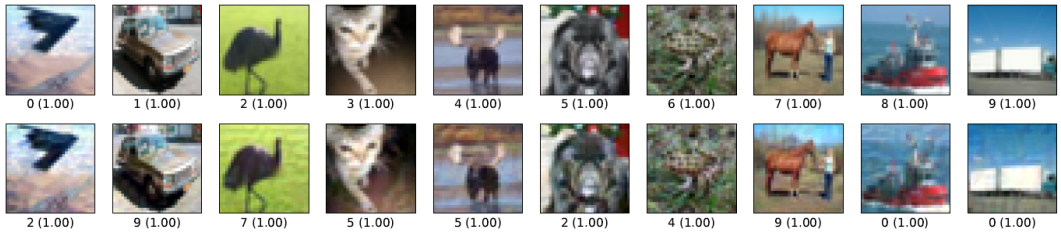


(e) Adversarial examples on MNIST with $\epsilon = 1$

Figure 3.2: Adversarial Images created with FGSM on MNIST for temperature 50



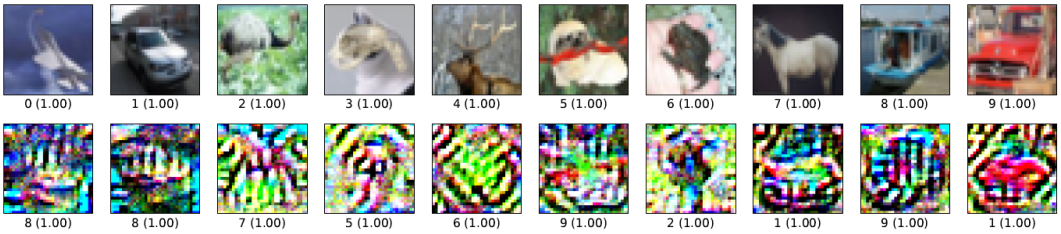
(a) Adversarial examples on CIFAR-10 with $\epsilon = 0.001$



(b) Adversarial examples on CIFAR-10 with $\epsilon = 0.005$

(c) Adversarial examples on CIFAR-10 with $\epsilon = 0.01$

(d) Adversarial examples on CIFAR-10 with $\epsilon = 0.05$



(e) Adversarial examples on CIFAR-10 with $\epsilon = 0.1$

Figure 3.3: Adversarial Images created with FGSM on CIFAR-10 for temperature 50

from the test data. Finally we compare the accuracy of the models on the test data as well as the adversarial data. For testing, the temperature of the softmax layer is set back to 1 as suggested by Papernot et al. (2015) to get more discrete categorizations.

3.3.2 Analysis

We analyze the performance of the defense and the attacks in multiple ways. For this, we always compare the results of neural networks trained with different temperature values ranging between 1 and 95. First, we compare the accuracy of the undistilled and the distilled network for the original network to verify that the defense does not influence the performance. Then we examine the accuracy of the attacks. To do so, we compare the accuracy of the undistilled and the distilled network against the according attack. For each network we take two accuracies into consideration. First, the accuracy of the network when tackled with the adversarial examples which were created using this network. We call them *direct attacks*. Second, the accuracy of the network when attacked by the adversarial examples which were created using the other network. We call them *transferred attacks*. Afterwards we compare the accuracy against direct and transferred attacks. For this we consider undistilled and distilled network. All accuracies mentioned are on the separate test data.

Chapter 4

Results and Discussion

In this chapter, the results of our experimentation are shown. Examples of adversarial images can be found in the appendix *(cf. Chapter 5)*.

4.1 Results

MNIST The undistilled network as well as the distilled network have an accuracy of ~99% on the original dataset for all temperatures. For some temperatures, the undistilled network is almost indistinguishably better, for other temperatures the distilled network is better *(cf. Figure 4.1)*.

When attacked by the FGSM attack, both networks lose accuracy. For higher temperatures (35-95) the accuracy loss is rather small, between 0.1 and 0.6 percentage points, so that both networks remain an accuracy of ~99%. For smaller temperatures (1-30) the accuracy decreases further. The undistilled network reaches its minimal accuracy at temperature 1 with 92.9%. The distilled network also reaches its minimal accuracy at temperature 1 with 92.3%. This corresponds to an accuracy loss of ~7 percentage points compared to the accuracy on the original dataset. The difference between undistilled and distilled network, however, is negligible (*cf. Figure 4.2a*).

Comparing direct FGSM attacks with transferred FGSM attacks, we can also see the trend that for higher temperatures (35-95) the accuracy decreases only little staying at ~99%. For these temperatures the transferred attacks are more successful, however, the difference is barely noticeable. For smaller temperatures (1-30) we can notice, that the direct FGSM attacks perform better by lowering the accuracy down to 89,2% at temperature 1. The transferred FGSM attacks only decrease the accuracy to 96% at temperature 1. Yet, there is a small upwards trend noticeable for the lowest few temperatures. But the increase is below 1 percentage point so it is arguable if this increase

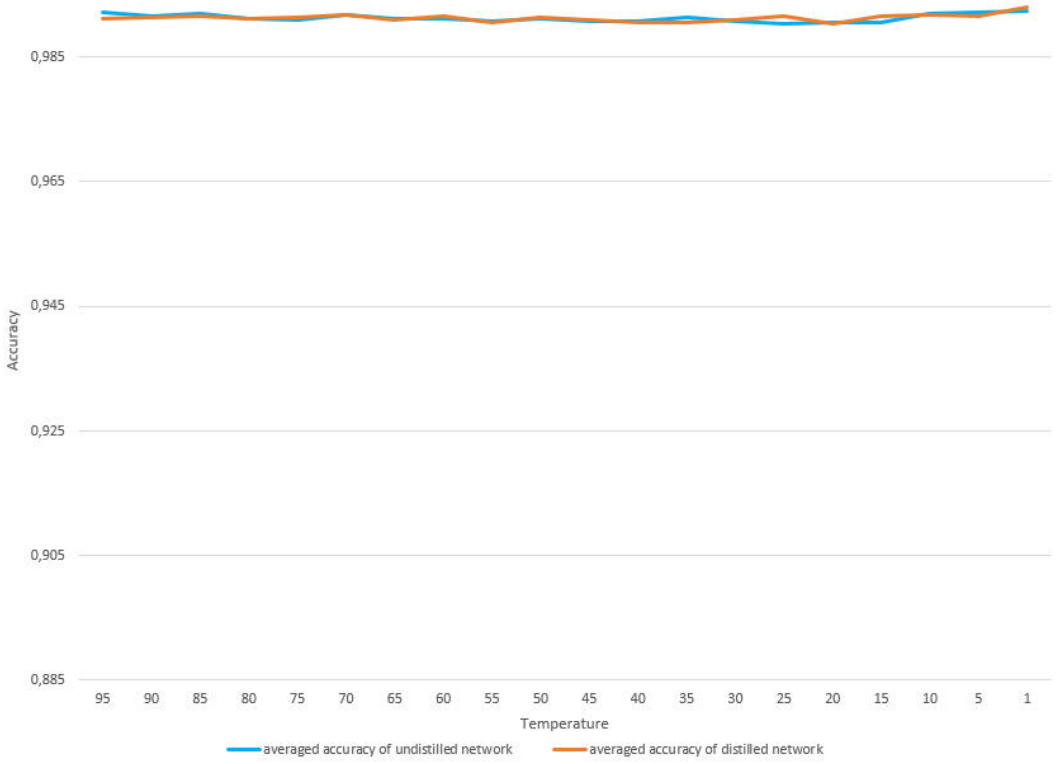
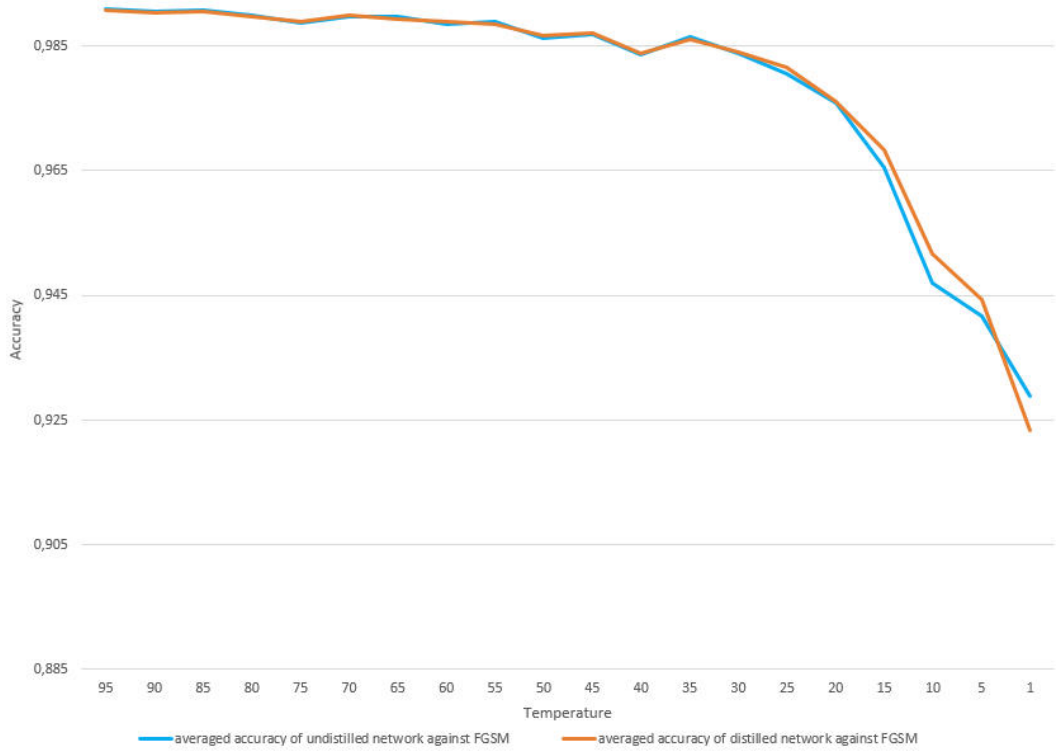


Figure 4.1: Averaged accuracy of the undistilled network (*blue*) and the distilled network (*orange*) for temperatures ranging from 1 to 95.

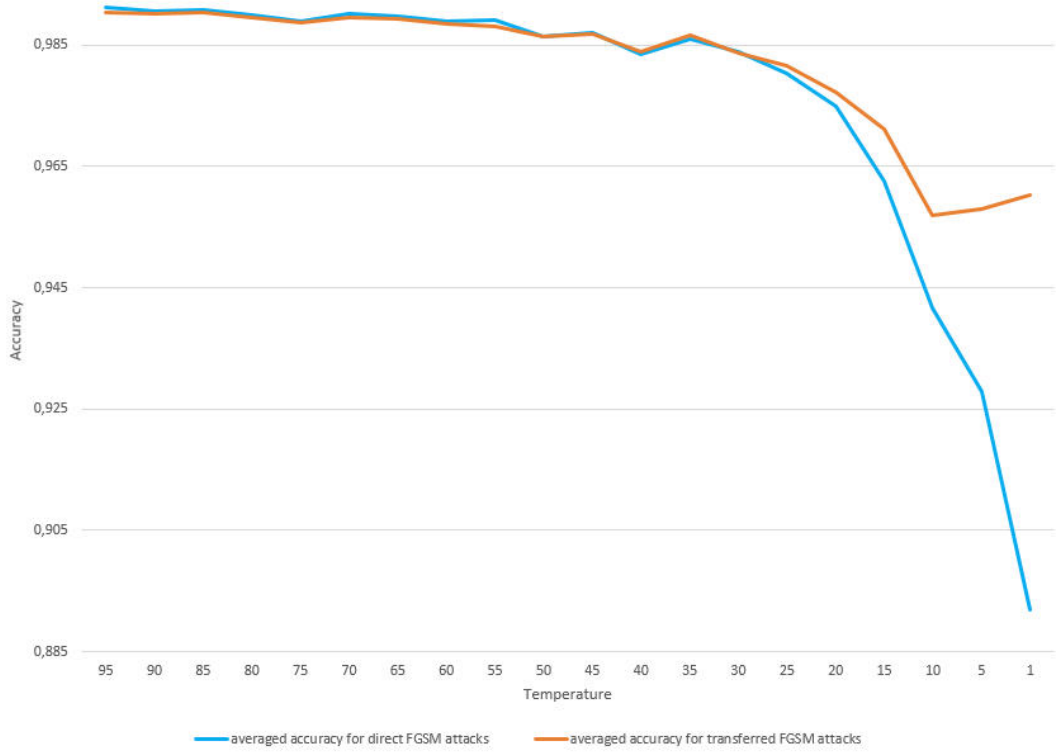
is mere chance (*cf. Figure 4.2b*).

When looking at the accuracy against the MI-FGSM attack we can see that the neural networks have higher accuracy for high temperatures than for low temperatures. However, for higher temperatures (35-95) the difference is rather small with the accuracy always staying ~99%. For smaller temperatures (1-30) the accuracy decreases down to 93% at temperature 1 for both the undistilled and the distilled network. This is an accuracy loss of ~6% as to the original dataset. Comparing the MI-FGSM attack against the undistilled network and the distilled network, there is no superior network. For some temperatures the undistilled network has higher accuracy for other temperatures the distilled network has higher accuracy, however, the difference is always below 1 percentage point *(cf. Figure 4.3a)*.

Figure 4.3b shows that for low temperatures (1-35) direct MI-FGSM attacks are, more successful than transferred MI-FGSM attacks. For high temperatures (40-95), on the other hand, the transferred attacks perform better. In the case of high temperatures, however, the difference is below 1 percentage

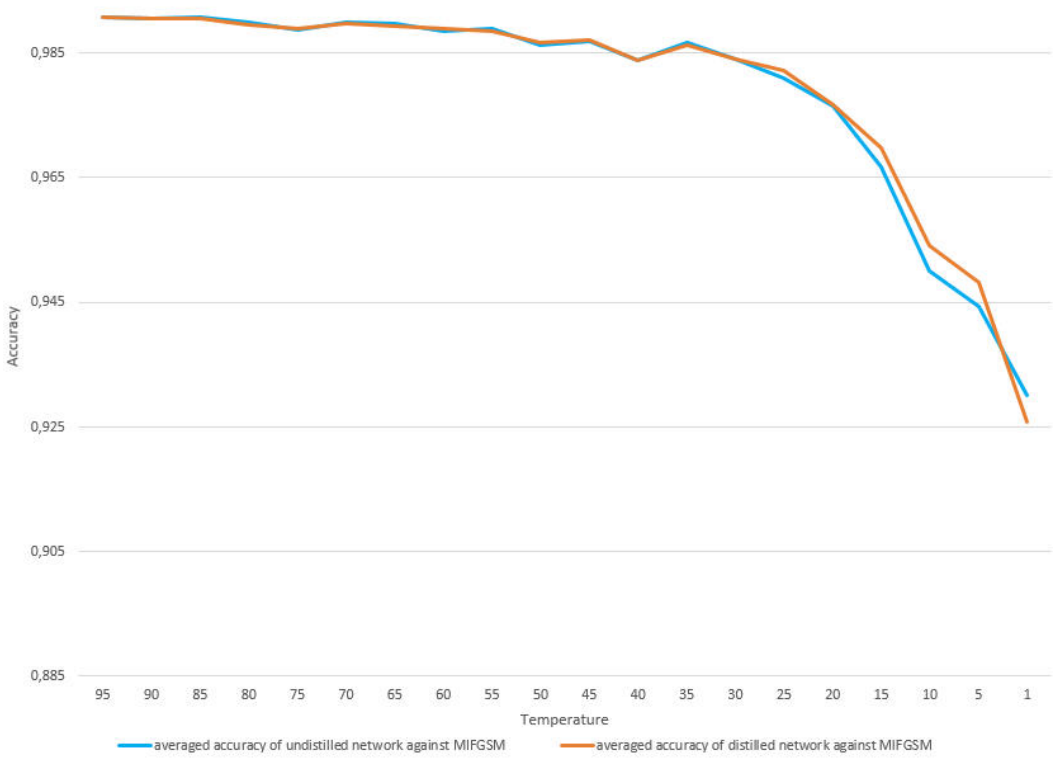


(a) Averaged accuracy of the undistilled network (*blue*) and the distilled network (*orange*) against FGSM attack for temperatures ranging from 1 to 95.

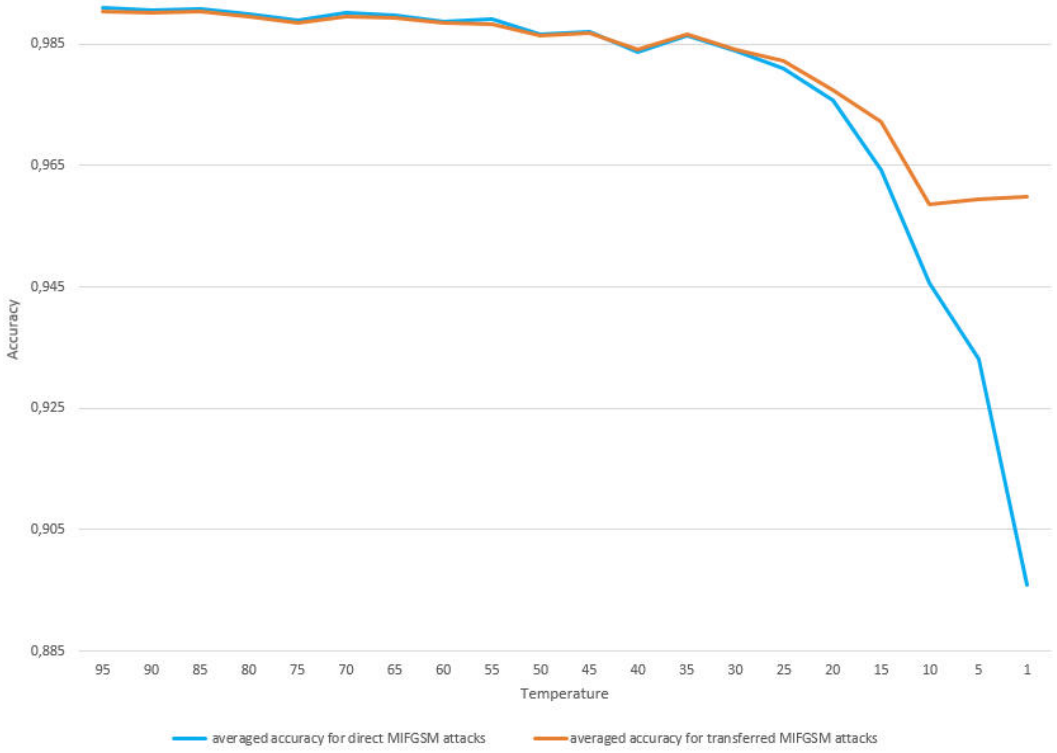


(b) Averaged accuracy of the accuracy against direct (*blue*) and against transferred (*orange*) FGSM attacks for temperatures ranging from 1 to 95.

Figure 4.2: Accuracies of neural networks trained on MNIST when attacked with FGSM



(a) Averaged accuracy of the undistilled network (*blue*) and the distilled network (*orange*) against MI-FGSM attack for temperatures ranging from 1 to 95.



(b) Averaged accuracy of the accuracy against direct (*blue*) and against transferred (*orange*) MI-FGSM attacks for temperatures ranging from 1 to 95.

Figure 4.3: Accuracies of neural networks trained on MNIST when attacked with MI-FGSM

Figure 4.4: Averaged accuracy of the undistilled network (*blue*) and the distilled network (*orange*) for temperatures ranging from 1 to 95.

point, with both accuracy staying ~99%. The difference then rises to 6,4 percentage points at temperature 1 with 89,6% for direct MI-FGSM attacks and 96% for transferred MI-FGSM attacks. As with the transferred FGSM attack, there is a small upwards trend noticeable for the smallest few temperatures.

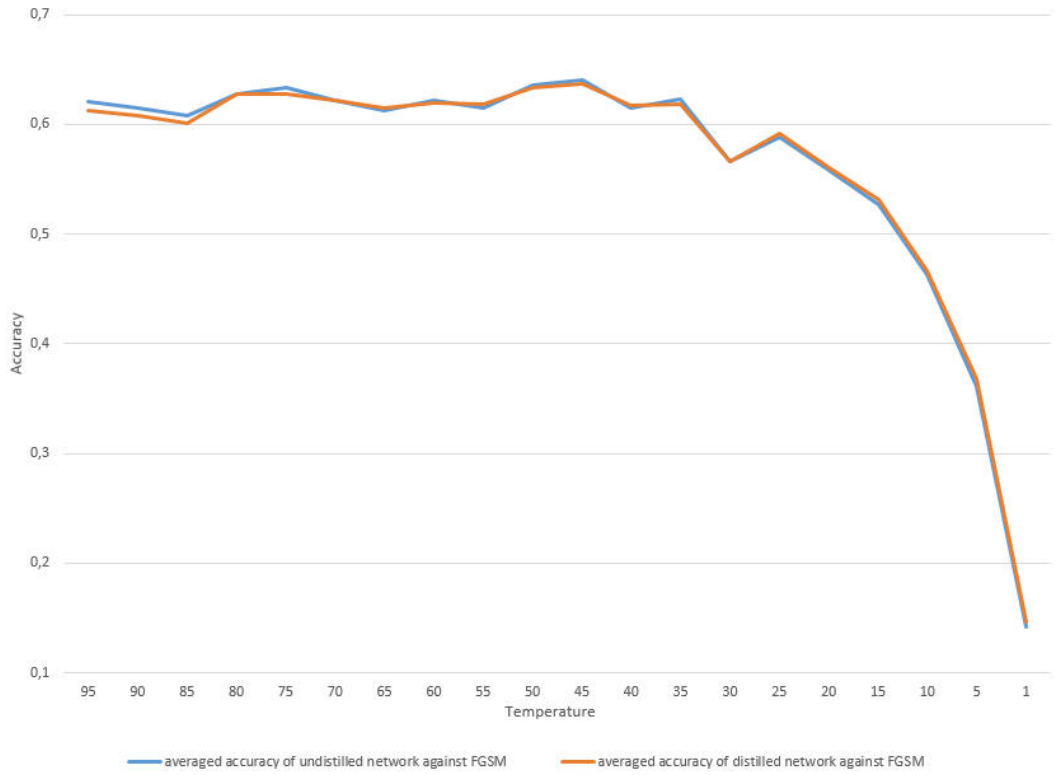
CIFAR Our undistilled network reaches its best accuracy of 75,34% on the original dataset at temperature 1. The distilled network also reaches its best accuracy 76,89% at temperature 1. Going more into detail we can see that for large temperature the undistilled networks performs slightly better. However, the difference only amounts to 1 to 2 percentage points. In the middle region (temperatures 30-70) the two networks perform equally well. For smaller temperatures, the distilled network outperforms the undistilled network, however, again just by 1 to 2 percentage points *(cf. Figure 4.4).*

When attacked with the FGSM attack both networks lose accuracy. However, the loss of accuracy is for most temperatures relatively small. Both networks stay just above 60% accuracy for temperatures between 35 and 95, resulting in a loss of 4 to 8 percentage points compared to the accuracy on the original dataset. For smaller temperatures, however, the accuracy decreases rapidly. At temperature 1, both networks — undistilled and distilled — reach their minimum with an accuracy of ~14%. At that point, the difference between the accuracy on the original dataset and the adversarial dataset is ~61 percentage points *(cf. Figure 4.5a).*

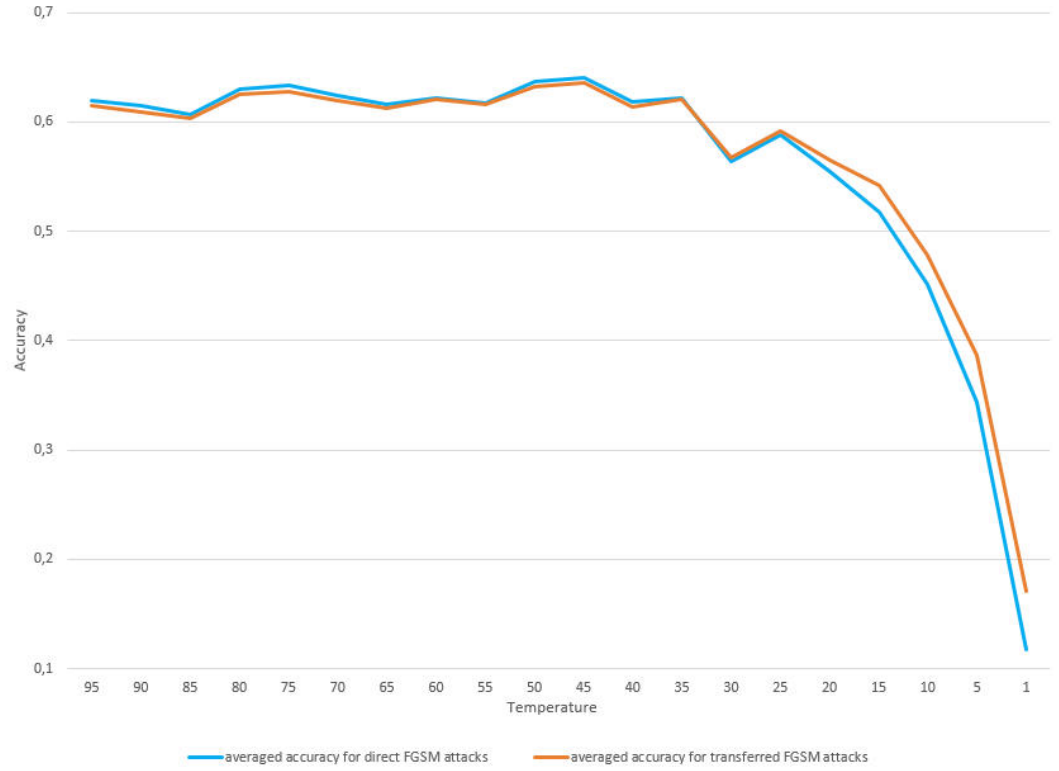
Looking at *Figure 4.5b* we can once again see that for high temperatures down to 35 the accuracy does not really change, always staying just above 60% for both direct and transferred attacks. For these temperatures the transferred attacks perform marginally better by lowering the accuracy a little more. However, this difference stays below 1 percentage point. For smaller temperatures between 30 and 1 the accuracy drops for both types of attack. Here, the direct attacks perform up to 6 percentage points better. The highest difference is reached at temperature 1 when both accuracies reach their minimum. The networks tackled with transferred attacks now only have an accuracy of 17,1% left and the networks tackled with direct attacks even fall to 11,7% accuracy.

Against the MI-FGSM attack we can also register a loss in accuracy. For higher temperatures (35-95) the loss is again rather small staying between 4 to 8 percentage points. There, the accuracy for both networks — undistilled and distilled — stays between 60% and 65%. For lower temperatures (1-30) the accuracy loss is more severe, reaching its maximum of ~62 percentage points at temperature 1. At this temperature, the undistilled network has an accuracy of 13,7% and the distilled network has an accuracy of 14%. The difference between the undistilled and the distilled network stays below 1 percentage point all the time without any network showing a consistently better performance *(cf. Figure 4.6a).*

When comparing direct and transferred MI-FGSM attacks, the change of accuracy for higher temperatures (35-95) is small as in the other experiments, staying below 1 percentage point with transferred attacks performing marginally better. Compared to the accuracy on the original dataset, the accuracy loss is only between 4-8 percentage points. When looking at the lower temperatures (1-30) we can see that the accuracy sinks rapidly. Compared to the accuracy on the original dataset the difference is ~63 percentage points at temperature 1. There, the difference between direct and transferred attacks is also biggest with 3,8 percentage point. At this temperature, direct attacks have an accuracy of 11,9% and transferred attacks have an accuracy of 15,7% *(cf. Figure 4.6b).*

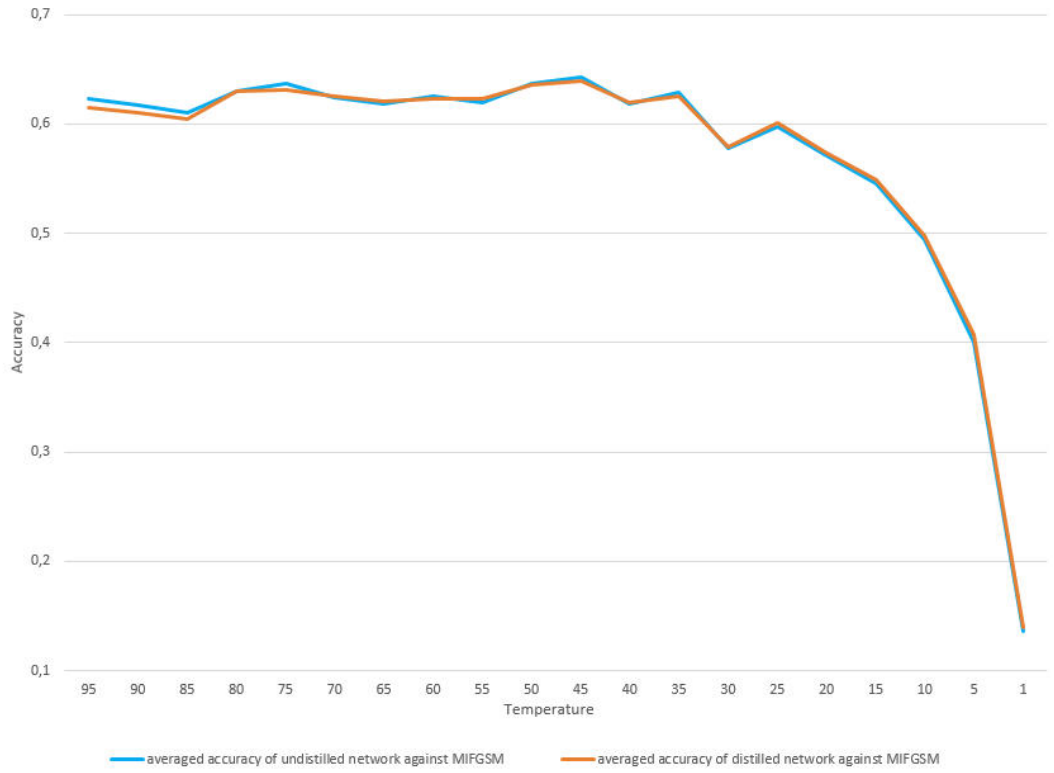


(a) Averaged accuracy of the undistilled network (*blue*) and the distilled network (*orange*) against FGSM attack for temperatures ranging from 1 to 95.

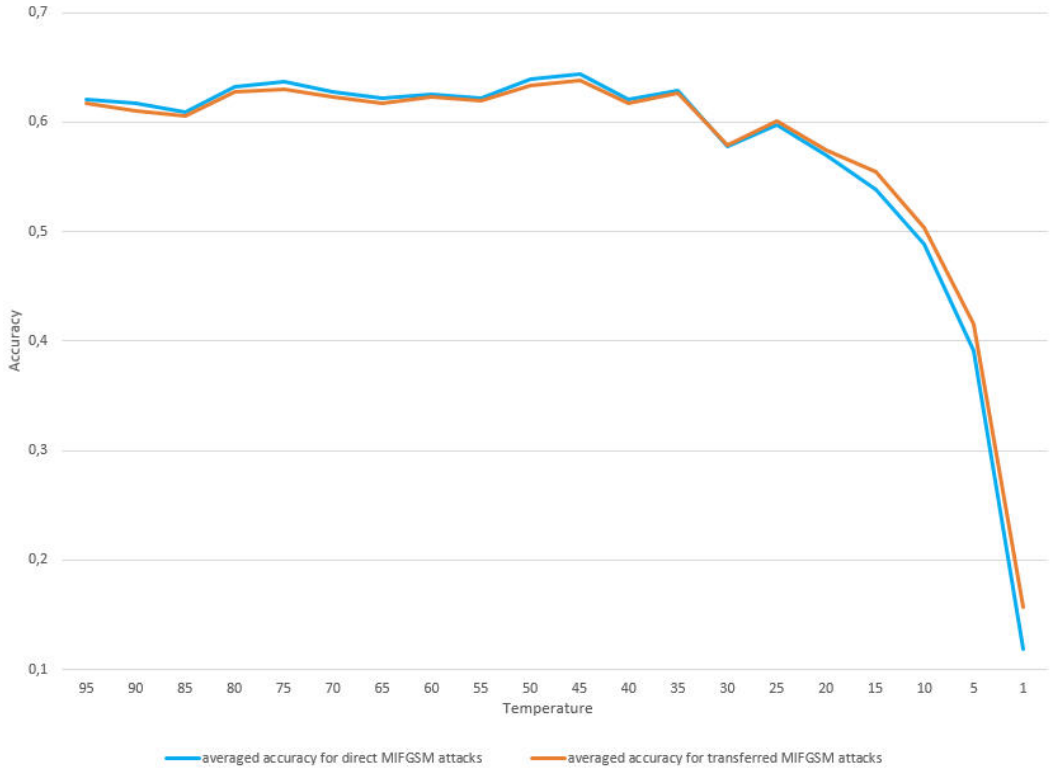


(b) Averaged accuracy of the accuracy against direct (*blue*) and against transferred FGSM attacks (*orange*) for temperatures ranging from 1 to 95.

Figure 4.5: Accuracies of neural networks trained on CIFAR when attacked with FGSM



(a) Averaged accuracy of the undistilled network (*blue*) and the distilled network (*orange*) against MI-FGSM attack for temperatures ranging from 1 to 95.



(b) Averaged accuracy of the accuracy against direct (*blue*) and against transferred MI-FGSM attacks (*orange*) for temperatures ranging from 1 to 95.

Figure 4.6: Accuracies of neural networks trained on CIFAR when attacked with MI-FGSM

4.2 Discussion

From our results, various trends can be identified. The most prominent trend is that as the temperature decreases, the accuracy on the original dataset increases whereas the accuracy on adversarial images generally falls with the temperature. So when using a softmax activation layer with temperature, one has to weigh off classification accuracy and robustness against adversarial attacks.

Effectiveness Defensive Distillation The fact that for higher temperatures (35-95) the accuracy of the distilled network does not decrease significantly when attacked suggests that for these temperatures defensive distillation is a successful defense. However, comparing the accuracy of the distilled network with the undistilled network when facing an adversarial attack, the differences are neglectable. This suggest that the distilled network is not more or less secure against an adversarial attack than an undistilled network. However, the undistilled network might not be the best base case to compare accuracy to. Because of the temperature parameter used in the softmax layer, one might argue that the neural network is already to some extent defended. So to get an authentic base case we have to consider the accuracy of a network without the temperature parameter. For temperature 1, softmax with temperature equation 3.1 can be reduced to the normal softmax equation 2.2 , yielding such a base case. Comparing the accuracy of the distilled network with this base case of temperature 1, we can see, that network distillation indeed increases the accuracy against adversarial attacks, with raising temperature even more so. However, just using higher temperatures without additional distillation achieves nearly equal results and would therefore suffice as a countermeasure of adversarial examples.

Effectiveness Adversarial Attacks Comparing the accuracy on the original datasets and the adversarial images, we can say that the adversarial attacks are not very successful for high temperatures. We investigated this behaviour more closely by looking at the added perturbations. We found that for many adversarial images the perturbation was 0, so the images did not differ from the original ones *(cf. Figure 4.7)*.

We then looked at the different components of the attack algorithm to find the reason for the nonexistence of the perturbation. For the fast gradient sign method *(cf. Equation 2.8)*, the basic components are ϵ, the sign function and the gradient of the loss function. ϵ is a fixed variable during our experiment determining the magnitude of the perturbation. Hence it does not affect whether there is perturbation at all or not. The sign function can take on three

values, namely -1, 1 and 0. This determines that for positive gradients the perturbation will be added to the original image and for negative gradients the perturbation will be subtracted. However, the third case is of interest: for a gradient equal 0 the sign will also be 0, resulting in a perturbation of 0. In our implementation, the gradient is dependent on the original image and the loss function. The gradient of a function depicts the slope: how much does the output change when we change the input by 1 unit. So in our case it illustrates the ratio of how a change of the original image also changes the loss function. This term will be 0 when a change in the input image x will not affect the loss H, independent of the change. We plotted the loss for the images for which there was no perturbation and found that the loss was also 0. As loss H we used the integrated softmax cross entropy function of tensorflow. As input we used the probability of the predicted labels of the original images y_i and the logits of the network when fed the adversarial example of the current epoch. The softmax cross entropy function uses the logits to calculate the probability of the predicted classes for the current adversarial example $\hat{y}_i$.

$$H(y, \hat{y}) = \sum_i y_i \cdot \log(\frac{1}{\hat{y}_i}) \tag{4.1}$$

This equation can be transformed to a simpler version:

$$H(y, \hat{y}) = -\sum_i y_i \cdot \log(\hat{y}_i) \tag{4.2}$$

The cross entropy $H(y, \hat{y})$ is 0 if for all i the dot product $y_i \cdot \log(\hat{y}_i)$ is 0. This is achieved when either $y_i = 0$ or $\log(\hat{y}_i) = 0$. The term y_i, in our case the probability of the predicted label, is only 0 if the neural network is very sure that the input image is not in class i. Setting the temperature parameter of the softmax layer back to 1 at test time as Yuan et al. (2017) suggested pushes the network to make more discrete classifications. Meaning that all but one y_i will be very close to 0 and only y_p of the predicted class p will be close to 1. This could be a reason why in our case y_i is so often 0. However, as the output of the softmax activation layer has to be a valid probability distribution of which the elements will sum to 1, the remaining probability prediction y_p will be 1, symbolizing that the network is very sure that the given image is of class p. In this case, $\log(\hat{y}_p)$ needs to be 0, in order for the dot product $y_p \cdot \log(\hat{y}_p)$ to be 0. The $\log(\hat{y}_p)$ is 0 if $\hat{y}_p$ is 1. This means, that the neural network must be very certain, that the adversarial input – even though there is a small perturbation – is still of class p as originally predicted.

Because in all cases the dot product is 0, the sum $\sum_i y_i \cdot \log(\hat{y}_i)$ will therefore equal to 0 and yield in a loss of 0. However, with a loss always equal 0 the gradient will be 0 as well so that no perturbation can be calculated. Summing up, we can say that our neural network is for high temperatures too sure in its predictions resulting in too discrete categorizations. This way the loss is 0 and no attack can be done with the algorithm.

However as figure 4.7 shows, this does not hold for low temperatures. The number of adversarial examples with an perturbation of 0 decreases together with the temperature. For temperature 1, all adversarial images have a perturbation greater than 0. To explain this we need to have a look at the reason why for high temperatures the network is too discrete. During training, the high temperature of the network makes the output of the activation function more ambiguous. But our true labels are discrete, so an ambiguous output will have a high error. So during training, the weights of the network are adapted in such a way, that the output will become more discrete. For this, the input to the activation function has to be larger in order to balance the high temperature value in the denominator of the exponent *(cf. Equation 3.1)*. So when setting back the temperature to a value of 1 at test time, the input to the activation function will still be very large, however, there is no temperature to balance this. Therefore the output of the activation becomes too discrete.

However, when using low temperatures during training this effect is less severe. For low temperatures the output is less ambiguous from the start, so that during training the network weights will not be pushed too far in a discrete direction. During testing, the output will not be overly discrete which is beneficial.

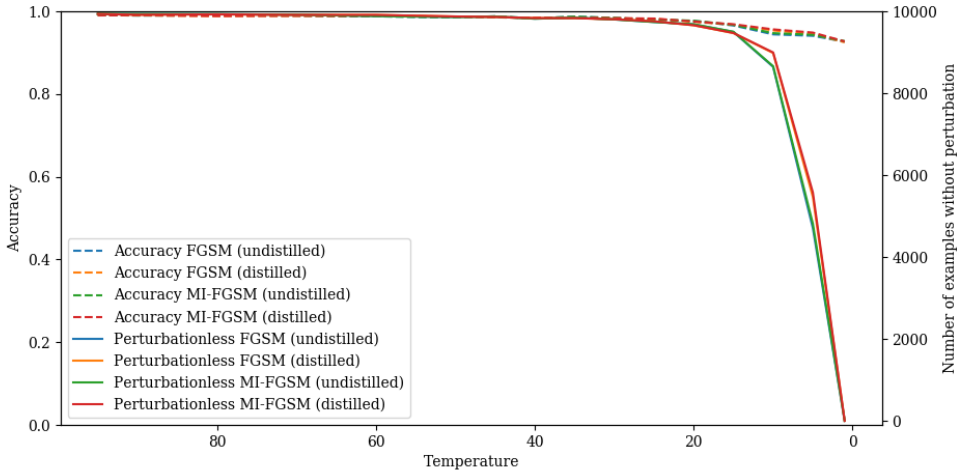
Effectiveness MI-FGSM compared to FGSM With this finding the fact that the results of FGSM and MI-FGSM do not differ significantly can be explained as well. For the momentum iterative fast gradient sign method *(cf. Equation 2.11)*, the basic components are very similar to FGSM. The effect of ϵ and the sign function stay the same and can be explained as above. The difference is in the input of the sign function, changing from just the current gradient to a velocity vector g_t which takes into account all gradients of the previous epochs. This means that for the first epoch, both attacks will give equal results since there are no previous gradients to consider yet. As mentioned above, the FGSM attack fails to perturb the image when the gradient is zero for all epochs. Since for the first epochs both attacks behave the same, we have to investigate the algorithm's behaviour from the second epoch onward. If the gradient was zero in the first epoch, the velocity vector of the second epoch will again only take into consideration the current

gradient. Resulting in both attacks being equivalent again. We can see that as long as the gradient stays zero, FGSM and MIFGSM will have equal results.

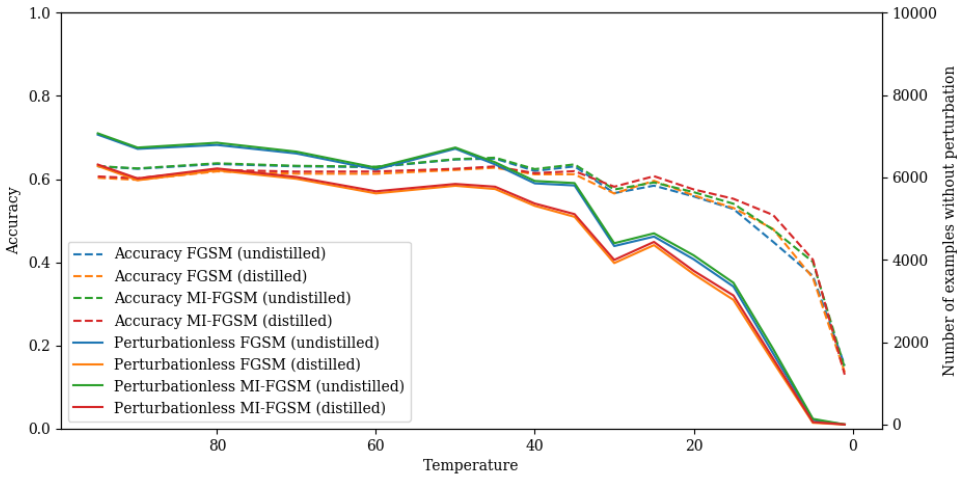
Suitability MNIST Another conclusion we can draw from our results is that the MNIST dataset is too simple for such a deep neural network. The images are only of size 28 x 28, resulting in an input dimension of 784. As mentioned in section 2.2.1 for a high number of dimensions, a small perturbation suffices to change the classification, for a smaller number of dimensions, however, the perturbation has to be bigger. In our implementation the magnitude parameter ϵ for MNIST was 0.01, which is a higher value than what we choose for CIFAR-10, but seems to be too low to produce effective adversarial examples nonetheless.

4.2.1 Conclusion

In this thesis, adversarial examples and ways to protect a machine learning system against them were investigated. Specifically, FGSM and MI-FGSM were employed of which MI-FGSM was found to be marginally better. Defending the neural network with defensive distillation was shown to be of small relevance compared to the highly beneficial use of temperature. Throughout all experiments, making use of a temperature higher than 35 proved to results in very good performance. These results imply that the use of temperature might more broadly offer an effective defense against gradient based adversarial attacks.



(a) MNIST



(b) CIFAR-10

Figure 4.7: Illustration of the number of perturbationless adversarial examples *(continuous line)* out of 10.000 and the accuracy *(dotted line)* for the same temperature. The accuracies are on the dataset created by the mentioned attack and with either the undistilled or distilled network. For each dataset we calculated the number of images which did not differ from the original dataset.

Chapter 5

Appendix

As an illustration we show several examples of adversarial images in this appendix. All figures follow the same structure. Each column represents an image from the according class. For MNIST these are the digits from 0 to 9 For CIFAR-10 the classes are *airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck*. The rows are showing the following data:

1. Original images with prediction and probability of undistilled network
2. Adversarial images created using undistilled network with prediction and probability of undistilled network (direct attack)
3. Adversarial images created using distilled network with prediction and probability of undistilled network (transferred attack)
4. Original images with prediction and probability of distilled network
5. Adversarial images created using undistilled network with prediction and probability of distilled network (transferred attack)
6. Adversarial images created using distilled network with prediction and probability of distilled network (direct attack)

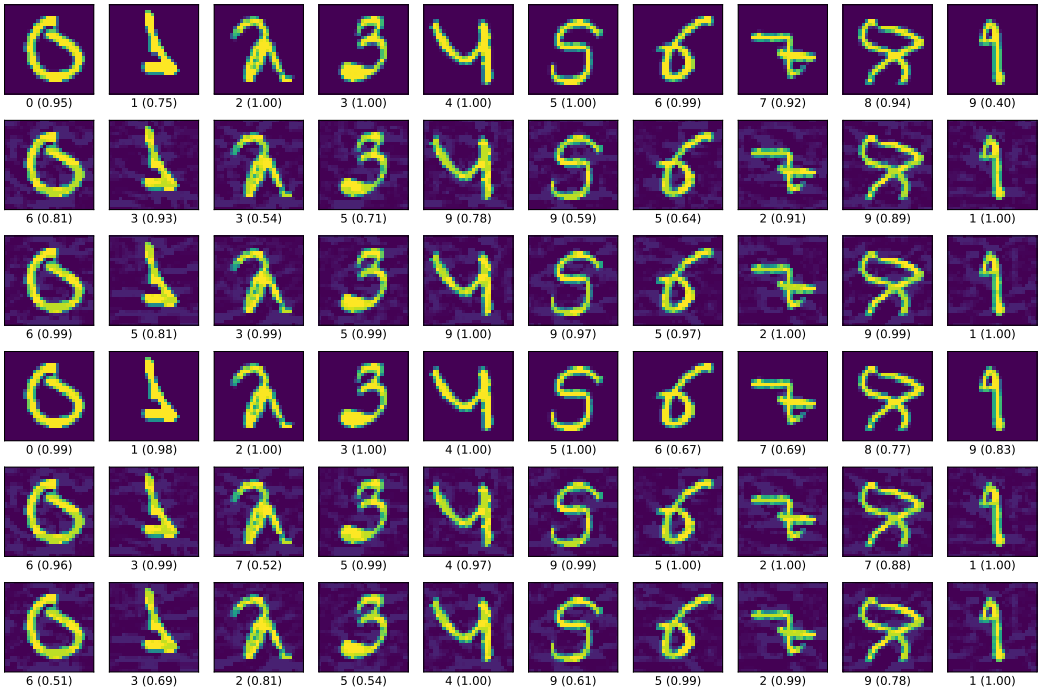


Figure 5.1: Images from the MNIST dataset and adversarial images created using FGSM classified using neural networks with a temperature of 1

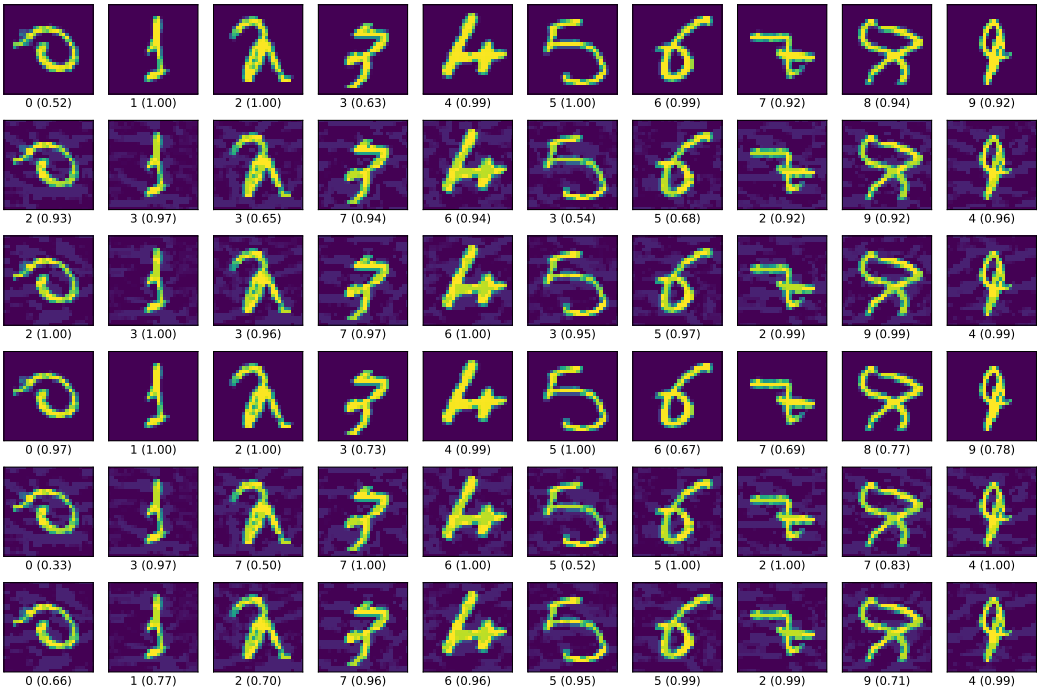


Figure 5.2: Images from the MNIST dataset and adversarial images created using MI-FGSM classified using neural networks with a temperature of 1

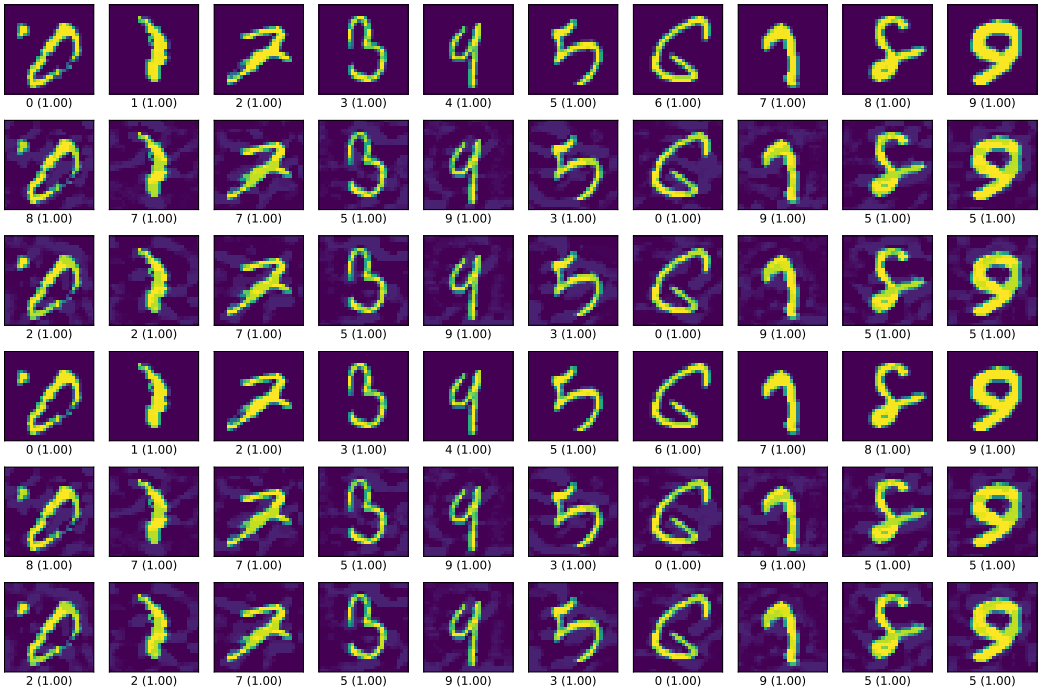


Figure 5.3: Images from the MNIST dataset and adversarial images created using FGSM classified using neural networks with a temperature of 35

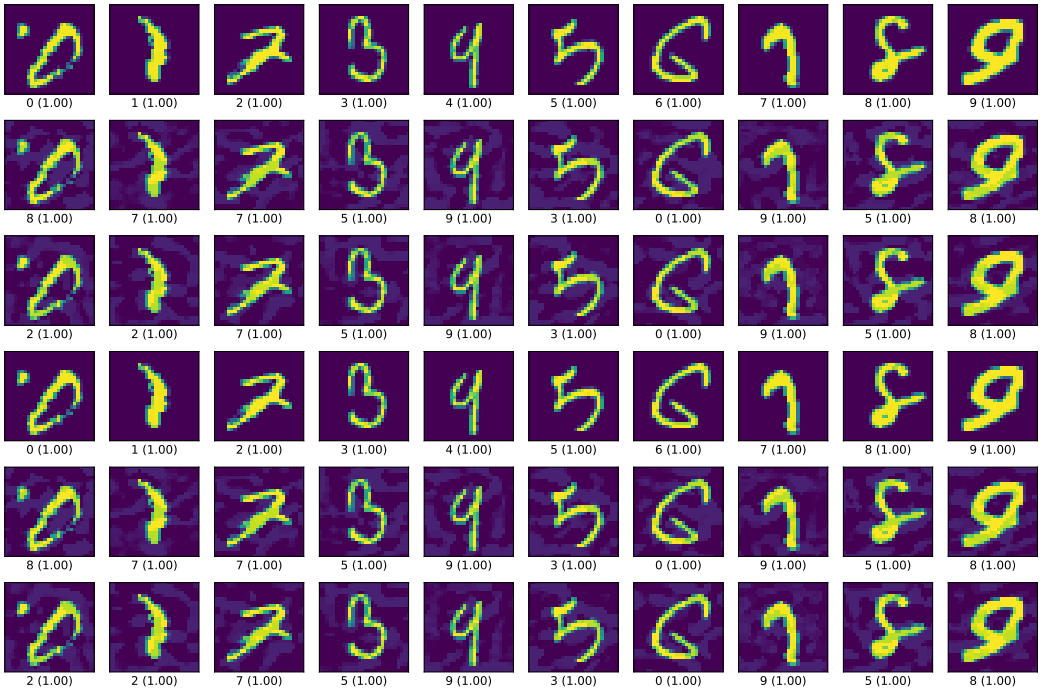


Figure 5.4: Images from the MNIST dataset and adversarial images created using MI-FGSM classified using neural networks with a temperature of 35

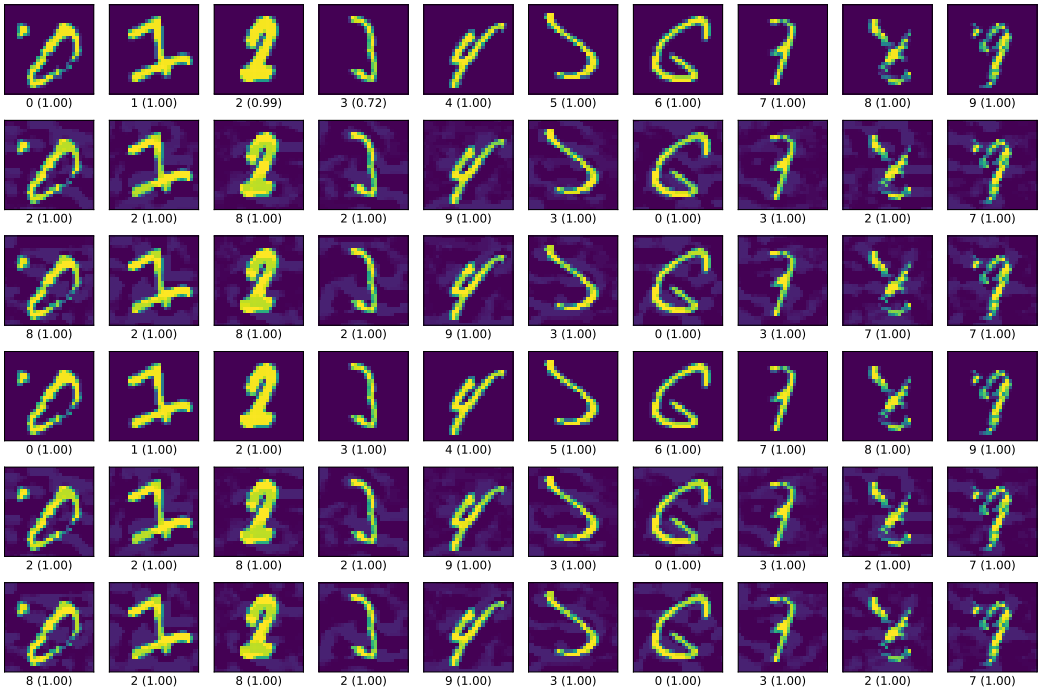


Figure 5.5: Images from the MNIST dataset and adversarial images created using FGSM classified using neural networks with a temperature of 50

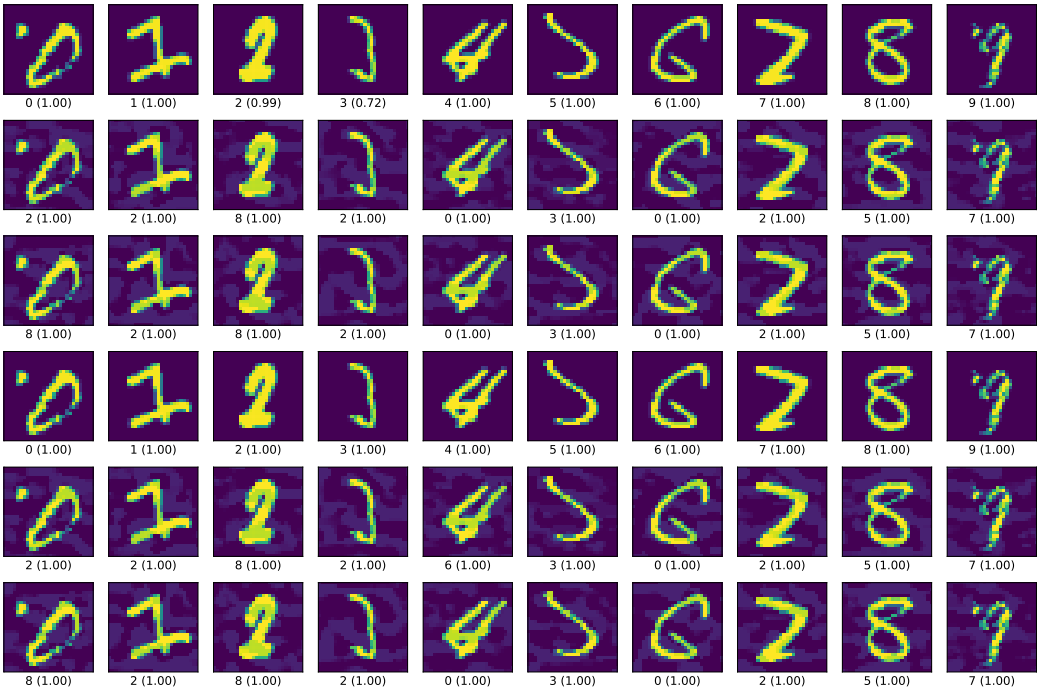


Figure 5.6: Images from the MNIST dataset and adversarial images created using MI-FGSM classified using neural networks with a temperature of 50

Figure 5.7: Images from the CIFAR dataset and adversarial images created using FGSM classified using neural networks with a temperature of 1

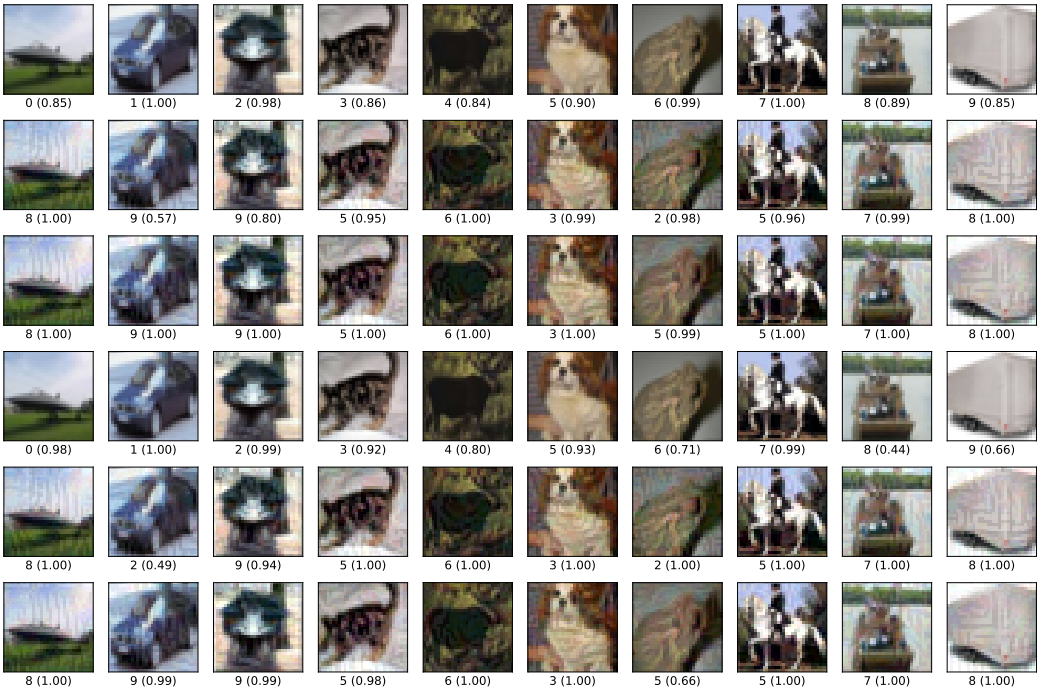


Figure 5.8: Images from the CIFAR dataset and adversarial images created using MI-FGSM classified using neural networks with a temperature of 1

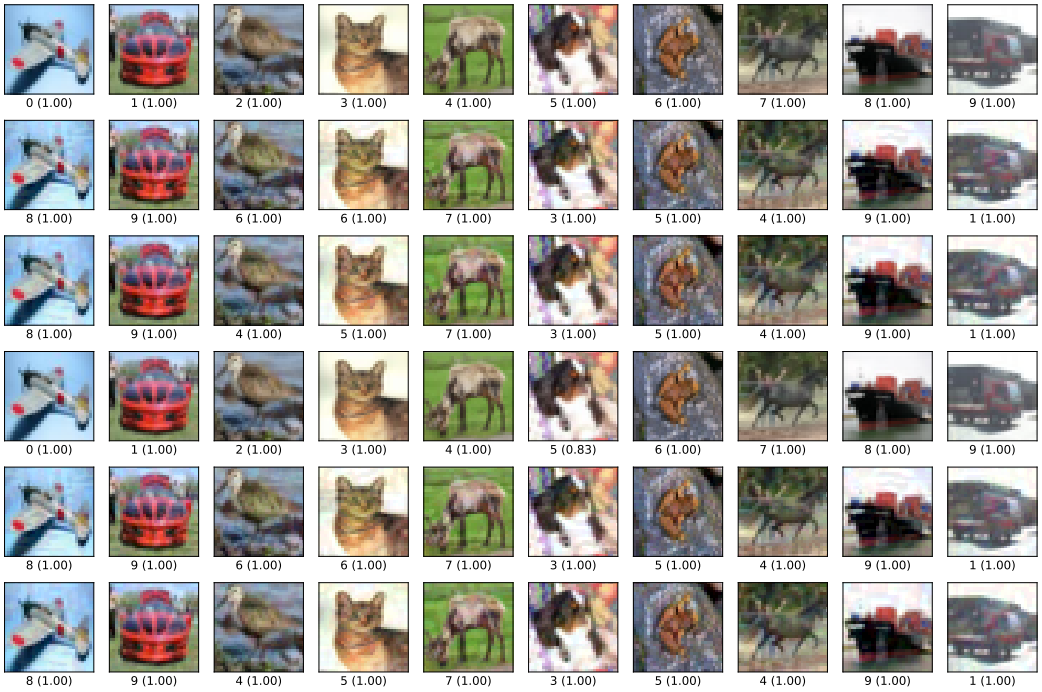


Figure 5.9: Images from the CIFAR dataset and adversarial images created using FGSM classified using neural networks with a temperature of 35

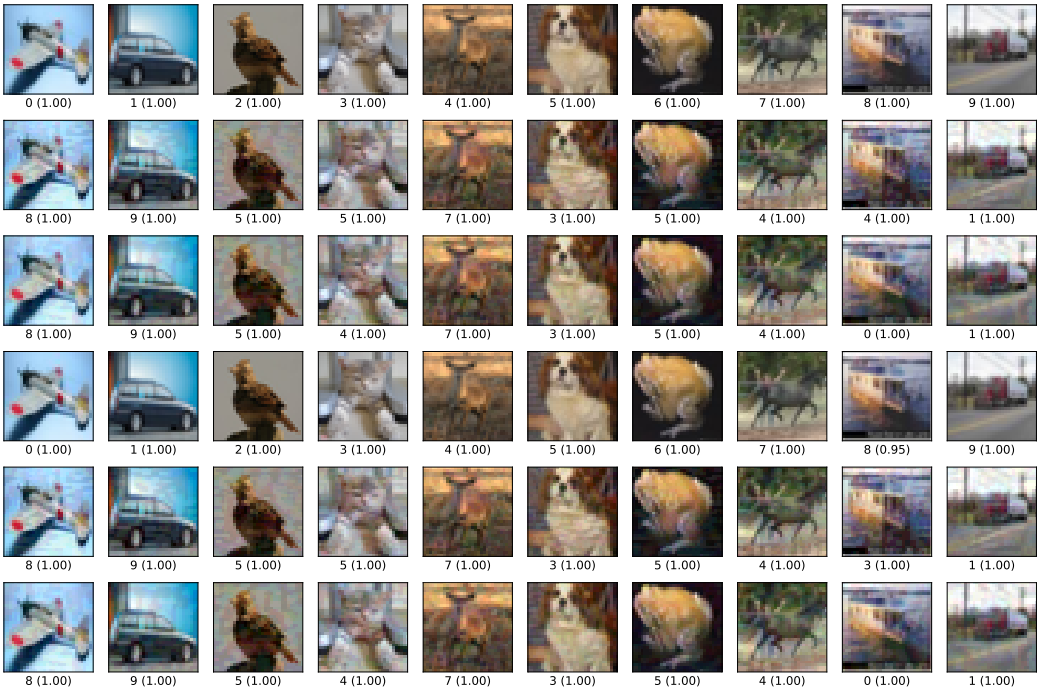


Figure 5.10: Images from the CIFAR dataset and adversarial images created using MI-FGSM classified using neural networks with a temperature of 35

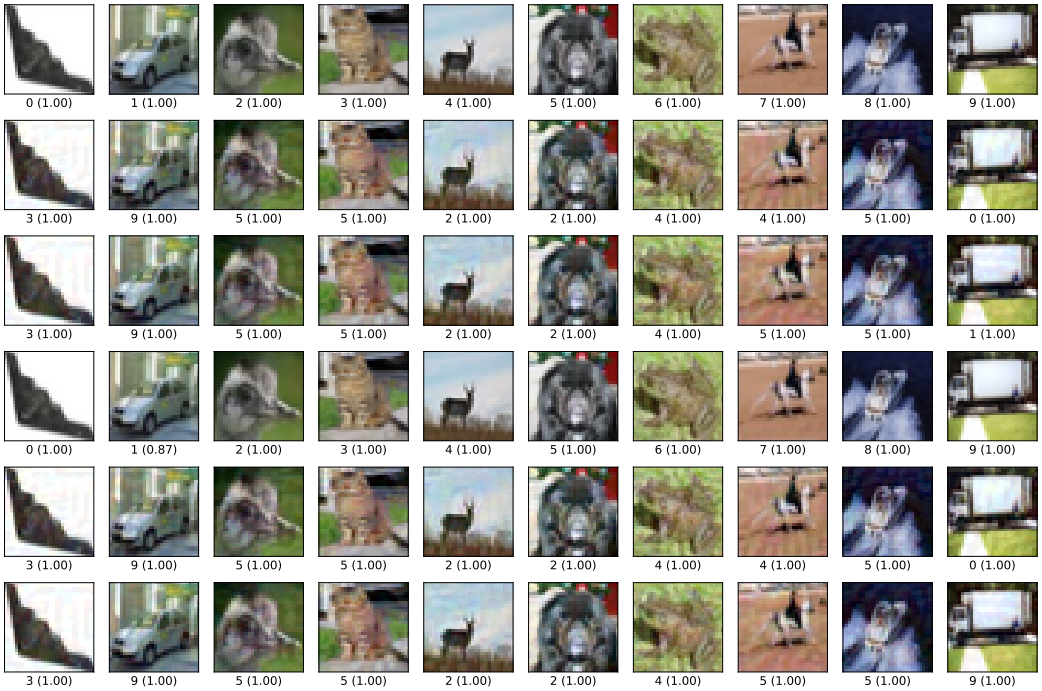


Figure 5.11: Images from the CIFAR dataset and adversarial images created using FGSM classified using neural networks with a temperature of 50

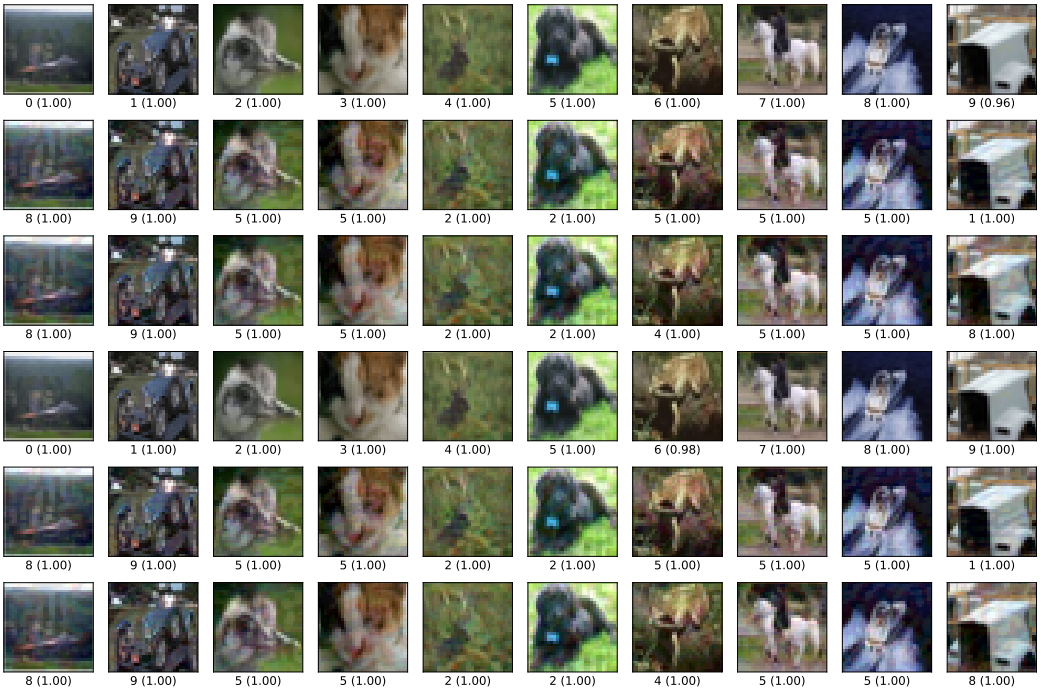


Figure 5.12: Images from the CIFAR dataset and adversarial images created using MI-FGSM classified using neural networks with a temperature of 50

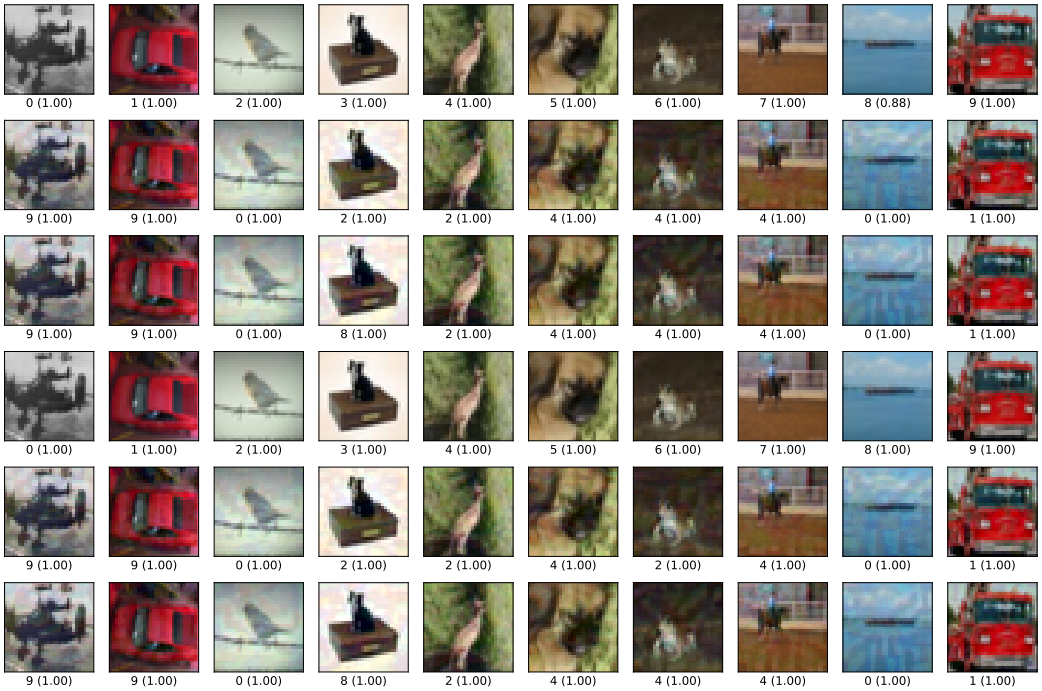


Figure 5.13: Images from the CIFAR dataset and adversarial images created using FGSM classified using neural networks with a temperature of 95

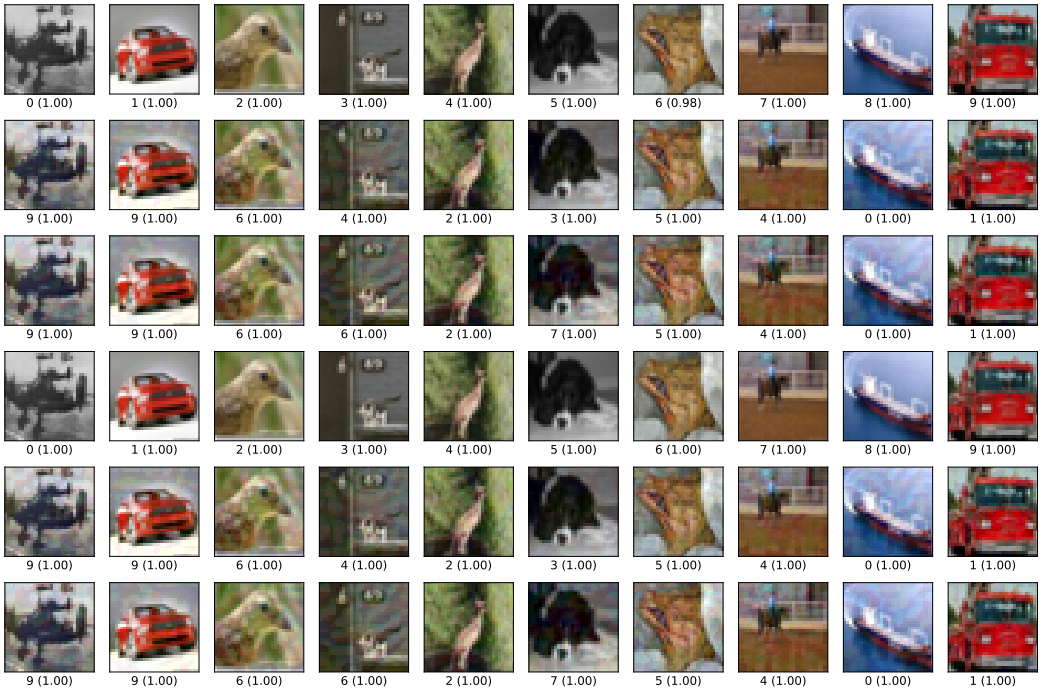


Figure 5.14: Images from the CIFAR dataset and adversarial images created using MI-FGSM classified using neural networks with a temperature of 95

List of Figures

References

7 industries leveraging machine learning. (2019). Retrieved from https://blog.nhlearningsolutions.com/blog/7-industries-leveraging-machine-learning

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... Zheng, X. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems.* Retrieved from https://www.tensorflow.org/ (Software available from tensorflow.org)

Audi uses machine learning to refine self-parking technology. (2019). Retrieved from https://www2.motorauthority.com/news/1107621_audi-uses-machine-learning-to-refine-self-parking-technology

Briefverteilanlage - wikipedia. (2019). Retrieved from https://de.wikipedia.org/wiki/Briefverteilanlage

Chollet, F., et al. (2015). *Keras.* https://keras.io.

Cifar-10 and cifar-100 datasets. (2019). Retrieved from https://www.cs.toronto.edu/~kriz/cifar.html

Deutsche Post - Beförderte Briefe | 2017. (2018). Retrieved from https://de.statista.com/statistik/daten/studie/38996/umfrage/anzahl-der-befoerderten-briefe-durch-die-deutsche-post/

Dong, Y., Liao, F., Pang, T., Hu, X., & Zhu, J. (2017). Discovering adversarial examples with momentum. *CoRR, abs/1710.06081*. Retrieved from http://arxiv.org/abs/1710.06081

Goodfellow, I. J. (2018). Defense against the dark arts: An overview of adversarial example security research and future research directions. *CoRR, abs/1806.04169*. Retrieved from http://arxiv.org/abs/1806.04169

Goodfellow, I. J., Shlens, J., & Szegedy, C. (2014). Explaining and harnessing adversarial examples (2014). *arXiv preprint arXiv:1412.6572*.

Grosse, K., Papernot, N., Manoharan, P., Backes, M., & McDaniel, P. (2017, 1 1). Adversarial examples for malware detection. In S. Foley, D. Gollmann, & E. Snekkenes (Eds.), *Computer security – esorics 2017 - 22nd european symposium on research in computer security, proceedings* (pp. 62–79). Germany: Springer Verlag. doi: 10.1007/978-3-319-66399-9_4

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, *2*(5), 359–366.

Jain, A. K., Mao, J., & Mohiuddin, K. M. (1996). Artificial neural networks: A tutorial. *Computer*, *29*(3), 31–44.

Jia, R., & Liang, P. (2017). Adversarial examples for evaluating reading comprehension systems. *CoRR*, *abs/1707.07328*. Retrieved from http://arxiv.org/abs/1707.07328

Kurakin, A., Goodfellow, I. J., & Bengio, S. (2016). Adversarial machine learning at scale. *CoRR*, *abs/1611.01236*. Retrieved from http://arxiv.org/abs/1611.01236

Mnist handwritten digit database, yann lecun, corinna cortes and chris burges. (2019). Retrieved from http://yann.lecun.com/exdb/mnist/

Moutarde, F., Bargeton, A., Herbin, A., & Chanussot, L. (2009). Modular traffic sign recognition applied to on-vehicle real-time visual detection of american and european speed limit signs. *arXiv preprint arXiv:0910.1295*.

Papernot, N., McDaniel, P. D., Goodfellow, I. J., Jha, S., Celik, Z. B., & Swami, A. (2016). Practical black-box attacks against deep learning systems using adversarial examples. *CoRR*, *abs/1602.02697*. Retrieved from http://arxiv.org/abs/1602.02697

Papernot, N., McDaniel, P. D., Wu, X., Jha, S., & Swami, A. (2015). Distillation as a defense to adversarial perturbations against deep neural networks. *CoRR*, *abs/1511.04508*. Retrieved from http://arxiv.org/abs/1511.04508

Prechelt, L. (1998). Automatic early stopping using cross validation: quantifying the criteria. *Neural Networks*, *11*(4), 761–767.

Sharif, M., Bhagavatula, S., Bauer, L., & Reiter, M. K. (2016). Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 acm sigsac conference on computer and communications security* (pp. 1528–1540).

Sitawarin, C., Bhagoji, A. N., Mosenia, A., Chiang, M., & Mittal, P. (2018). DARTS: deceiving autonomous cars with toxic signs. *CoRR*, *abs/1802.06430*. Retrieved from http://arxiv.org/abs/1802.06430

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. J., & Fergus, R. (2013). Intriguing properties of neural networks. *CoRR*, *abs/1312.6199*. Retrieved from http://arxiv.org/abs/1312.6199

Torralba, A., Fergus, R., & Freeman, W. T. (2008). 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, *30*(11), 1958–1970.

Yuan, X., He, P., Zhu, Q., Bhat, R. R., & Li, X. (2017). Adversarial examples: Attacks and defenses for deep learning. *CoRR*, *abs/1712.07107*.

Retrieved from http://arxiv.org/abs/1712.07107

Declaration of Authorship

I hereby certify that the work presented here is, to the best of my knowledge and belief, original and the result of my own investigations, except as acknowledged, and has not been submitted, either in part or whole, for a degree at this or any other university.

Osnabrück, February 4, 2019