Attacking and Defending a Convolutional Neural Network

Franka van Jaarsveld
University of Twente
Netherlands
f.vanjaarsveld@student.utwente.nl

Agata Sowa
University of Twente
Netherlands
a.m.sowa@student.utwente.nl

Quang-Hung Nguyen
University of Twente
Netherlands
nguyenquanghung@student.utwente.nl

Jakub Nail
University of Twente
Netherlands
j.f.nail@student.utwente.nl

1 INTRODUCTION

In 2014, Szegedy et al. [14] discovered that machine learning classifiers are highly vulnerable to adversarial examples - samples of input data that have been slightly altered. These alterations are so subtle that humans can still easily classify the image, and may not even notice any change. However, they cause machine learning models to misclassify the image, sometimes with surprisingly high confidence. Adversarial examples pose a severe security threat, as they can be used to attack a machine learning algorithm without needing access to the underlying model [10]. Therefore, the defence of classification algorithms against such adversarial attacks is an important field of research.

There are a few 'flavours' of adversarial attacks. First of all, an attack can be targeted or non-targeted. A targeted attack attempts to get the model to misclassify samples as a specific target class, whereas an untargeted attack does not care which label is predicted, as long as it's incorrect. Furthermore, an attack can be implemented in a scenario where everything is known about the model or one where not much is known about it. These scenarios are called 'white-box' and 'black-box', respectively. Finally, an attack can be either digital or physical. This research is geared towards untargeted white-box digital attacks. A few examples of white-box digital attacks are L-BFGS, the Fast Gradient Sign Method (FGSM), and the Basic Iterative Method (BIM) [9]. The same paper by Kurakin et al. [9] describes various defence strategies, such as gradient masking, detecting adversarial examples, and adversarial training.

In this research, a Convolutional Neural Network (CNN) is trained to classify different types of fashion items in the Fashion-MNIST dataset. This trained model is then attacked using two methods that generate adversarial images: the Fast Gradient Sign Method and the Basic Iterative Method. Multiple strategies are utilised to defend the model against these attacks, namely data augmentation, adversarial training, and defensive distillation.

This report aims to describe the process behind this research. Section 2 first gives some theoretical background on the attack and defence strategies, explaining how the attacks work, and why the defences should mitigate their effects. Next, section 3 describes the choice of dataset and network, and how the network is trained and tested. Furthermore, it gives some insight into how the various attack and defence methods have been implemented. Then, the accuracy and loss of the original network are compared to that of the attacked and defended network in section 4. Finally, sections 6 and 7 give a conclusion and suggest further work.

2 THEORY

This section aims to describe the chosen adversarial attack from the context of research papers, and suggested defences to mitigate the effect of the attacks.

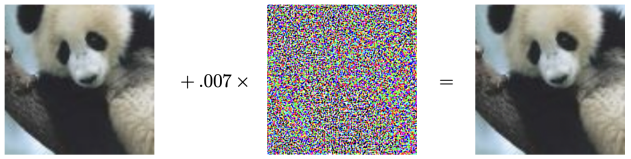
2.1 Attack 1: Fast Gradient Sign Method

The Fast Gradient Sign Method (FGSM), as described in *Explaining and Harnessing Adversarial Examples* by Goodfellow et al. [6], is an attack that uses the gradients of a neural network to maximise loss and generate adversarial samples. FGSM was one of the first and most popular attacks to trick a neural network [2]. Equation 1 shows the mathematics behind FGSM:

$$x^{adv} = x + \epsilon * \text{sign}(\Delta_x J(\theta, x, y)) \tag{1}$$

Where x is the input image, y is the target associated with x, θ is the parameters of the model, and $J(\theta, x, y)$ is the cost function used to train the neural network. The last part of the equation, $\text{sign}(\Delta_x J(\theta, x, y))$, calculates the sign of the gradient of the cost function with respect to x, which is also referred to as noise. This noise is scaled by epsilon ϵ and added to the original image. Changing the value of ϵ allows one to make the attack more or less severe. Figure 1 is a well-known example from Goodfellow et al.'s paper

Figure 1: Demonstration of FGSM adversarial sample generation



[6] that helps explain FGSM further. The left-most image is the original input image, which has been correctly classified as a panda. In the middle, one can see the calculated noise, which is scaled by the epsilon (0.007). The addition of this scaled noise to the original image generates the right-most image. To the human eye, it still

looks like a panda, but it is misclassified as a gibbon by the classifier, with a very high confidence level.

2.2 Attack 2: Basic Iterative Method

The Basic Iterative Method (BIM) was first introduced in *Adversarial Examples in the Physical World* by Kurakin et al. [10]. It is an extension of FGSM and is also referred to as Iterative-FGSM. As this alternative name suggests, BIM applies FGSM multiple times instead of just once. It does this with a small step size and clips pixel values of intermediate results after each step to ensure that they are in an ϵ-neighbourhood of the original image [10]. Equation 2 shows the mathematics behind BIM:

$$\begin{aligned} x_0^{adv} &= x \\ x_{N+1}^{adv} &= Clip_{x,\epsilon}\{x_N^{adv} + \alpha * \text{sign}(\Delta_x J(x^{adv}, y))\} \end{aligned} \tag{2}$$

Where x is the input image, y is the target associated with x, $J(x^{adv}, y)$ is the cost function used to train the neural network, and $Clip_{x,\epsilon}$ is the clipping function. In Kurakin et al.'s implementation [10], α is set to 1, such that the value of each pixel is only changed by 1 on each step. The number of iterations must be chosen such that the adversarial example to able to reach the edge of the ϵ max-norm ball while the computation costs stay manageable.

2.3 Defense 1: Data Augmentation

In general, data augmentation is a widely used method to avoid overfitting in Machine Learning models, especially with imbalanced datasets and/or datasets with limited sizes. First proposed by Krizhevsky et al. in their famous AlexNet paper [8], data augmentation has proven to be a useful way to expand the dataset, which forces the model to learn exact features and migrates the risk of performance loss on novel examples. The augmented data is derived from the original data by performing various geometric transformations (scaling, cropping, flipping, etc.) or colour transformations (brightness, contrast, hue, saturation).

Similarly to Zeng et al.'s research [16], we'll tackle the idea behind the Stochastic Affine Transformation and keep the idea of performing random rotations and random translations, but not use the scaling and adding random horizontal flips (see images 3 and 4). Zeng et al. show that this defence outperforms the state-of-the-art methods of image compression, feature distillation, feature squeezing and pixel deflection against the most common attacks (FGSM, BIM, Carlini& Wagner, LBFGS).

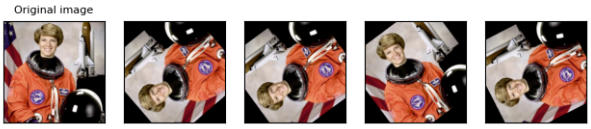
Moreover, there are numerous ways of adding noise to the images. This can be done on different levels. One can either add a random noise layer to the model after a regular layer, which would perturbate the outputs from the previous layer, or perturbate the images directly in the training dataset. Three types of noise perturbations can apply to the dataset, are Gaussian noise (see image 2), salt and pepper noise, and speckle noise.

Through data augmentation, we seek to increase the variability of the data samples in order to avoid over-fitting. This helps mitigate the effects of adversarial attacks, as a lot of them are based on adding specifically designed noise that would lead the model to misclassify samples.

Figure 2: Gaussian blur on images

Figure 3: Random Horizontal flips on images

Figure 4: Random rotations on images



2.4 Defense 2: Adversarial Regularisation

Bui et al. [4] introduced the concept of training neural networks using graphs and structured signals based on the original input, regularising the training and allowing the model to learn the hidden features of an input's neighbours on a graph. In addition to the original cross-entropy loss to measure the model's ability, they introduced "neighbour's loss", which is shown in figure 5 by Tensorflow [1].

In figure 5, the neighbour's loss is defined for Neural Graph Learning. However, it can be modified to equation 3, which is defined as the loss between the label and the neighbour's prediction. In this case, the neighbours are the adversarially perturbed inputs. Neural Structured Learning (NSL) combines raph learning with training on adversarially perturbed images to make data augmentation. Therefore, adversarial regularisation is a method of data augmentation, however, with the use of Neural Graph Learning and the Neural Structured Learning (NSL) package from Tensorflow, the augmented data does not need to be labelled. Goodfellow et al. [6] and Miyato et al. [11] have shown that the robustness of the model can be increased by augmenting the data with adversarial perturbations.

$$\text{neighbour loss} = \sum_{x_j \in N(x_i)} \mathcal{E}(y_i, g_\theta(x_j)) \tag{3}$$

2.5 Defense 3: Defensive Distillation

Hinton et al. [7] introduced the concept of distillation in neural networks. Papernot et al.[12] then implemented it, to create Defensive Distillation, a technique used to improve the robustness of a deep neural network model against adversarial examples. This technique is based on the use of the so-called 'teacher' and 'student' models. The student model uses 'soft' labels for training - the labels that are predicted by the teacher model. This way, it is possible to capture more of the model's internal representation of the data than

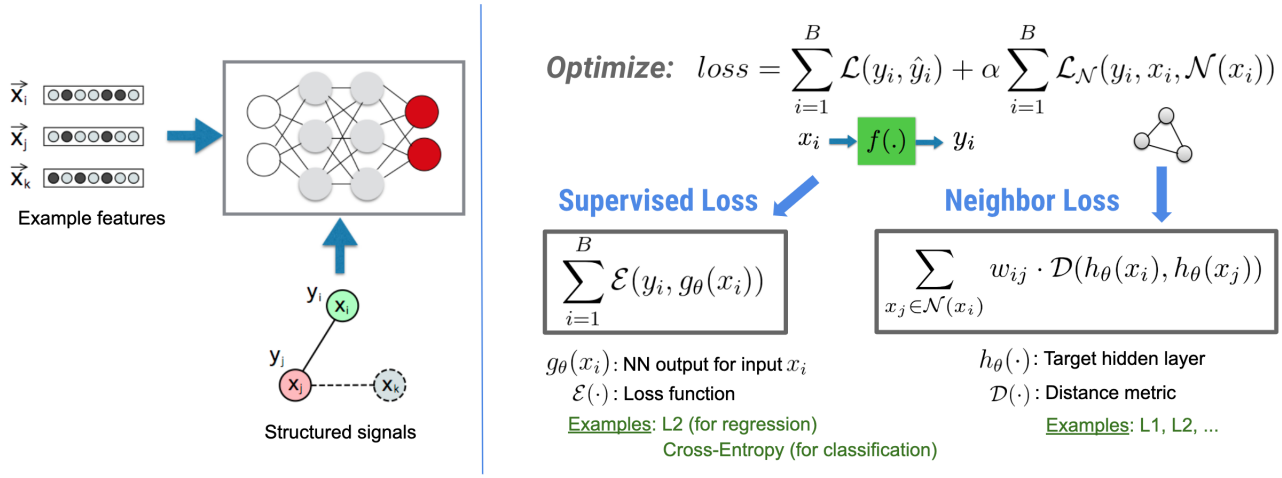


Figure 5: Training using TensorFlow's Neural Structured Learning

with the hard labels (single-class representation). Since the student model is trained on the soft labels, the decision surface is smoothed [5], and the model learns a more robust representation of the data. Papernot et al. [12] state that using Defensive Distillation allows the model to generalise around the training points instead of fitting the data too tightly. Moreover, the architecture of the student and the teacher remains the same, while the student achieves better performance when dealing with adversarial examples. Nevertheless, minor modifications to the architecture of the student are allowed if the speed and accuracy are maintained and the changes have a low impact on the architecture. For example, on the CIFAR10 data set, the adversarial attack success rate was reduced from 87.89% to 5.11%, and the model only experienced a 1.37% drop in accuracy, proving that Defensive Distillation can be used against adversarial examples.

3 IMPLEMENTATION

This section introduces the dataset and neural network used in this research and explains how the initial training was done. Then, it explains how the attacks and defences described in section 2 were implemented.

3.1 Dataset

The dataset used in this research is the Fashion-MNIST dataset, which was first introduced by Xiao et al. [15] in 2017. The Fashion-MNIST dataset was described by the creator as the new benchmark dataset, replacing the old MNIST which has been proven to be too easy to be a benchmark. The dataset contains 70,000 grey-scaled images of fashion products with the training set containing 60,000 images and the test set containing 10,000 images, each with a dimension of 28x28, and divided into 10 categories with equal representation. Figure 6 describes the 10 classes of the dataset and some examples:

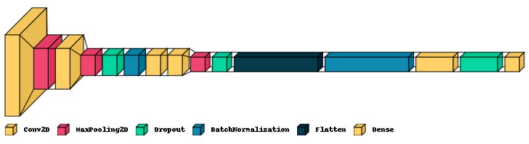
Figure 6: Examples of Fashion-MNIST dataset [15]

Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	

3.2 Network

For the purpose of classifying fashion product types, we decided to use a Convolutional Neural Network. These neural networks excel in spatial data recognition, as they have the ability to develop an internal representation of a two-dimensional image, which allows the model to learn the structures of the data. We based our architecture on the architecture from the Kagle tutorial by Rutvik Deshpande [13] and applied a couple of modifications. A detailed description of the network and its layers can be seen in appendix A.

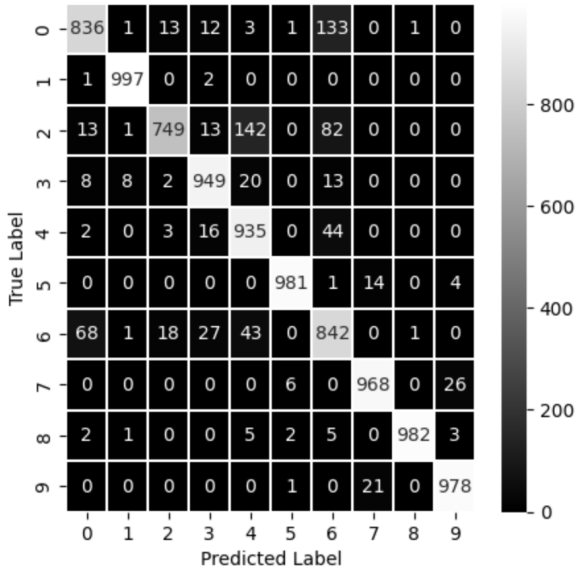
Figure 7: Model architecture



3.3 Initial training

The initial training of the network was done by following a tutorial on Kaggle [13]. This tutorial is a Python implementation using TensorFlow's machine learning module Keras. The accuracy of the model after training is 93.7%, and the confusion matrix in figure 8 shows that the performance is very good, with accuracies ranging from 74% to 99%. Some interesting insights can be gained from the confusion matrix. First of all, the classes that are most easily classifiable are trousers, bags and sandals. The most difficult classes to identify are pullovers, t-shirts and shirts, where pullovers are most often misclassified as coats, and t-shirts and shirts are often misclassified as one another.

Figure 8: Confusion matrix for the initial training

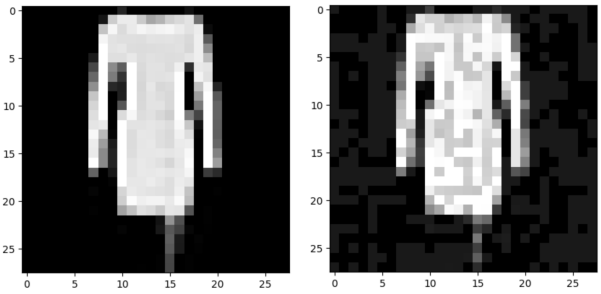


3.4 Attack

3.4.1 FGSM:. The implementation of FGSM was done by following the TensorFlow tutorial on FGSM [3]. The first step is to create perturbations, which will be used to distort the original image to create an adversarial image. These perturbations are created by watching a TensorFlow Gradient Tape and then taking the sign of the gradient of the loss object of the input image and its label. An example image from and dataset, before and after applying FGSM with an epsilon of 0.1, can be seen in figure 9. FGSM was executed on the network using 7 different values of epsilon to analyse the attack's effect. The result is shown and explained in section 4.1.

3.4.2 BIM:. The Basic Iterative Method uses the same code to generate perturbations as the normal FGSM attack (3.4.1). However, the perturbations are applied to the same image multiple times. In total, we performed two analyses. Initially, we took into account the difference in the value of the epsilon, with the number of iterations remaining the same. The number of iterations for that case is 10. Furthermore, we also included the clipping method in the first two analyses. Firstly, we analysed both the epsilon impact and iteration

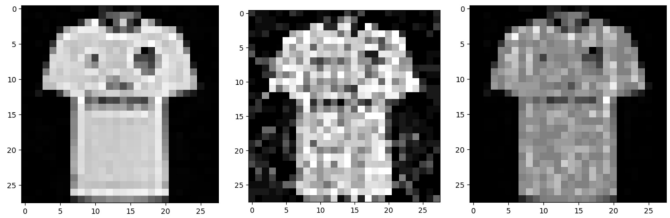
Figure 9: Before and after applying FGSM with ϵ=0.1



impact for the clipping method described in the section about the Basic Iterative Method (2.2). The second clipping method we used was identical to the one used by normal FGSM - the value of the pixels was clipped from 0 to 1. The main difference between those two clipping methods is that for the first one, 0 is out of the clipping range for small epsilons. The examples of those two methods are shown in figure 10. The first clipping method has proven to be more effective. Therefore for the second analysis, we only took the first clipping method into account.

Subsequently, the varying factor is the number of iterations, while the value of the epsilon remains the same. We chose the value of the epsilon after conducting the first analysis. We did it by comparing which value had an impact on the accuracy while the created image was still understandable for a human observer. The chosen epsilon value is 0.05.

Figure 10: Original image, image after applying 10 iterations with ϵ=0.05 with clipping from 0 to 1, and with clipping function

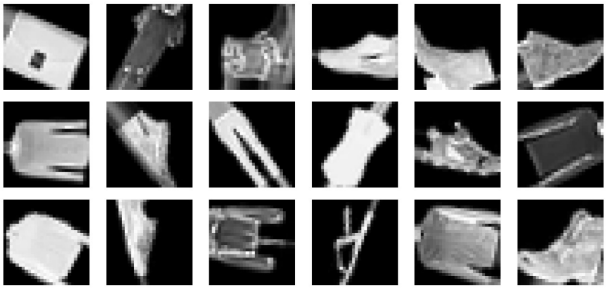


3.5 Defense

3.5.1 Data Augmentation: As described in section 2, a possible defence method is to augment the dataset with more samples. In our case, we utilized six ways of augmenting the data. First, an extension of the original dataset was created by applying a level of Gaussian noise. We used TensorFlow's Gaussian filter with $\sigma = 1$ and a filter size of 3x3. After that, using TensorFlow's `ImageDataGenerator` class, we built a data pipeline to generate different altered images (width and height shift by 0.1, horizontal and vertical flip, and 90° rotation) and fed it to the model randomly every epoch. Image 11 shows an example of this. The augmented data was fed to a new model, using the same architecture described in section 3.2.

Finally, an early-stopping algorithm was deployed using the Keras `EarlyStopping` module, which stops the model if the validation loss does not improve after 5 consecutive iterations. We trained the model using the generated data and then tested it on the unaugmented data.

Figure 11: Augmented data with random transformations and Gaussian blur



3.5.2 Adversarial Regularisation: After training the model with augmented data, we further generalize it by training the model with adversarially generated images. This was done using the `NeuralStructuredLearning` package from TensorFlow, as described in section 2.4. We take into account 50% of the adversarial loss, in addition to the original categorical cross-entropy loss, and the adversarially perturbated data has an ϵ-value of 0.2 and a minimum clipping of 0.25. For the adversarial training process, another early-stopping algorithm was introduced to stop the model if the adversarial loss does not improve after 5 consecutive iterations.

To extend the Adversarial Training, we trained the model four times with different ϵ-values: starting with 0.1, then 0.15, 0.2, and finally 0.25. For each of the ϵ-values, we took into account 10% of the adversarial loss (i.e., the training loss = 90% categorical cross-entropy loss + 10% (scaled) adversarial loss). Theoretically, by training with multiple ϵ-values, the model avoids overfitting on one specific ϵ and generalizes. However, this method does not yield better results compared to the model trained only once, so this approach will not be discussed further.

3.5.3 Defensive Distillation: The basic implementation of Defensive Distillation in the project consists of using the same model architecture for both teacher and the student, with no changes introduced. This model is described in section 3.2. Furthermore, we experimented with minor modifications to the student architecture in order to achieve better performance while still maintaining the speed and accuracy of the initial model. The introduced changes made the student model simpler than the teacher, this can help prevent the student model from overfitting. The detailed architecture of the adjusted student model is in Appendix B.

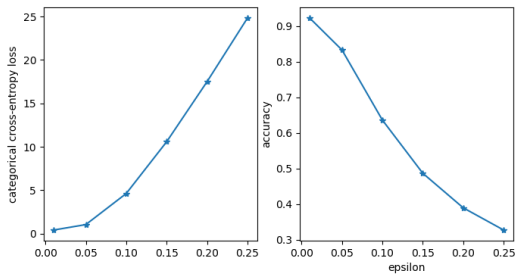
4 RESULTS

This section shows and explains the performance of our model after it has been attacked using FGSM and BIM. This performance is then compared to the performance of the defended network when it is attacked.

4.1 Attacked network

This section shows the performance of the initial network after it is attacked using the Fast Gradient Sign Method and Basic Iterative Method.

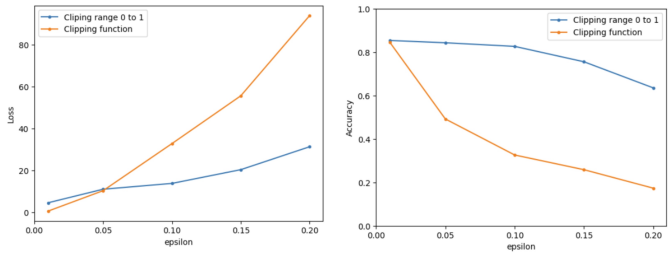
Figure 12: Loss (left) and accuracy (right) of the original model attacked using FGSM



4.1.1 FGSM. The FGSM has proven to be a simple, yet powerful attack on the original model, with the results shown in 12. The accuracy drops from 94% to 63.6% after being attacked. We choose $\epsilon = 0.1$ to show the results since with $\epsilon > 0.1$ the images start to distort and the only noise is left.

4.1.2 BIM. From the figure 13 for accuracy and loss for the initial model, it can be seen that the Basic Iterative Method with clipping range following the equation from section 2.2 is more effective than the one using a simple range from 0 to 1, both in terms of accuracy and loss. Therefore we chose the BIM with clipping function for the second proposed analysis, where the epsilon value is constant and the number of iterations changes. The epsilon value was set to 0.05. Both accuracy and loss are presented in figure 14. The accuracy initially drops rapidly till it reaches a value of 37.48% at 25 iterations, then it remains level while the loss value grows constantly.

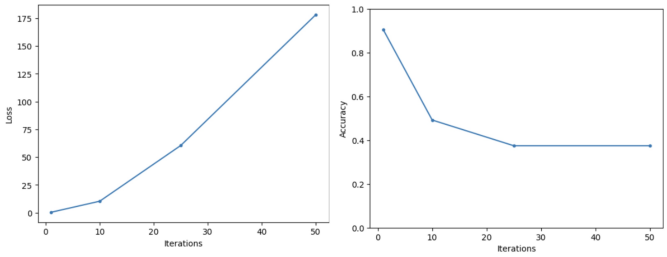
Figure 13: Loss (left) and accuracy (right) for Basic Iterative Method for various epsilon values



4.2 Defended network

To defend the network, we re-train the model (for adversarial training) or train it from scratch (with data augmentation), before it is attacked again. This section will show the results of the defensive methods.

Figure 14: Loss (left) and accuracy (right) for Basic Iterative Method for various number of iterations

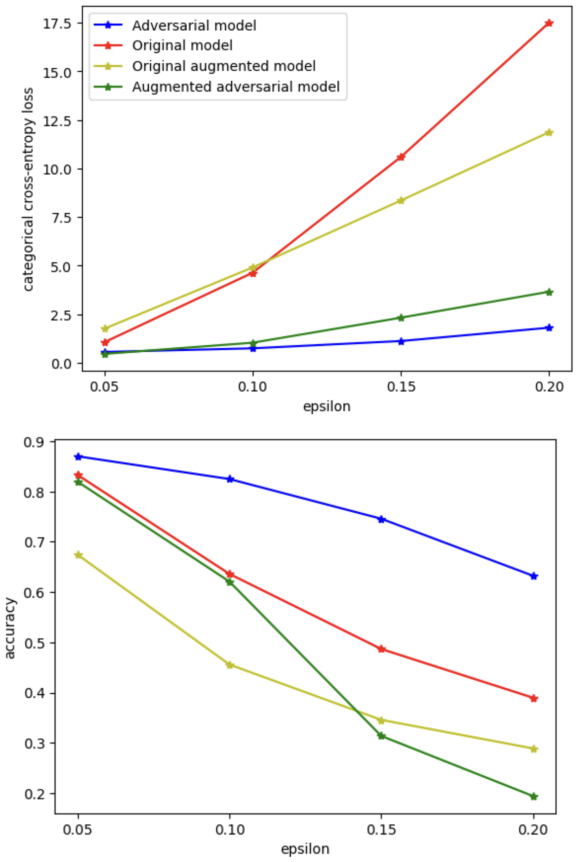


4.2.1 Data Augmentation. The result for the data augmentation method can be seen in figure 15, in yellow. The augmented model (with the method described in section 3.5.1) does worse in terms of accuracy compared to the original model, however with much less categorical cross-entropy loss. We witness a validation accuracy and test accuracy that is much higher (around 0.89) than the training accuracy (around 0.83) meaning that the model is not over-fitting like it did previously. In fact, it rather under-fits the data.

4.2.2 Adversarial Training. The result for adversarial training is also shown in figure 15; there are two additional lines: the blue line is the result of adversarial training for the original model, and the green line is the result of adversarial training for the augmented model. Overall, we witness the same result as with the original model: the augmented adversarial model does much worse in terms of accuracy, however, its loss value is on par with the original model's adversarial training variant. The adversarial training on the original model performs the best: It retains an accuracy of up to 70% for the highest value $\epsilon = 0.2$. We have also analysed the performance of the original and adversarial model for the Basic Iterative Method for both with and without the clipping function. In both cases, the original model performs better than the adversarially trained one in terms of accuracy.

4.2.3 Defensive Distillation. The accuracy of the teacher model is 93.76%, with a loss of 0.2374 on the initial test data. For the student with the same architecture as the teacher, those numbers are 93.78% accuracy with 0.1919 loss. The time it took for one epoch for the teacher was 4.12s and for the student 5.14s. As can be seen from figure 16 the student model always has smaller loss values compared to the teacher. However, the accuracy is only higher for the student in the case of normal FGSM, but the difference between the accuracy of the teacher. For both BIMs, the accuracy of the teacher model is higher.
For the modified student model, the time for one epoch was 6.18s. The loss trend where the loss of the teacher is higher than the loss of the student was preserved. There were, however, significant changes in the accuracy. For the BIM with clipping function, the accuracy of the student was higher than the accuracy of the teacher till the epsilon value of 0.1. And the discrepancy between those two values was smaller than in the case of the student identical to the teacher. Moreover, in the case of FGSM, the loss of the changed student is smaller and its accuracy is higher.

Figure 15: Loss (top) and accuracy (bottom) for the original model and three other defence methods against FGSM



5 DISCUSSION

During the execution of attacks and defence, we noticed that the initially trained model was overfitting. This means that from a certain point of training, the model also starts learning the noise on the image instead of the features. This led to the overfitting problem on the student model as well. As a result, all the models were performing worse under attack and were less responsive to the defences. Therefore, for enhanced performance, it is essential to stop the training of the model before it starts learning the noise - early stopping. Below we'll be able to present some of the results, but this time conducted for the non-overfitted model. As can be seen in figure 17 the not overfitted model performs significantly better than the overfitted one.

The discovery of overfitting also results in another defensive method: model regularization. Unfortunately, we did not have time to properly conduct research into this method. However, initial results are promising: The regularized model is less sensitive to attacks compared to the original model.
With the adversarial training method using Neural Graph Learning, we observed robust performance when dealing against FGSM attack. However, there is one problem that have not been considered:

Figure 16: Loss (left) and accuracy (right) at multiple epsilon values when attacking the student and teacher model using FGSM, BIM with 0 to 1 clipping, and BIM with clipping function

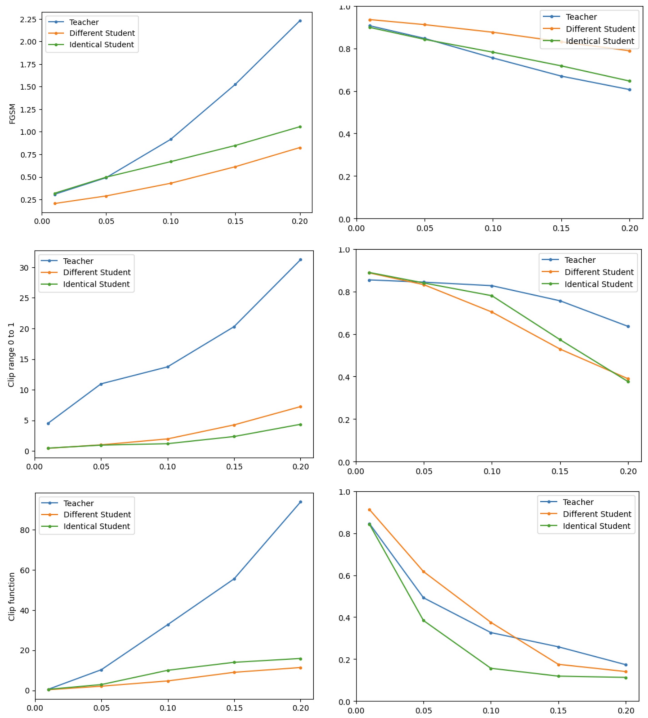
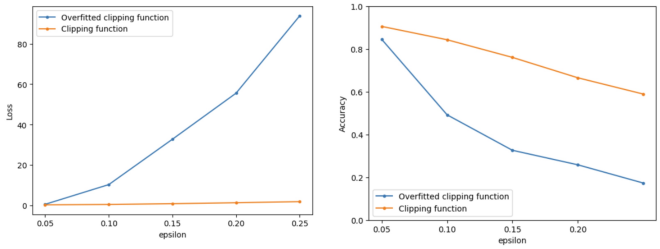


Figure 17: Loss (left) and accuracy (right) at multiple epsilon values for FGSM for overfitted and non-overfitted model



overfitting on the perturbated image set. This is a hypothesis for now, and we will discuss further in section 7 on how the future research might discover and handle the overfit problem.

On the other hand, we observe the opposite phenomenon when training the model on augmented data. The training accuracy drops significantly in favour of the test accuracy, a sign of under-fitting. We believe that this is due to a too-restrictive architecture of our convolutional network being unable to detect enough valuable features from the perturbated data. It is also worth mentioning that the loss is lower on the model trained on augmented data than the loss of the model trained on the clear dataset, but its accuracy is worse than its counterpart. This shows that the augmented model is less confident in making predictions. A more elaborated architecture of a wider and deeper neural network seemingly helps to handle both these issues.

Another point worth mentioning is the effect of the clipping function used in BIM on the image. The colours in the images are coded as floats from zero the one, where zero represents black and one white. In the normal FGSM, the clipping range is from zero to one, which allows the perturbations to occur on the whole image, including the background, which is black (zero). The clipping method introduced by BIM for the small epsilons with a small number of iterations often has a lower clipping boundary which is greater than zero. Because of that, the perturbations only occur on the pixels that are part of the clothing item. This makes the perturbated image harder to spot for a human observer since there is no noise in the background.

In the defensive distillation, all the losses for students in any student-teacher configuration are lower than the ones for the teachers. At the same time, the accuracy for the student is not always higher than the accuracy of the teacher model. This results from the fact that the student uses soft labels for the training. Therefore, it might result from the student model being less confident than the teacher one.

On another note, we've noticed that there was a distinctive misclassification issue between the two classes of labels shirt and t-shirt on every model we've trained for the purpose of this project (see confusion matrix on figure 8, for example). When one thinks about these two objects the boundaries between them are relative, even humans are entitled to assign the opposite label. This mistake might not necessarily be the result of the impossibility of distinguishing the object but the fact that a shirt for someone can be a T-shirt for someone else, which would also justify this issue on the level of the model. This also suggests that a targeted attack on one of these two classes could turn out to be particularly effective.

6 CONCLUSION

The goal of this research was to analyse various adversarial attack and defence methods on a Convolutional Neural Network used to classify images from the fashion-MNIST data set. The initial training resulted in an accuracy of 93.7%. The chosen attacks are the Fast Gradient Sign Method and the Basic Iterative Method.

Both attacks resulted in a drop in accuracy and an increase in loss in the undefended network. To mitigate the effects of these attacks, three defence strategies were applied: data augmentation, adversarial training, and defensive distillation. The results of these strategies were very varied. Data augmentation did not do much to defend the network against any attacks. This is likely because the network is unable to detect enough valuable features, but a different, more elaborate network may solve this. Defensive distillation had a more promising result, as the loss of the student network was much lower than the original (teacher) network. However, the accuracy still dropped quite a bit when applying this defence. One possible explanation for this is that the original model was overfitted. This is an important takeaway for any future research, as it gives rise to two possible improvements: early stopping and regularisation.

Finally, adversarial training helped mitigate the effects of FGSM quite well, giving us the best results for accuracy and loss.
From this research, adversarial training has emerged as the strongest method with which to defend a classification algorithm against adversarial attacks. However, there are many improvements to be made with regard to the network architecture, the training of the original model, and model regularisation.

7 SUGGESTED IMPROVEMENTS AND FURTHER WORKS

The main thing that could be improve it the regularization of the model. As mentioned on section 5, we have discovered that by adding kernel regularizer for weight decay, and early-stopping mechanism based on validation loss, the model generalizes better against novel perturbated images. Therefore, further experiments on model regularization and implement it before trying other defence methods would improve the accuracy.

In our reserach, we have attempted three different defence method: Data Augmentation, Adversarial Training using Neural Graph Learning, and Defence distillation. The Adversarial training was one of the most successful initially, however as discussed in section 5, further experimemt could focus on dealing with the overfitting problem in a step-by-step method: (1) Train a more robust model that generalizes better; (2) Mix original/augmented training data with the perturbated training data in the adversarial training process, and (3) experiment with the `multiplications` hyperparameter of the `NeuralStructuredLearning` package to see which multiplications of the scaled adversarial loss is the most balance between robustness and generalization. For the augmentation method, the only reasonable next step is to try with a more regularized and deeper architecture, to increase the expressiveness of the model, allowing it to capture more features of the training data while not overfitting. At that point, data augmentation would make more sense. The Defensive Distillation also performed better than the original model in terms of loss but was not always successful in outperforming the model in terms of accuracy. Having the teacher model not be overfitted could improve the performance of that method. Moreover, the student architecture could be further changed, as we only analyzed a limited number of combinations, in order to find a more robust combination of the two models.

REFERENCES

[1] 2021. The Neural Structured Learning Framework. https://www.tensorflow.org/neural_structured_learning/framework

[2] 2022. Adversarial example using FGSM. https://www.tensorflow.org/tutorials/generative/adversarial_fgsm

[3] 2022. Adversarial regularization for image classification. https://www.tensorflow.org/neural_structured_learning/tutorials/adversarial_keras_cnn_mnist

[4] Thang D. Bui, Sujith Ravi, and Vivek Ramavajjala. 2017. Neural Graph Machines: Learning Neural Networks Using Graphs. *CoRR* abs/1703.04818 (2017). arXiv:1703.04818 http://arxiv.org/abs/1703.04818

[5] Ian Goodfellow, Patrick McDaniel, and Nicolas Papernot. 2018. Making machine learning robust against adversarial inputs. *Commun. ACM* 61, 7 (2018), 56–66.

[6] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. EXPLAINING AND HARNESSING ADVERSARIAL EXAMPLES. *ICLR* (2015). https://github.com/lisa-lab/pylearn2/tree/master/pylearn2/scripts/

[7] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet Classification with Deep Convolutional Neural Networks. 60, 6 (2017). https://doi.org/10.1145/3065386

[9] Alexey Kurakin, Ian Goodfellow, Samy Bengio, Yinpeng Dong, Fangzhou Liao, Ming Liang, Tianyu Pang, Jun Zhu, Xiaolin Hu, Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, Alan Yuille, Sangxia Huang, Yao Zhao, Yuzhe Zhao, Zhonglin Han, Junjiajia Long, Yerkebulan Berdibekov, Takuya Akiba, Seiya Tokui, Motoki Abe, Samy Bengio Google Brain Yinpeng Dong, Jianyu Wang Baidu Research USA Zhou Ren, Sangxia Huang Sony Mobile Communications, Sweden Yao Zhao Microsoft corp Yuzhe Zhao, and Yerkebulan Berdibekov Independent Scholar Takuya Akiba. 2018. Adversarial Attacks and Defences Competition. (2018).

[10] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. 2017. Adversarial Examples in the Physical World. *ICLR* (2017). https://arxiv.org/pdf/1607.02533.pdf

[11] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. 2017. Virtual Adversarial Training: A Regularization Method for Supervised and Semi-Supervised Learning. https://doi.org/10.48550/ARXIV.1704.03976

[12] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2016. Distillation as a defense to adversarial perturbations against deep neural networks. (2016), 582–597.

[13] Deshpande Rutvik. 2021. Fashion MNIST / CNN Beginner (98%). https://www.kaggle.com/code/rutvikdeshpande/fashion-mnist-cnn-beginner-98

[14] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. *ICLR* (2014).

[15] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. https://doi.org/10.48550/ARXIV.1708.07747

[16] Yi Zeng, Han Qiu, Gérard Memmi, and Meikang Qiu. 2020. A Data Augmentation-based Defense Method Against Adversarial Attacks in Neural Networks. *CoRR* abs/2007.15290 (2020). https://arxiv.org/abs/2007.15290

A DETAILED DESCRIPTION OF THE CONVOLUTIONAL NEURAL NETWORK

- A 2D Convolutional layer consisting of:
  - 32 kernels (filters)
  - 3x3 kernel size
  - kernel initializer HE Normal
  - same padding
  - ReLu activation function
- A (2,2) MaxPooling2D:
- A 2D Convolutional layer consisting of:
  - 64 kernels (filters)
  - 3x3 kernel size
  - same padding
  - ReLu activation function
- A (2,2) MaxPooling2D:
- A dropout layer with frequency 0.3
- A batch normalization layer without specific parameters
- Two 2D Convolutional layers consisting of:
  - 128 kernels (filters)
  - 3x3 kernel size
  - same padding
  - ReLu activation function
- A (2,2) MaxPooling2D:
- A dropout layer with frequency 0.4
- A flattening layer without parameters
- A batch normalization layer without specific parameters
- A dense layer consisting of:
  - 512 units (the dimensionality of the output space)
  - ReLu activation function
- A dropout layer with frequency 0.25
- A final dense layer consisting of:
  - 10 units (the total number of classes)
  - SoftMax activation function

B DETAILED DESCRIPTION OF THE STUDENT NETWORK

- A 2D Convolutional layer consisting of:
  - 32 kernels (filters)
  - 3x3 kernel size
  - kernel initializer HE Normal
  - same padding
  - ReLu activation function
- A batch normalization layer without specific parameters
- A 2D Convolutional layer consisting of:
  - 32 kernels (filters)
  - 3x3 kernel size
  - kernel initializer HE Normal
  - same padding
  - ReLu activation function
- A batch normalization layer without specific parameters
- A (2,2) MaxPooling2D:
- A dropout layer with frequency 0.5
- A 2D Convolutional layer consisting of:
  - 64 kernels (filters)
  - 3x3 kernel size
  - same padding
  - ReLu activation function
- A batch normalization layer without specific parameters
- A 2D Convolutional layer consisting of:
  - 64 kernels (filters)
  - 3x3 kernel size
  - same padding
  - ReLu activation function
- A batch normalization layer without specific parameters
- A (2,2) MaxPooling2D:
- A dropout layer with frequency 0.5
- A flattening layer without parameters
- A dense layer consisting of:
  - 512 units (the dimensionality of the output space)
  - ReLu activation function
- A dropout layer with frequency 0.5
- A final dense layer consisting of:
  - 10 units (the total number of classes)
  - SoftMax activation function