Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Halvor Mundal

Why are neural networks vulnerable to adversarial examples?

Master's thesis in Computer Science
Supervisor: Jingyue Li
June 2020

NTNU
Norwegian University of Science and Technology

Halvor Mundal

Why are neural networks vulnerable to adversarial examples?

Master's thesis in Computer Science
Supervisor: Jingyue Li
June 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Abstract

Neural networks have in the last decade shown to have excellent performance in a wide range of tasks and even outperform humans in some areas. However, they have also shown to be vulnerable to small, imperceptible perturbations in the input, called adversarial examples, causing them to miss-classify instances they would otherwise have classified correctly. As neural networks are being applied a wide specter of solutions, many requiring complete trust in their predictions, it is crucial that we find a way to defend against adversarial examples. Although much research is done on adversarial examples, and some hypotheses have been presented, it still remains a mystery why they are able to fool the neural networks. This thesis aims to explain why neural networks are vulnerable to adversarial examples and how to negate them. I divide the different hypotheses about adversarial examples from the literature into four research questions and investigate the research questions by measuring the robustness of neural networks with different hyperparameters and with adversarial examples as training input. I show that the neural networks' vulnerability to adversarial examples most likely is caused by the neural networks having the decision boundary too close to the training examples. Additionally, the results show that the defensive method of Madry et al. (2017) is robust to any possible adversarial attacks under certain conditions.

Sammendrag

Nevrale nettverk har det siste tiåret vist seg å ha utmerket ytelse i et bredt spekter av oppgaver, og kan til og med utkonkurrere mennesker på enkelte områder. Imidlertid har de også vist seg å være sårbare for små, umerkelige forandringer i inndataen, kalt motstandereksempler(adversarial examples), som får de nevrale nettverkene til å feilklassifisere eksempler de normalt ville klassfisert korrekt. Siden nevrale nettverk blir brukt i et bredt spekter av løsninger som krever full tillit til deres prediksjoner, er det avgjørende at vi finner en måte å forsvare oss mot motstandereksemplene. Selv om det forskes mye på motstandereksempler, og enkelte hypoteser er blitt presentert, er det fortsatt et mysterium hvorfor de klarer å lure de nevrale nettene. Denne masteroppgaven har som mål å forklare hvorfor nevrale nettverk er sårbar mot motstandereksempler og hvordan man kan unngå dem. Jeg inndeler hypotesene om motstandereksempler fra litteraturen i fire forskningsspørsmål, og unersøker forskningsspørsmålene ved å måle robustheten til nevrale nettverk med forskjellige hyperparametre og med motstandereksempler som treningsdata. Jeg viser at den mest sannsylige hypotesen for hvorfor neverale nettverkene er sårbare for motstandereksempler er at de nevrale nettverhene har beslutningsgrensen(decision boundary) for nær treningseksemplene. I tillegg viser resultatene at forsvaret til Madry et al. (2017) er robust mot alle mulige motstanderangrep under enkelte forrhold.

Preface

This Masters thesis was written for the Department of Computer Science at NTNU, and was supervised by Jingyue Li. The thesis builds upon an preliminary term paper, which was written in the autumn of 2019. I would like to thank Jingyue Li and Jin Zhang, who have provided invaluable help when I was writing this master thesis.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Neural networks

Neural networks are a type of machine learning that in the last decade have given us artificial intelligence systems with exceptionally high performance. They are very versatile and are applied to a wide range of tasks such as detecting particles at CERN (Ciodaro et al. (2012)), playing the game Go (Silver et al. (2017)) and classifying skin cancer (Esteva et al. (2017)). Their versatility and ability to perform just as good as humans, or even better as shown by He et al. (2015b) and Taigman et al. (2014), make them very appealing and push their appliance into new fields. Many of those fields are security-critical, such as self-driving cars (Bojarski et al. (2016)) and unmanned aircrafts (Julian et al. (2016)).

1.2 Adversarial examples

Recent research has shown that neural networks are vulnerable to adversarial examples. Szegedy et al. (2013) showed that perturbations in the input that are too small for humans to recognize can fool the networks and change their predictions to something completely wrong. These perturbated examples, called adversarial examples, make the neural networks vulnerable to malicious attacks that humans cannot detect. Liu et al. (2016) showed that these adversarial examples were transferable between different neural networks, making it possible to perform black-box attacks where the attacker has no knowledge of the underlying system. Eykholt et al. (2017) and Kurakin et al. (2016a) showed that it was possible to create adversarial examples in the real world. This indicates that adversarial examples are not just carefully selected noise maximized to fool a model, but true shortcomings of neural networks that can be repeated over multiple models and in the real world. If we want to use neural networks for security-critical use, it is essential that we can trust them, which is impossible as long as adversarial examples can fool the networks.

Negating adversarial attacks

Many defenses against adversarial attacks have been created, such as defensive distillation by Papernot et al. (2015b) or training on adversarial examples by Shaham et al. (2018). However, these two defensive methods were defeated by Carlini and Wagner (2017) and Madry et al.

(2017) respectively. Although new defensive methods are being developed to combat new attacks, it is apparent that it is not enough to prove that a defensive method holds against known attacks to guarantee robustness. Other researchers such as Kolter and Wong (2017) and Raghunathan et al. (2018) train neural networks that can be proven to be robust within an L_p ball, but these methods are computationally exhausting and not likely to scale to bigger problems.

1.3 Causes of adversarial examples

We do not know why neural networks are vulnerable to adversarial examples. Although several hypotheses about adversarial examples are proposed, we do not know which of them, if any, are correct. No research systematically tests the hypotheses, and thus it is very difficult to create neural networks robust against adversarial examples, as the premise of a robust neural network is uncertain. It seems reasonable that before we can defend against adversarial examples, we must understand why neural networks are vulnerable to them.

1.4 Testing the hypotheses

This paper aims to test hypotheses about NN and adversarial examples found in the literature. To do this, I categorize the hypotheses into four categories; 1) neural networks are vulnerable to adversarial examples because they are too non-linear, 2) neural networks are vulnerable to adversarial examples because they are too linear, 3) neural networks are vulnerable to adversarial examples because they overfit, and 4) neural networks are vulnerable to adversarial examples because the decision boundary of the neural networks is too close to the data samples. The first three categories of hypotheses are tested by systematically changing the hyperparameters and measuring how they affect the neural networks' robustness to adversarial examples. The last category of hypotheses is tested by training on adversarial examples. By doing this, I show that the decision boundary is the most factor to influence the robustness of neural networks and that the other factors investigated in this study are unlikely to be relevant to the robustness of neural networks. In my experiments, I also show that the defensive method of Madry et al. (2017) creates neural networks robust to any attack under certain conditions.

Chapter 2

Background

Parts of this chapter is based on a term paper written in the autumn of 2019.

2.1 Neural networks

Neural networks are machine learning algorithms designed to learn and solve tasks, which have shown to perform exceptionally well. The idea behind neural networks is to mimic the biological neurons found in the brain, as described by Rosenblatt (1958). The artificial neurons are represented by nodes that lie in layers, where each node in a layer is connected to all neurons in the next layer by an edge. When the node receives a signal, represented by a number, it is sent along all edges. In the first layer, the input signal is a data sample from the task we want to learn. Each edge contains a weight, which is a number that the signal is multiplied with. In the next layer, each neuron sums up all the individual signals multiplied by the different weights connected to the neuron plus a bias. The sum is then applied to a non-linear function, called the activation function, and sent forward as a signal to the next layer. Commonly used activation functions are the sigmoid function, shown in equation 2.1, and the relu function, shown in equation 2.2.

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

$$relu(x) = max(0, x) \tag{2.2}$$

At the last layer, the softmax function is commonly used, which gives a probability estimate for each class given the input sample. The softmax function for the i-th output in a network with j output nodes is shown in equation 2.3.

$$softmax(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j)} \tag{2.3}$$

To train the neural networks, the difference between the output, y, and the desired output values for a data instance, y', is calculated using a loss function. Common loss functions are the cross-entropy and the mean square error (MSE), seen in equation 2.4 and 2.5. The gradients of the loss function w.r.t. the weights and biases are then calculated, and the weights and biases are updated in the gradients' negative direction to minimize the loss. The size of the update is

controlled by a variable called the learning rate, which is multiplied with the gradients before they are applied to the weights and biases. This is done multiple times with multiple data instances until the neural network converges at a loss value.

$$crossentropy(y, y') = -\sum_{i=1}^{n} y_i' log_2(y_i) \tag{2.4}$$

$$MSE(y, y') = \frac{1}{n}\sum_{i=1}^{n}(y_i' - y_i)^2 \tag{2.5}$$

Neural networks can be seen as a way to approximate functions that solve specific problems and have in recent years been applied to an increasingly wide range of tasks. Helmstaedter et al. (2013) used neural networks to reconstruct neural circuits in the mouse retina, Ciodaro et al. (2012) used neural networks for detecting particles at CERN, Silver et al. (2017) used neural networks to create a program that defeated the world champion in the game of Go and Esteva et al. (2017) used neural networks to create Dermatologist-level classification of skin cancer. Clearly, neural networks have been implemented in a wide range of fields. In addition to being applied to a broad range of tasks, neural networks have also shown to perform exceptionally well, sometimes even better than humans, as shown by He et al. (2015b) and Taigman et al. (2014). As neural networks have gained popularity and larger neural nets are being used, depth has been preferred to width as wide networks seem to memorize the samples more than generalize over them, causing the neural networks to overfit. Deeper neural networks have shown to generalize well since they can learn features at various levels of abstraction in each layer.

2.1.1 Convolutional neural networks

Convolutional neural networks (CNN) (Lecun et al. (1998)) is a type of neural network primarily used for image classification, which is better at detecting patterns in the images. In a convolutional neural network, the signal goes through a feature mapping before the activation function. In the feature mapping, pixels close to each other are multiplied with a matrix, called the filter or the kernel, and then summed together. This is shown in figure 2.1. By using multiple filters, we get different feature maps that can learn different patterns in the images. In the case of multiple channels such as in an RGB image or when we have multiple feature maps, three-dimensional kernel arrays are used. The kernel size is defined as the size of the two dimensions which are not affected by the number of channels.

2.2 Adversarial examples

Although neural networks have shown to have superhuman performance, they also become easy victims of malicious attacks. By slightly distorting the data input, researchers have been able to make state of the art neural networks produce incorrect outputs while still being highly confident in the results. Szegedy et al. (2013) were the first to describe this counter-intuitive property of neural networks. Before this paper, it was assumed that neural networks generalized well locally. Researchers already knew that neural networks have a superior ability to generalize non-locally to examples far away from the training examples, as long as they have some of the same

[1]Convolutional Neural Network. Brilliant.org. Retrieved 12:38, April 28, 2020, from https://brilliant.org/wiki/convolutional-neural-network/

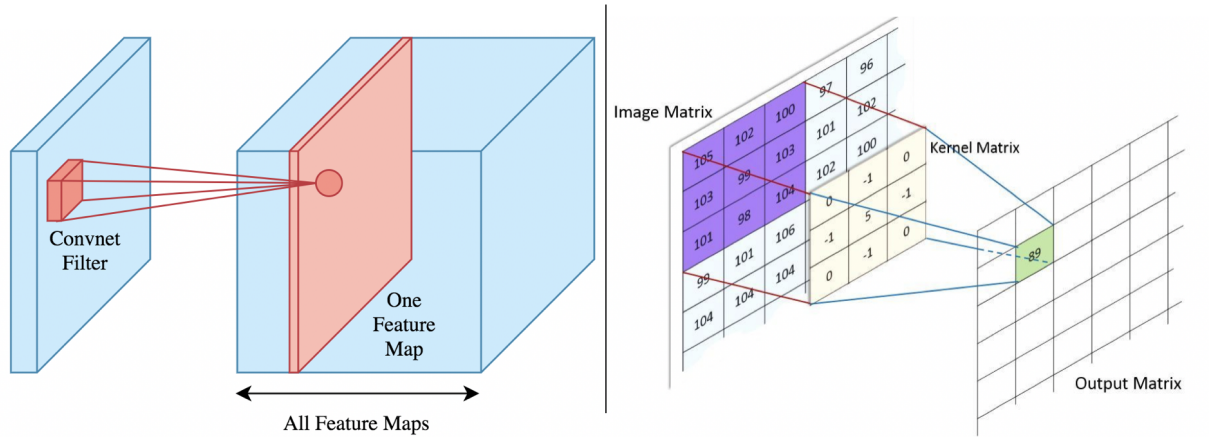


Figure 2.1: Image visualizing neural network convolution. To the left, we see how multiple pixels correspond to a single pixel in one feature map. To the right, the calculation done by the feature mapping shown. The images are from brilliant.org[1].

structural properties. Because of this, it was believed they also generalized to examples in close vicinity to the training examples. However, Szegedy et al. (2013) showed that this assumption does not hold and that very small perturbations are able to alter a neural network's classification of an example. They named these slightly distorted examples "adversarial examples." In figure 2.2, we see adversarial noise added to an image of a panda. While the original example and the adversarial example look identical to human eyes, it was able to change the prediction given by the neural network "GoogLeNet" of Szegedy et al. (2014a) from a panda to a gibbon.

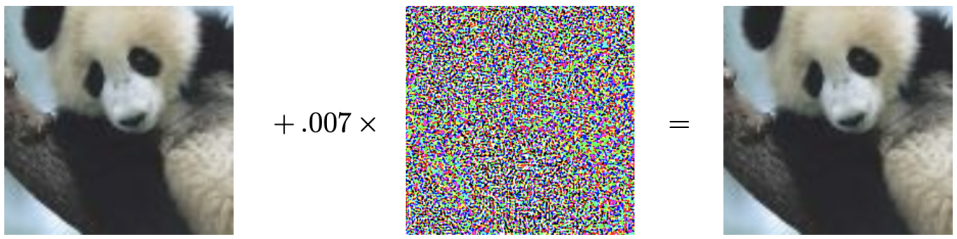


Figure 2.2: A demonstration of an adversarial example from Goodfellow et al. (2014). By applying equation 3.2 on an image of a panda, they are able to alter the prediction of GoogLeNet from "panda" to "gibbon" while keeping the difference between the original and the altered image imperceptible to human eyes.

Adversarial examples also have other interesting properties, such as transferability and real-world applicability. Although the strongest attacks create noise that is tailored to maximize the error for a specific model, we see that there are other ways to create adversarial examples. Liu et al. (2016) show that it is possible to create adversarial examples that generalize to multiple neural networks which they were not made for, making it possible to fool a model without having direct access to it. Athalye et al. (2017) show that simple augmentation, such as re-scaling, translation, and rotation, is able to produce adversarial examples. They also print adversary 3D models that are adversary over various angles, showing that it is possible to create real-world

physical adversarial examples. Eykholt et al. (2017) and Kurakin et al. (2016a) also show that real-world adversarial examples are possible with other methods such as creating subtle stickers that alter a model's prediction.

2.2.1 Why are neural networks susceptible to adversarial examples?

While it is relatively easy to show that neural networks are susceptible to adversarial examples, we still do not know why this behavior occurs. Although there are multiple hypotheses about adversarial examples, it is difficult to reach a conclusion to which hypothesis is correct and to why adversarial examples exits. Szegedy et al. (2013) hypothesize that the adversarial examples consist of high-dimensional "pockets" with values rarely observed in the data sets that still are close to the samples, much like rational numbers have decimal numbers that are rarely used, but close to the often used integers. Gu and Rigazio (2014) show that the sizes of these "blind-spots" are rather large and relatively continuous.

With the hypothesis that high-dimensional pockets of rarely seen values cause adversarial examples, it is implicit that the high non-linearity of neural networks makes them vulnerable to adversarial examples, according to Goodfellow et al. (2014). They argue that "Previous explanations for adversarial examples invoked hypothesized properties of neural networks, such as their supposed highly non-linear nature." They deny that adversarial examples are caused by high non-linearity, but rather too much linearity in the neural network. Although the activation functions are supposed to provide non-linearity, the relu function, which is the de facto standard activation function for neural networks, is linear for all inputs over zero. Other, more non-linear activation functions such as sigmoid, are usually tuned to spend the most time in their more linear parts of the function to avoid saturation. They argue this linearity makes the neural networks vulnerable to attacks that maximize their activations over multiple dimensions.

Tanay and Griffin (2016) disagree with the linear hypothesis of Goodfellow et al. (2014). They argue that if the linearity hypothesis is true, then increasing the resolution of the images should let the attacks maximize their activations over more dimensions, and thus decrease the neural networks' robustness. They show that increasing the resolution of the images does not make the neural networks more vulnerable to adversarial examples, and present the "boundary tilting perspective" hypothesis. This hypothesis states that the decision boundary lies in such a way that the training data is classified correctly, but with the boundary tilted close to the training data in some places. They believe the models are overfitted on components with low variance in the data set and that adversarial examples occur when we change these components in the directions of the low variance. Although Tanay and Griffin (2016) reject the theory of Goodfellow et al. (2014) that adversarial examples are caused by linearity, Goodfellow et al. (2014) use the linearity hypothesis to create the state of the art Fast Gradient Sign Method (FGSM) attack.

Rozsa et al. (2016) argue that adversarial examples stem from what they call evolutionary stalling. When the network is training, the weights are gradually changed until they classify the samples correctly. When the samples are correctly classified, they no longer contribute to the change in the weights. Although the samples are correctly classified, we also need broad flat regions around them to prevent adversarial examples. They argue that since the samples' contribution to the weights stops once they are classified correctly, the samples are left close to the decision boundary, making the neural networks susceptible to adversarial attacks. Madry et al. (2017) also believe the decision boundary is left too close to the samples. They create a defensive technique that trains on adversarial examples to push the boundary further away from

the samples.

In summary, we can divide the different hypotheses for the existence of adversarial examples in four categories: 1) Adversarial examples stem from "pockets" of low probability examples in the data distribution that the neural networks are not able to generalize over as they are too non-linear. 2) The neural networks are too linear, causing them to give too high probability to adversarial examples that exploit their linearity. 3) Adversarial examples are caused by overfitting. 4) The neural networks' decision boundary is too close to the data samples.

2.3 Applications of neural networks to safety-critical systems

Due to neural networks' ability to learn complex problems, they are being considered and tested in many areas where it is essential that they behave as expected. Bojarski et al. (2016) use a convolutional neural network for self-driving cars in an end to end approach where images from a camera are fed to a network which outputs the steering command of the vehicle. Julian et al. (2016) use neural networks in a collision-avoidance system for aircraft. Robertson et al. (2011) use neural networks to predict blood glucose levels in the body, which could be used for a device administering insulin. In all of these cases, an error in the neural network's decision could result in significant damages and even death, and self-driving cars have already resulted in several fatalities (Yadron and Tynan (2016); Levin and Wong (2018)). Even in systems where lives are not necessarily at risk, we must be able to trust the neural networks before we can use them. For example, Graves et al. (2013) use neural networks in speech recognition. If neural network-based speech recognition is used in voice commands, an attacker could use adversarial attacks to create silent adversarial noise, which would be detected as a voice command by the neural network. These commands could be used to perform malicious tasks such as giving the attacker sensitive information about the owner. Neural network-based malware classification, as done by Dahl et al. (2013), is another such example since adversarial malware would not be detected, rendering the malware classifier useless. As long as neural networks are being used in increasingly more fields, the need for them to be robust against adversarial attacks will also increase.

2.4 Measuring the robustness of a neural network

To measure the robustness of a model for a sample, x, we can look at the samples in close proximity to x and the model's predictions for these samples. We would expect a robust model to have the same predictions for x and the samples in close proximity. The smallest adversarial distortion needed on x to change the model's classification would then be the model's robustness for x [Boopathy et al. (2018)]. Consider a classifier $f(x) \rightarrow y$ where x is an sample consisting of n features $\{x_1, x_2, ..., x_n\}$ in the input space X and y is a class of Y. The robustness of f for x is then

$$\min_{x' \in X} D(x, x'), \quad f(x) \neq f(x') \tag{2.6}$$

where $D(x, x')$ is the distance between x and x'. The robustness of a model for a sample is visualized in figure 2.3.

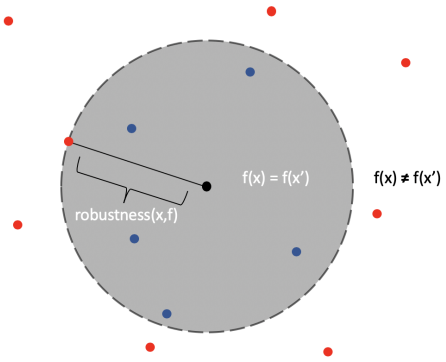


Figure 2.3: A 2D visualization of the robustness for a function, f, given an sample, x. The blue and red dots represents adversarially distorted versions of x, x', with the blue dots giving the same output as x, $f(x) = f(x'_{blue})$, and the red dots giving a different output, $f(x) \neq f(x'_{red})$. The stapled line shows the robustness boundary of f for x and the gray area shows the area where we consider f robust for x.

2.4.1 Distance metrics

For images, which we focus on in this paper, previous work [Grosse et al. (2016); Carlini and Wagner (2017); Boopathy et al. (2018)] suggests that the conventional way to characterize the proximity between an instance and its neighbor is by an L_p norm ball around x with radius ϵ. In this case $\epsilon = D(x, x')$. To calculate the L_p norm we have

$$\epsilon = ||x - x'||_p = (\sum_{i=1}^{n} |x_i - x'_i|^p)^{\frac{1}{p}} \tag{2.7}$$

We find that there are four different L_p norms which are used in previous works; L_0, L_1, L_2 and L_∞. It should be noted that the L_0 "norm" is not a proper norm and does not fit into equation 2.7. Instead, it is the number of non-zero elements in the vector $x - x'$.

- L_0 **"norm" distance.** The L_0 distance "norm" counts the number of pixels where $x_i \neq x'_i$, which is the number of pixels altered. Carlini and Wagner (2017) use this "norm" to minimize the adversarial distortion in the images created by their L_0-attack, and Papernot et al. (2015c) use it to measure the robustness of their models when arguing that defensive distillation is secure against adversarial attacks.

- L_1 **norm distance.** The L_1 norm gives us the Manhattan distance between x and x', which is the sum of all pixel differences. Chen et al. (2017) use the L_1 norm when creating adversarial examples in their Elastic-Net Attacks, and Boopathy et al. (2018) use this norm for calculating the lower bound of neural network models' robustness.

- L_2 **norm distance.** The L_2 norm gives us the Euclidean distance between x and x'. The Euclidean distance penalizes larger pixel changes. Szegedy et al. (2013) uses the L_2 norm to generate adversarial examples with minimal distortion. Carlini and Wagner (2017) use this norm to minimize the adversarial distortion in the images created by their L_2-attack. This norm is also used by Boopathy et al. (2018) and Madry et al. (2017).

- L_∞ **norm distance.** The L_∞ norm gives us the largest pixel distortion, which can be written as $L_\infty = max(|x_1 - x'_1|, ..., |x_n - x'_n|)$. Carlini and Wagner (2017) use this norm

to minimize the adversarial distortion in the images created by their L_∞-attack. Grosse et al. (2016) argue for the L_∞ norm to be used in computer vision, but use the L_1 norm as they have binary input, making the L_∞ norm inappropriate. This norm is also used by Kurakin et al. (2016b), Boopathy et al. (2018), Singh et al. (2019b), Singh et al. (2019a), Katz et al. (2017), Papernot and McDaniel (2016) and Madry et al. (2017).

Using different norms changes the calculated proximity of distorted images and, consequently, the neighborhood around a sample. Figure 2.4 shows the different adversarial examples when different norms are used to minimize the distortion created by the attacks while still causing the model to misclassify. In addition to changing the neighborhood using different norms also changes the range of ϵ. While using the L_∞ norm, ϵ ranges from 0, when no pixel is distorted, to 1 where we have maximal distortion of a pixel. The L_1 norm, however, ranges from 0 to the number of pixels in the image, where all pixels have maximal distortion. For a 28x28 pixel large image, the maximum L_1 value is 784.

Although these metrics make it easy to compare various neural networks, they are not a perfect measurement for how similar humans perceive the images. For example, a perturbation on only one pixel in an image could give a maximum L_∞ distance, even though the images would be considered relatively equal by a human. It is also important to note that this definition of a neural network's robustness only gives the robustness for one instance. A network proven robust on an instance may not be robust on the whole data set, and if proven robust on the data set, it may not be robust on other data sets.

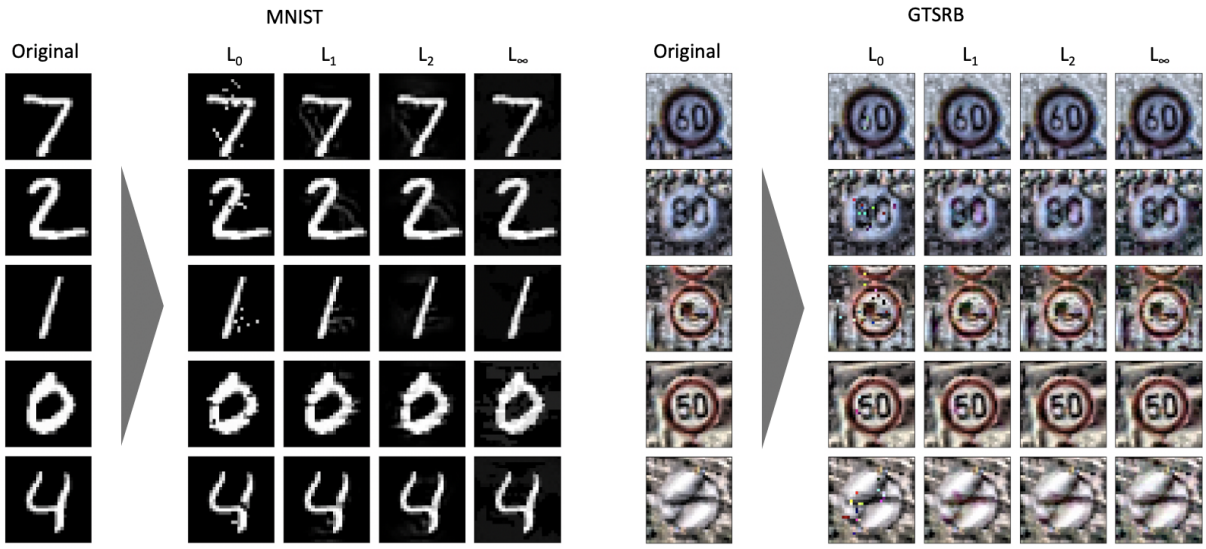


Figure 2.4: The result of attack norms on five different classes from the MNIST (left) and GTSRB (right) data sets. We see the original images on the left side of the data sets, all correctly classified by a neural network. On the rights side of the data sets, we see adversarial examples created by attacks using different L_p norms, all misclassified by the network. The L_0, L_2 and L_∞ attacks are created using the attack algorithms of Carlini and Wagner (2017), and the L_1 attack uses the Elastic-Net attack of Chen et al. (2017).

2.5 Estimating the robustness of a neural network

To find the exact ϵ where the neural network no longer classifies the the samples in close proximity to x correctly is an NP-complete problem and thus not computationally possible for large networks, as shown by Katz et al. (2017). In the literature, I find three different approaches to evaluate the robustness of a neural network, which are less time-consuming than the NP-complete solution. The first is to prove an upper bound of the robustness. The second is to prove a lower bound of robustness, and the third is to evaluate the accuracy of a network on multiple adversarial examples with a given ϵ.

2.5.1 Proving an upper bound

To find the upper bound robustness of a neural network, we can use an attack method to find the smallest distortion that the neural network cannot correctly classify. This requires the attack algorithm to be sufficiently strong to find a small enough distortion to be useful. The upper bound is often used to compare the strength of different attacking algorithms. Carlini and Wagner (2017) and Moosavi-Dezfooli et al. (2015) use this evaluation technique when comparing their attack method against other methods. The strength of this method is that it tests the neural networks' robustness against existing threats. However, this is also a weakness, as this estimation of the robustness cannot be stronger than the strongest attack.

2.5.2 Proving a lower bound

The lower bound is found by proving there is a neighborhood bounded by a max distance, ϵ, where the neural network always classifies the adversarial examples correctly. This can be seen as finding the worst-case robustness and has lately received much attention. Notable examples are CNN-cert of Boopathy et al. (2018) and RefineZono of Singh et al. (2019a). The strength of this method is that it is attack-agnostic and ensures robustness for the neural networks regardless of future attacks. However, this method might create very loose bounds, giving the impression that the networks are less robust than they are.

2.5.3 Success rate

The model's accuracy on multiple adversarial examples, given a set attack distance, gives the success rate. To evaluate the robustness of a neural network with this method, we use an attack algorithm to create multiple adversarial examples that maximize the error of a neural network, while keeping the distortion within a predetermined bound. After the adversarial examples are created, we test the accuracy of the model on the distorted examples. This method is often used to show the efficiency of methods to defend against adversarial attacks. Papernot et al. (2015b), Shaham et al. (2018) and Madry et al. (2017) use this technique to prove the robustness of their defensive methods. A term that is closely related to the success rate is the adversarial error, which is the model's accuracy on the non-perturbed test set minus the model's accuracy on adversarial examples created from the test set. The adversarial error gives us the effectiveness of the attack on the neural network.

The strength and weaknesses of this method are much the same as with upper bounds. It shows the neural networks' performance against existing threats but does not show how they

perform against future attacks. This method also requires that the ϵ of the attack is held constant, which limits the attacks.

Chapter 3

Related work

Parts of this chapter is based on a term paper written in the autumn of 2019.

3.1 Adversarial attacks

3.1.1 L-BFGS

Szegedy et al. (2013) were the first to show that neural networks are vulnerable to adversarial examples. To show this, they use box-constrained limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) to find an approximation of the closest adversary that fools the neural network. That is, given an image x in the input space X, a model f and a targeted class for the adversary y_a which is different from the true label class y_l, they find an adversarial perturbation x' within X that minimizes the sum of the loss and the l_2 distance of x' times a constant, c.

$$\min_{x' \in X} c||x - x'||_2 + L_f(x', y_l) \tag{3.1}$$

The value for c is found using linear search by finding the minimal $c > 0$ where the output of equation 3.1 gives an x' so that $f(x') = y_a$. By using this method, they were able to create adversarial perturbations, not visible to humans, which altered the class of the image.

3.1.2 FGSM

Goodfellow et al. (2014) hypothesize that the adversarial examples are caused by neural networks being too linear and that they are vulnerable to linear distortions. To exploit this vulnerability, they create the Fast Gradient Sign Method (FGSM) attack. The idea of the attack is to perturb the image x in the sign direction of the gradient of the loss function w.r.t. the input pixels. The function used to calculate the FGSM is

$$x' = x + \epsilon Sign(\nabla_x L_f(x, y_l)) \tag{3.2}$$

where $L_f(x, y)$ is the loss for the model f given x and its correct label y_l. ϵ defines the l_∞ distance between x' and x. This method of creating adversarial neural networks is very fast and has a low computational cost compared to other methods, such as L-BFGS, BIM (see section 3.1.5) and the C&W (see section 3.1.6) attacks. However, it also creates weaker adversarial examples.

3.1.3 JSMA

Papernot et al. (2015a) present the Jacobian-based Saliency Map Attack (JSMA) based on the L_0 norm. The attack uses the gradients of the network output, w.r.t the input to compute a saliency map, which is used to monitor the pixels' effect on the network's classification. The attack then modifies the most effective pixel to change the classification of the image. This is done on one pixel at a time until the classification is changed.

3.1.4 DeepFool

Moosavi-Dezfooli et al. (2015) proposed the DeepFool attack, which is based on the L_2 norm. The attack approximates the decision boundary to a polyhedron using an iterative approach. Then, the robustness of the images is found by selecting the part of the polyhedron, which is closest to the original image. They show that DeepFool is able to create adversarial examples with lower perturbation than FGSM.

3.1.5 BIM and MIM

Kurakin et al. (2016a) extend the idea of FGSM by taking multiple smaller steps in the gradients' direction, instead of taking one large step. In each step, the step direction is adjusted with the gradients' direction. The algorithm then becomes:

$$x'_{n+1} = Clip_{X,\epsilon}(x'_n + \alpha Sign(\nabla_{x'_n} L_f(x'_n, y_l))), \quad x'_0 = x \tag{3.3}$$

where x'_n is the adversarial example at step n, α is the step size and $Clip_{X,\epsilon}$ keeps the adversarial images within the valid pixel range of X and ϵ, the predetermined L_∞ distance of x. They call the method the Basic Iterative Method (BIM), but other names such as Projected Gradient Decent (PGD) and Iterative Gradient Sign Method (IGSM) are also used for this method in the literature.

Dong et al. (2018) extend this idea even further and add momentum to the BIM method, making it the Momentum Iterative Method (MIM):

$$x'_{n+1} = Clip_{X,\epsilon}(x'_n + \alpha Sign(g_{t+1})), \quad g_{t+1} = \mu g_t + \frac{\nabla_{x'_n} L_f(x'_n, y_l)}{||\nabla_{x'_n} L_f(x'_n, y_l)||_1} \tag{3.4}$$

where μ is the decay rate of the momentum. The gradients are also normalized to account for the different scales of the gradients in each iteration.

3.1.6 C&W

Carlini and Wagner (2017) created their attacks as a follow up to the defensive distillation of Papernot et al. (2015b), which was a proposed defense against adversarial examples. The Carlini-Wagner (C&W) attacks are similar to the L-BFGS attack. However, they have some differences in that they use the logits instead of the softmax loss and use tanh to constrain the range of the adversarial examples. The attacks were created on the l_0, l_2, and l_∞ norm, all of which defeats defensive distillation. They also show that their attacks are able to create adversarial examples with less perturbation than other attacks such as FGSM, BIM, Deepfool and JSMA. For FGSM and BIM, they searched over ϵ to find the smallest perturbation able to fool the network.

3.1.7 EAD

Chen et al. (2017) noted that there had been little development on adversarial attacks relying on the L_1 norm, despite the L_1 norm being popular in fields of image processing such as denoising and restoration. To account for this, they present the Elastic-Net Attack (EAD), which is based on the C&W attacks and Elastic-Net Regularization. They show that this attack creates adversaries with a much smaller L_1 distance than the C&W L_2 attack.

3.1.8 BA and HSJA

Brendel et al. (2017) propose the Boundary Attack (BA) that does not need access to any parts of the neural network, but only its predictions. The idea of the algorithm is to use an adversarial example x', which is pixel-wise far from the original image and then change x' towards the original image. When the border where x' no longer is an adversarial example is found, the attack changes x' along with the decision border so that x' still is an adversarial image and the distance to the original image is reduced. They show that this attack gives adversarial images which have a slightly higher distance from the original image than the C&W, making BA slightly worse. However, it also uses much less information from the neural networks to create adversarial examples.

Chen et al. (2019) present the HopSkipJumpAttack (HSJA) based on BA and improve it by using an estimation of the pixel gradient direction. The estimated pixel gradient direction is derived from the results gained in the previous steps in the algorithm and is used to update the steps along the boundary more effectively. They show that HSJA creates adversarial examples with much fewer perturbations than BA while also requiring fewer iterations to converge. The HSJA uses the L_2 and L_∞ norm to minimize the perturbation.

3.2 Defences against adversarial attacks

3.2.1 Defensive distillation

Papernot et al. (2015b) suggest using *distilation* to create more robust neural networks. Distillation is a method where knowledge is transferred between a large network or an ensemble of networks to a smaller network (Hinton et al. (2015)). Using this method Papernot et al. (2015b) were able to make the network gradients exploited by adversarial attacks smaller and reduce the variations around the input. This made the network generalize better to adversarial examples, and they were able to reduce the effectiveness of created adversarial examples from 95% to less than 0.5%. Although defensive distillation proved to be very promising, it was later bypassed by new attack methods. Carlini and Wagner (2017) argue that defensive distillation does not increase the robustness of neural networks. By creating three new attack algorithms, they fooled the neural networks created using defensive distillation 100% of the time. Papernot and McDaniel (2017) propose a new variant of defensive distillation that addresses these attacks and are less susceptible to them.

3.2.2 Training on adversarial examples

Shaham et al. (2018) propose to increase the robustness by using perturbated examples to train their neural network. They use a minimum-maximum procedure where the training data is

iteratively replaced by the worst-case perturbed data. The function they wish to optimize is:

$$\min_{\theta} \sum_{i=1}^{m} \max_{\delta_i \in S_i} L(\theta, x_i + \delta_i, y_i) \tag{3.5}$$

Where (x, y) are observation pairs, θ are the trained parameters, and L denotes the loss of the network given x, y, and θ. δ is the amount of perturbation, and S is the set of allowed perturbations. The worst case, $x_i + \delta_i$, for the data samples, was calculated using a method similar to the FGSM method of Goodfellow et al. (2014). For each mini-batch, all the samples were replaced with the worst case before the network parameters were updated using gradient descent. Shaham et al. (2018) show that their networks achieve much higher accuracy than a standard neural network when tested on adversarial examples created in the same way they created the worst-case perturbed data. Going from 0% accuracy using a standard neural network to 79.96% using their network on the MNIST dataset and from 0% to 65.01% on the CIFAR-10 data set.

Madry et al. (2017) also suggest using a minimum-maximum procedure like in equation 3.5 to minimize the loss over the architecture parameters given the maximum loss over the allowed perturbations. Instead of using FGSM to maximize the inner part of the equation, they propose to use BIM to find the worst case of $x_i + \delta_i$. By doing this, they managed to get 45.8% accuracy on the CIFAR-10 data set when tested against adversarial examples created by the BIM algorithm. Comparatively, a normal neural network got 3.5% accuracy, and a network trained the FGSM attack achieved 0%, breaking the FGSM-training defense method. Training on the BIM attack also showed good results against the FGSM attack and the C&W attacks with over 90% accuracy on the MNIST set against both attacks. They also show that wider neural networks achieve higher accuracy against adversarial attacks.

3.2.3 Defensive architectures using verified evaluation

Raghunathan et al. (2018) argue that adversarial training such as the minimum-maximum techniques in section 3.2.2 is problematic as the worst-case loss is based on a lower bound. This causes the optimizer to be misled when the bound is loose and thus makes it hard to generalize to new attack methods. However, calculating the exact worst-case perturbation is not a good solution either as it is computationally infeasible, as shown by Katz et al. (2017). Raghunathan et al. (2018) proposes to use an upper bound on the worst-case loss using a method based on semidefinite relaxation. By using this method, their neural network had a 16% error on the MNIST set against the BIM attack algorithm with a perturbation of 0.1 in the L_∞ norm, and a certified upper bound of 35% error against any attack. Comparatively, the worst-case error of the network from Madry et al. (2017) was at 11% on the MNIST when tested against multiple attacks with a perturbation of in the L_∞ norm. The network trained on the BIM attack algorithm thus performs better than the one based on semidefinite relaxation against known attacks. However, Raghunathan et al. (2018) argue this might be because they use a smaller network with fewer layers. The neural network they used was a relatively small network of only two layers, which is too small for most problems.

Kolter and Wong (2017) propose a similar approach by creating a convex outer adversarial polytope, which is "the set of all final-layer activations that can be achieved by applying a norm-bounded perturbation to the input"[Kolter and Wong (2017)]. Using the convex outer bound, they compute the worst-case loss and use this to minimize the loss over the architecture parameters like minimum-maximum methods in section 3.2.2. With this approach on the MNIST set,

they obtain a classifier that can be proven to have less than 5.8% error for any adversarial attack with perturbations less or equal to 0.1 on the L_∞ norm. It should be noted that this is a computationally expensive method and used 5 hours to train on a Titan X GPU. MNIST is a rather small problem, on which you can easily achieve over 98% accuracy with a standard laptop CPU in a couple of minutes. It is unlikely that this method could scale to bigger problems.

3.3 The relationship between accuracy and robustness

Su et al. (2018) analyze the robustness of several neural networks to explore the tradeoff between robustness and accuracy. To get the robustness of the networks, they use the evaluation methods listed in section 2.5, the lower bound, the upper bound, and the success rate. However, instead of using the success rate, they use the attack success rate, which is one minus the success rate. For the success rate, they use the FGSM, BIM, C&W, and EAD attack. For the upper bound, they use the BIM attack with a predetermined ϵ and the C&W attack, and for the lower bound, they use CLEVER of Weng et al. (2018b). With these robustness estimation techniques, they show that robustness and accuracy are linked, and that robustness increases logarithmically with the classification error. Additionally, their results indicate that the size and depth of the neural networks have little influence on the neural networks' robustness.

3.4 The effect of the hyperparameters of nerual networks

Burkard and Lagesse (2019) test the robustness of various nerual networks with different hyperparameters. To evaluate the robustness of the neural networks, they use the C&W attack on the L_2 norm to create an upper bound of the robustness on the MNIST dataset. They find that the hyperparameters impact the neural networks' robustness. However, changing the hyperparameters alone is not enough to create a viable defense against adversarial examples. The hyperparameters with a high impact on the robustness was dropout, pool size, and kernel size, whereas the number of filters in the CNNs and depth were not important for the robustness. The and activation function was not important for the robustness either, except if tanh was used, which lowered the robustness of the neural networks substantially.

Chapter 4

Design and implementation

4.1 Motivation

With the increased application of neural networks to solve security-critical real-life tasks, such as self-driving cars, it is crucial that we can trust the neural networks and their results. With the discovery of adversarial examples, there has been much research to understand why neural networks are vulnerable to them [Szegedy et al. (2013); Gu and Rigazio (2014); Goodfellow et al. (2014); Tanay and Griffin (2016); Rozsa et al. (2016)] and how to negate them [Papernot et al. (2015b); Papernot and McDaniel (2017); Shaham et al. (2018); Madry et al. (2017); Raghunathan et al. (2018); Kolter and Wong (2017)]. Unfortunately the defensive methods are either susceptible to new attack methods, such as the methods of Papernot et al. (2015b) and Shaham et al. (2018), or not proven to be able to scale to big problems as the methods of Raghunathan et al. (2018) and Kolter and Wong (2017). To make the matter even worse, we still do not know precisely why neural networks are vulnerable to adversarial examples or what causes them. Although there are several hypotheses about adversarial examples, as listed in section 2.2.1, we do not know which of them is correct. Some of theme are even contradictory such as the linearity and non-linearity hypotheses. In the literature, I find no experiments to test the hypotheses up against each other.

The linear, non-linear, and overfitting hypotheses can be directly tested by changing relevant hyperparameters. The decision boundary hypothesis can be tested by changing the training input given to the neural networks. In this thesis, I intend to test the four hypotheses listed in section 2.2.1. By doing this, I aim to strengthen or disprove the hypotheses, which I believe will help to create more robust neural networks in the future by furthering our understanding of adversarial examples.

4.1.1 Research questions

The motivation behind this research was to understand what makes neural networks vulnerable to adversarial examples and how we can make more robust neural networks. Based on this research motivation, I formulated four research questions:

- **RQ1** Are the neural networks vulnerable to adversarial examples because they are too linear? If so, how can this be used to improve the robustness of neural networks?

- **RQ2** Are the neural networks vulnerable to adversarial examples because they are too non-linear? If so, how can this be used to improve the robustness of neural networks?
- **RQ3** Are the neural networks vulnerable to adversarial examples because they overfit to the adversarial examples? If so, how can this be used to improve the robustness of neural networks?
- **RQ4** Are the neural networks vulnerable to adversarial examples because their decision boundary is too close to the training samples? If so, how can this be used to improve the robustness of neural networks?

4.2 Method

To answer the research questions, I conducted two experiments that tested how the hyperparameters and the training input affected the neural networks. For the first experiment, I trained multiple neural networks with different hyperparameters and calculated their robustness using multiple robustness estimation methods. By doing this I expected to answer **RQ1**, **RQ2** and **RQ3**, which is explained further in section 4.2.1, 4.2.2 and 4.2.3. For the second experiment, I trained on adversarial examples in order to answer **RQ4**. This is further explained in section 6.5. In both experiments, I used multiple datasets with varying complexity.

4.2.1 Design to answer RQ1

Independent variables

To answer **RQ1**, I wanted to test the linearity hypothesis described in section 2.2.1. Of the hyperparameters in a neural network, I expected the activation function and depth to affect the linearity. Certain activation functions are more non-linear than others, and thus using a more non-linear activation function should decrease the linearity of a neural network. For each layer in a neural network, the activation function is applied, making the neural network increasingly non-linear with the depth.

Because I expected the activation function and the depth to change the linearity of neural networks. They were used as independent variables in my first experiment, where I tested the robustness of neural networks with different hyperparameters.

Dependent variables

The dependent variable of my first experiment would optimally be the neural networks' robustness. However, as written in section 2.5, calculating the exact robustness of a neural network is an NP-complete problem and not computationally possible. To overcome this, I used the estimations of the neural networks' robustness as dependent variables. In section 2.5, there are listed three ways to estimate the robustness of neural networks that are used in the literature; calculating the success rate, the upper bound, and the lower bound. As mentioned, there are strengths and weaknesses in all techniques.

When Su et al. (2018) compare the robustness of multiple neural networks, they use all three techniques. To ensure the potential trends seen in the robustness of the neural networks are correct, I used decided all three methods as well. For example, if increasing some hyper-parameter also increases the upper bound, there could be two possible explanations. The first explanation

is that the actual robustness of the model was increased. However, a second explanation is that the attack algorithm just performed worse even though the true robustness of the network was not increased. Using multiple methods will give a broader picture and make it easier to understand how the hyperparameters correctly affect the robustness. The implementations of the robustness estimation techniques are listed in section 4.3.

Hypotheses concerning the effect of the depth on the neural networks' robustness

From the independent and dependent variables I had several predictions about how the depth affect the robustness that I wanted to prove. The dependent variables chosen are described in section 4.3. For each prediction I formulated two hypothesis statements:

Null Hypothesis 1 (NH1)**.** *The depth does not correlate with the lower bound.*

Alternative

Alternative Hypothesis 1 (H1)**.** *The depth is correlated with the lower bound.*

Null Hypothesis 2 (NH2)**.** *The depth does not correlate with the C&W based upper bound.*

Alternative Hypothesis 2 (H2)**.** *The depth is correlated with the C&W based upper bound.*

Null Hypothesis 3 (NH3)**.** *The depth does not correlate with the HSJA upper bound.*

Alternative Hypothesis 3 (H3)**.** *The depth is correlated with the HSJA based upper bound.*

Null Hypothesis 4 (NH4)**.** *The depth does not correlate with the BIM success rate.*

Alternative Hypothesis 4 (H4)**.** *The depth is correlated with the lower BIM success rate.*

Null Hypothesis 5 (NH5)**.** *The depth does not correlate with the lower MIM success rate.*

Alternative Hypothesis 5 (H5)**.** *The depth is correlated with the lower MIM success rate.*

Hypotheses concerning the effect of the activation functions on the neural networks' robustness

As with the depth, I had several predictions about how the activation functions affect the robustness that I wanted to prove:

Null Hypothesis 6 (NH6)**.** *The more non-linear activation functions and relu affect the lower bound equally.*

Alternative Hypothesis 6 (H6)**.** *The more non-linear activation functions have either a positive or a negative effect on the lower bound when compared to relu.*

Null Hypothesis 7 (NH7)**.** *The more non-linear activation functions and relu affect C&W based upper bound equally.*

Alternative Hypothesis 7 (H7)**.** *The more non-linear activation functions have either a positive or a negative effect on the C&W based upper bound when compared to relu.*

Null Hypothesis 8 (NH8)**.** *The more non-linear activation functions and relu affect the HSJA upper bound equally.*

Alternative Hypothesis 8 (H8)**.** *The more non-linear activation functions have either a positive or a negative effect on the HSJA upper bound when compared to relu.*

Null Hypothesis 9 (NH9)**.** *The more non-linear activation functions and relu affect the BIM success rate equally.*

Alternative Hypothesis 9 (H9)**.** *The more non-linear activation functions have either a positive or a negative effect on the BIM success rate when compared to relu.*

Null Hypothesis 10 (NH10)**.** *The more non-linear activation functions and relu affect the MIM success rate equally.*

Alternative Hypothesis 10 (H10)**.** *The more non-linear activation functions have either a positive or a negative effect on the MIM success rate when compared to relu.*

Data analyses

To analyze the data, I used correlation analysis, t-tests, and linear regression analysis. The correlation analysis gave me the correlation between each independent and dependent variable, which was calculated for each dataset individually and on the result of all the datasets combined. The correlation was used on NH1 to NH5. For NH6 to NH10 t-test was used. However, correlation analyses and t-tests do not control for the effect of other independent variables, meaning that the other variables can interfere with the result of the tested variable. To account for this, I also used multiple linear regression, which gives the effect of the independent variables after controlling for the impact of the other variables.

Accounting for the accuracy

According to Su et al. (2018), the robustness of the neural networks' decreases with their accuracy. Because of this, I would have preferred that all neural networks I had the same accuracy. However, this was not the case as different hyperparameters led to different accuracies in the neural networks. There were two possible strategies to avoid that the hyperparameters that gave neural networks with high accuracy were penalized. The first was to discard all neural networks without sufficient accuracy and only use the remaining neural networks for the analyses. The second strategy was to let accuracy be an independent variable of the regression so that the regression would account for the accuracy. The second strategy was only possible with regression.

I decided to mainly use the first strategy and discard all neural networks with an accuracy of less than 95%. However, for the most challenging datasets, the second approach was used for the regression. For these datasets, the neural networks with less than 50% accuracy were discarded regardless as the robustness became very high for neural networks with very low accuracy. In the correlation analyses, these datasets were not used at all. The test accuracy was used to determine the accuracy of the network.

Analyses of the categorical variables in the regression analysis

According to dum (1993), we can use regression on categorical variables by defining a set of dummy variables and use the dummy variables in the regression. To do this, we create a unique dummy variable for each category and set them to 1 if the category is present and 0 if not. Since

the last category value can be represented by the absence of the other category values, we need one less dummy variable than there are categories. As the activation function is a categorical variable, this strategy will be used for the activation function, and relu will be the category left out.

Norms

Since it is uncertain how different the norms affect the neural networks' robustness in relation to the hyperparameters, I decided to use multiple L_p norms when estimating the robustness. This is described in further detail in section 4.3.

Range of the hyperparameters

When running the experiment, I noticed that very deep networks caused the estimation techniques to use too much time to compute. Because of this, I did not test neural networks deeper than five layers. Thus the depths of the neural networks ranged from 1 to 5. For the activation functions, I used relu, which is the most commonly used activation function in addition to sigmoid, arctan, and tanh, which are more non-linear than relu. To correctly determine the impact of the activation function, all the layers in the neural networks used the same activation function.

Neural network architecture

To limit the scope of the experiment, I restricted the data sets to image data sets and the type of neural networks to CNNs. I chose image datasets as most research on adversarial examples is done on images, making it easier to compare to and build upon previous works. Additionally, images and differences between images are easy to display and understand. CNNs were chosen as most state of the art image classification models such as the GoogLeNet of Szegedy et al. (2014b), the VGGNet of Simonyan and Zisserman (2014) and the ResNet of He et al. (2015a) use convolution in their networks. Different varieties of CNNs, such as the skip connections, found in the ResNet, or inception layers, found in the GoogLeNet, are not considered in this experiment to keep the scope within a reasonable size.

4.2.2 Design to answer RQ2

To answer **RQ2**, I wanted to test the non-linearity hypothesis described in section 2.2.1. Since this hypothesis is just a contradiction of the linearity hypothesis, the same method used to answer **RQ1** was used to test this hypothesis.

4.2.3 Design to answer RQ3

Independent variables

According to Goodfellow et al. (2016), overfitting occurs when neural networks with high capacity memorize properties of the training set. As deeper and wider networks have higher capacity, increasing the depth and width should cause more overfitting and thus make the neural networks less robust if the overfitting hypothesis described in section 2.2.1 is true. Because

of this, the number of filters, which is the width of a CNN, was added as an independent variable to the first experiment. The depth was already added as an independent variable in order to answer **RQ1** and **RQ2**.

In the experiments, I used early stopping to avoid overfitting when training the neural networks. However, the early stopping only considers the loss on the validation data, which does not contain adversarial examples, and thus overfitting on adversarial examples should happen anyway.

Dependent variables, Data analysis and Configuration

The same dependent variables, norms, and architecture used to answer **RQ1** and **RQ2** were used to answer **RQ3**. The range of the depth is described in section 4.2.1. The width, or the number of filters, ranged from 4 to 80. To test hypothesis NH11 to NH15, listed in section 4.2.3, I used correlation analyses.

The number of filters is usually increased with the depth of the CNNs so that they start with few filters in the first layer and have many in the last layer. To correctly determine the impact of the width, all the layers in the neural networks had an equal amount of filters.

Hypotheses concerning the effect of the activation functions on the neural networks' robustness

As with the depth and activation functions, listed in section 4.2.1 and 4.2.1, I had several predictions about how the width of the neural networks affect their robustness that I wanted to prove:

Null Hypothesis 11 (NH11)**.** *The width does not correlate with the lower bound.*

Alternative Hypothesis 11 (H11)**.** *The width is correlated with the lower bound.*

Null Hypothesis 12 (NH12)**.** *The width does not correlate with the C&W based upper bound.*

Alternative Hypothesis 12 (H12)**.** *The width is correlated with the C&W based upper bound.*

Null Hypothesis 13 (NH13)**.** *The width does not correlate with the HSJA upper bound.*

Alternative Hypothesis 13 (H13)**.** *The width is correlated with the HSJA based upper bound.*

Null Hypothesis 14 (NH14)**.** *The width does not correlate with the BIM success rate.*

Alternative Hypothesis 14 (H14)**.** *The width is correlated with the BIM success rate.*

Null Hypothesis 15 (NH15)**.** *The width does not correlate with the MIM success rate.*

Alternative Hypothesis 15 (H15)**.** *The width is correlated with the MIM success rate.*

4.2.4 Design to answer RQ4

Independent variable

To answer **RQ4**, I decided to train neural networks on adversarial examples. If the neural networks' vulnerability to adversarial examples is caused by the decision boundary being to close to the training examples, then training on adversarial examples should increase the robustness closely to the ϵ used in the attack. Because of this, the ϵ of the attack is used when training the networks was used as the independent variable in the second experiment.

Adversarial attack and dependent variable

The attack algorithm chosen for the adversarial examples was the BIM attack. Madry et al. (2017) had already created neural networks trained on this attack, which they show to perform well against the FGSM, C&W, and BIM attack. However, the defensive methods of Shaham et al. (2018) and Papernot et al. (2015b) show that it is not enough to prove defenses are effective against known attacks, as both were broken when newer attacks were created. Because of this, I wanted to see if this defensive method also increased the lower bound, as the lower bound is attack agnostic. Thus, the lower bound of the neural networks was used as the second experiment's dependent variable. For **RQ4** I have one pair of hypotheses that I want to test:

Null Hypothesis 16 (NH16)**.** *The ϵ of the attack used to train a neural network does not correlate with the lower bound of the neural network.*

Alternative Hypothesis 16 (H16)**.** *The ϵ of the attack used to train a neural network correlates with the lower bound of the neural network.*

Data analysis and configuration

I used a correlation analysis to find the correlation between the ϵ of the attack and the lower bound. To have a decent population size, I trained 25 neural networks for each ϵ. For the hyperparameters, the relu activation function was used. The depth and number of filters ranged from 1 to 5 and 8 to 72. As the hyperparameters are not used as independent variables in this experiment, I do not consider them to be particularly relevant, and they were chosen arbitrarily. Still, I have listed them for reproducibility purposes.

For this experiment, all neural networks with a test accuracy of less than 50% were discarded. Additionally, when Madry et al. (2017) trained on adversarial examples, they used ten steps of the BIM attack for Cifar and 40 for MNIST, with a step size of $\frac{1.33*\epsilon}{steps}$. For this experiment, I used 20 steps for all datasets and the same step size as Madry et al. (2017).

4.3 Implementation

4.3.1 Trainig the neural networks

To train the neural networks for my experiments, I used Keras[1] with tensorflow[2] as the backend. Tensorflow is an open-source machine learning library, and Keras is a high-level neural networks API running on top of TensorFlow, making it easier to create neural networks. Both data augmentation and early stopping were applied when training the neural networks to obtain as high accuracy as possible. Early stopping measures the loss on the validation set between each epoch. When the validation loss has not improved over a certain amount of epochs, the weights from the epoch with the lowest validation loss is restored. This ensures that the neural network is trained for an optimal number of epochs and that overfitting and underfitting is avoided. Augmentation lets us create synthetic data by applying different augmentation techniques such as zooming and rotation to the training images. By doing this, we get a larger training set, which makes it possible to train on more samples before the neural networks start to overfit.

[1] https://keras.io/

[2] https://www.tensorflow.org/

4.3.2 Calculation of lower bound

As mentioned in section 2.5.2, recent research has provided methods that can give us a certified minimum value of the robustness. To find the optimal tool to calculate the lower bound, I searched previous literature and found and compared seven state of the art methods: CLEVER by Weng et al. (2018b), DeepZ by SINGH et al. (2018), DeepPoly by Singh et al. (2019b), RefineZono by Singh et al. (2019a), Fast-Lin by Weng et al. (2018a), CROWN by Zhang et al. (2018) and CNN-Cert by Boopathy et al. (2018). For the lower bounds, I wanted to use the method that gave the highest lower bounds that were fast enough to test all my neural networks within a reasonable time.

CLEVER, which is used by Su et al. (2018), does not give a certified lower bound, meaning that it cannot guarantee that the robustness is lower than the robustness, making it less trustworthy than the other lower bound methods. Figure 4.1 and 4.2 show that CNN-Cert performs as good as or better than Fast-Lin, DeepZ, CROWN, and DeepPoly in both speed and tightness of bound. Figure 4.3 shows that RefineZono gives a tighter bound than DeepZ and DeepPoly, which have bounds comparable to CNN-Cert, but is also a lot slower, sometimes using over 1000 times more time than DeepZ and DeepPoly to reach their results. Thus, CNN-Cert was chosen over RefineZono as RefineZono was considered too computationally expensive.

Note: x = not supported, * = computationally infeasible (94 GB RAM, AMD Zen CPU)

Network		Certified Bounds						Attack	Improvement
	p norm	Fast-Lin	LP	DeepZ	DeepPoly	CNN-Cert ReLU	CNN-Cert general	CW/EAD	vs. Fast-Lin (CNN-Cert ReLU)
MNIST, 4 layer	L∞	0.04	0.05	0.04	0.05	0.04	0.05	0.15	+21%
5 filters	L2	0.15	0.17	x	x	0.15	0.18	3.14	+23%
8680 hidden nodes	L1	0.28	0.31	x	x	0.28	0.34	14.45	+22%
CIFAR, 5 layer	L∞	0.004	*	*	*	0.004	0.004	0.02	+14%
10 filters	L2	0.03	*	x	x	0.03	0.03	0.42	+17%
29360 hidden nodes	L1	0.08	*	x	x	0.08	0.10	11.65	+18%
MNIST, ResNet-3	L ∞	x	x	x	x	0.02	0.02	0.04	+13%
	L 2	x	x	x	x	0.08	0.08	0.39	+5%
	L 1	x	x	x	x	0.15	0.15	5.95	+4%
MNIST CNN 2-layer	L ∞	x	x	0.03	x	x	0.04	0.06	-
5 filters	L 2	x	x	x	x	x	0.15	0.76	-
Tanh	L 1	x	x	x	x	x	0.31	9.31	-

Note: x = not supported, * = computationally infeasible (94 GB RAM, AMD Zen CPU)

Network		Average Computation Time (sec)					CNN-Cert Speed-up			
	p norm	Fast-Lin	LP	DeepZ	DeepPoly	CNN-Cert	vs. Fast-Lin	vs. LP	vs. DeepZ	vs. DeepPoly
MNIST, 4 layer	L∞	9.03	853.20	3.72	25.26	2.33	x3.9	x366.1	x1.6	x10.8
5 filters	L2	9.19	236.30	x	x	0.88	x10.5	x270.1	-	-
8680 hidden nodes	L1	8.98	227.69	x	x	0.86	x10.5	x265.2	-	-
CIFAR, 5 layer	L∞	169.29	*	*	*	20.87	x8.1	-	-	-
10 filters	L2	170.42	*	x	x	16.93	x10.1	-	-	-
29360 hidden nodes	L1	168.30	*	x	x	17.07	x9.9	-	-	-
MNIST, ResNet-3	L ∞	x	x	x	x	10.04	-	-	-	-
	L 2	x	x	x	x	10.11	-	-	-	-
	L 1	x	x	x	x	10.15	-	-	-	-
MNIST CNN 2-layer	L ∞	x	x	3.98	x	0.05	-	-	x77.4	-
5 filters	L 2	x	x	x	x	0.05	-	-	-	-
Tanh	L 1	x	x	x	x	0.05	-	-	-	-

Table 4.1: Tables comparing the speed and bounds of Fast-Lin, DeepZ, DeepPoly, and, CNN-Cert. The tables are from the Github page of CNN-Cert[3].

CNN-Cert allows for testing the L_1, L_2 and L_∞ norms, and all these norms were used to test the lower bound. For each neural network in the experiments, CNN-Cert was used to estimate the average lower bound on ten images for all three norms, using the top-2 prediction

[3]https://github.com/IBM/CNN-Cert

Network	Certified Bounds			CNN-Cert-Ada Imp. (%)	Average Computation Time (sec)		CNN-Cert-Ada Speed-up
	ℓ_p norm	CNN-Cert-Ada	CROWN (Zhang et al. 2018)	vs. CROWN	CNN-Cert-Ada	CROWN	vs. CROWN
MNIST, 8-layer	ℓ_∞	0.0203	0.0203	0%	18.34	45.92	2.50
5 filters	ℓ_2	0.0735	0.0734	0%	18.25	47.44	2.60
ReLU	ℓ_1	0.1284	0.1284	0%	18.35	112.29	6.12
MNIST, 8-layer	ℓ_∞	0.0237	0.0237	0%	18.27	208.21	11.40
5 filters	ℓ_2	0.0877	0.0877	0%	18.22	208.17	11.42
Ada	ℓ_1	0.1541	0.1540	0%	18.51	126.59	6.84
MNIST, 8-layer	ℓ_∞	0.0841	0.0827	2%	18.81	186.71	9.93
5 filters	ℓ_2	0.3441	0.3381	2%	18.83	180.46	9.58
Sigmoid	ℓ_1	0.7319	0.7185	2%	19.40	202.25	10.43
MNIST, 8-layer	ℓ_∞	0.0124	0.0051	146%	20.31	188.15	9.26
5 filters	ℓ_2	0.0735	0.0215	242%	19.70	219.83	11.16
Tanh	ℓ_1	0.1719	0.0478	260%	20.00	182.73	9.14
MNIST, 8-layer	ℓ_∞	0.0109	0.0067	62%	19.03	210.42	11.06
5 filters	ℓ_2	0.0677	0.0364	86%	19.05	203.11	10.66
Arctan	ℓ_1	0.1692	0.0801	111%	19.36	179.29	9.26

Table 4.2: Tables from Boopathy et al. (2018) comparing the speed and bounds of CROWN and CNN-Cert.

Dataset	Model	ϵ	*DeepZ*		*DeepPoly*		*RefineZono*	
			precision(%)	time(s)	precision(%)	time(s)	precision(%)	time(s)
MNIST	5×100	0.07	38	0.6	53	0.3	53	381
	6×100	0.02	31	0.6	47	0.2	67	194
	9×100	0.02	28	1.0	44	0.3	59	246
	6×200	0.015	13	1.8	32	0.5	39	567
	9×200	0.015	12	3.7	30	0.9	38	826
	ConvSmall	0.12	7	1.4	13	6.0	21	748
	ConvBig	0.2	79	7	78	61	80	193
	ConvSuper	0.1	97	133	97	400	97	665
CIFAR10	6×100	0.0012	31	4.0	46	0.6	46	765
	ConvSmall	0.03	17	5.8	21	20	21	550

Table 4.3: Table from Singh et al. (2019a) comparing the speed and bounds (in the percentage of the true robustness) of DeepZ, DeepPoly, and RefineZono.

for the adversary's target. Although ten images don't give the lower bound of the whole dataset, I expect it to reveal trends when used on multiple neural networks and data sets. The top-2 prediction was used as CNN-Cert only can calculate the robustness for one target, and the top-2 prediction was expected to give adversarial example closest to the original.

4.3.3 Calculation of upper bound

As described in section 2.5.1, finding the upper bound for an image consists of finding the smallest perturbation that can fool the network. For this, I find the C&W attacks and the EAD attack, described in section 3.1, the best candidates since they create adversarial examples with the least perturbation when compared to other attacks using the same distance norm. Another potential candidate is the HSJA, as it performs better than BA, which is slightly worse than C&W. I was not able to find any comparison between C&W and HSJA in the literature, making me unsure which of them are the best. Because of this, both attacks were used.

Like with the lower bound, many norms were used to calculate the upper bound of the robustness. To compare the results with the lower bounds, the same methods were used, with ten images for each neural network and estimating the robustness with the top-2 prediction as the target. Additionally, the same norms used in the lower bounds were used in the upper bounds as well. The EAD was used for the L_1 norm and both HSJA and C&W for the L_2 and L_∞ norm. However, I experienced the HSJA algorithm to get stuck while searching for examples predicted to be within the adversarial target's class, even when it was initiated with an example of the target class. Because of this, I decided to use non-targeted HSJA, which lets the algorithm use any class as a target for the adversarial examples.

For the C&W and EAD attacks, the parameters were set to the recommended values, using 10000 iterations for the L_1 and L_2 attack and 1000 iterations for the L_∞ attack. For the HSJA attack, the parameters were also set to the recommended values, except for the number of iterations, which increased from 40 to 150 in an effort to create tighter bounds.

4.3.4 Calculation of success rate

Although the C&W, EAD, and HSJA attacks have shown to give the lowest bounds when creating adversarial examples, they do not seem to find the strongest adversary when we search within a predetermined border. Zhang and Li (2019) compares the percentage of adversarial examples able to fool the network of several adversarial attacks. Their results can be seen in table 4.4. Their work seems to indicate that non-targeted BIM and non-targeted Deepfool creates the adversarial examples that most often fool the neural networks. However, they do not explain how their results are obtained. Additionally, which parameters are used, such as whether they use a maximum perturbation boundary, how many iterations of the attacks are used, and which type of neural networks used remains elusive. Vargas and Kotyan (2019) also compares the strength of several adversarial examples, and their results can be seen in table 4.5. Their results indicate that BIM is much stronger than Deepfool. However, they too do not explain how they obtain their results and what parameters are used.

The literature seems to indicate that MIM and BIM create the strongest adversaries when a predetermined maximum boundary for the perturbation is set. In addition to this, they are also more suited to find adversarial examples when we have a predetermined boundary. C&W, EAD, and HSJA are made to find the smallest adversary perturbation that is able to change the model's prediction. BIM and MIM find the adversarial example within a predetermined

THE SUCCESS RATE OF ADVERSARIAL EXAMPLES

Method \ Type	Targeted	Non-targeted
L-BFGS [24]	96.1%	NA
FGSM [27]	NA	74.6%
IGSM [33]	85.4%	99.1%
MIM [50]	99.3%	100%
DeepFool [45]	NA	100%
CW Attack [47]	93.2%	98.4%
Uni.perturbations [46]	NA	88.2%
JSMA* [31]	-	-

* Out of Memory.

Table 4.4: Tables from Zhang and Li (2019) comparing the success rate of different adversarial attacks. Note that the success rate used by Zhang and Li (2019) is not the success rate used in section 2.5.3 but the success rate for the adversarial examples.

Attack Name	ResNet		CapsNet		AT		FS	
	Acc.	L_2 Score	Acc.	L_2 Score	Acc.	L_2 Score	Acc.	L_2 Score
Fast Gradient Sign Method	77	2586.77	94	2594.83	-	-	75	2601.25
Basic Iterative Method	100	3476.71	100	3445.87	100	3459.23	100	3426.23
Carlini and Wagner	95	2487.60	48	2750.75	39	2274.21	93	2551.54
Deep Fool	43	2799.44	64	2599.08	23	2232.59	38	2646.84
Newton Fool	47	2748.40	75	2525.77	57	2224.21	39	2817.67
Virtual Adversarial Method	39	2777.07	43	2786.98	34	2738.42	10	2016.27
Few-Pixel Attack $th = 1$	10	271.81	14	307.75	22	264.23	17	247.74
Few-Pixel Attack $th = 3$	48	434.01	34	531.78	52	488.92	49	446.29
Few-Pixel Attack $th = 5$	72	527.88	45	660.61	66	622.65	69	551.29
Few-Pixel Attack $th = 10$	82	656.75	62	790.81	86	787.24	78	677.31
Threshold Attack $th = 1$	28	39.16	9	39.16	3	38.50	26	39.17
Threshold Attack $th = 3$	78	124.55	38	129.46	12	125.73	63	124.48
Threshold Attack $th = 5$	78	209.24	74	224.55	25	218.87	66	208.81
Threshold Attack $th = 10$	83	402.93	98	443.38	57	440.96	74	399.77

Table 4.5: Tables from Vargas and Kotyan (2019) comparing the adversarial success rate and the L_2 distance achieved by different adversarial attacks.

boundary, which the model gives the least probability for having the original label. Because of this, I decided to use MIM and BIM when calculating the success rate. The MIM and BIM attacks are only made for the L_∞ norm, and thus only this norm will be used for the success rate.

A problem with the success rate is that it penalizes neural networks with low natural accuracy as the neural network's initial accuracy limits the maximum success rate. The adversarial error overcomes this as the success rate is subtracted from the initial accuracy. However, this penalizes neural networks with high accuracy as they need a higher success rate to keep the error low. To overcome this, I decided to use the success rate, but divide it by the initial accuracy to account for lower accuracy neural networks.

4.3.5 Datasets

To ensure that the results from the experiments were generalizable, I chose to have six different data sets to train neural networks and measure robustness on. It was important that the data sets were different enough to ensure generalization. However, they also had to be simple enough for the convolutional neural networks to classify nearly all instances in the sets correctly.

Four metrics for choosing data sets that I expected could be relevant for the end robustness, were used when selecting the data sets. The first metric was content as different image contents may be easier or harder to create adversarial examples, causing the neural networks to have different robustness levels. The second metric was the number of classes. I expect this to be relevant for the robustness since increasing the number of classes increases the attack surface of the neural networks, giving the attacker more ways to fool the neural network. The third metric is the number of color channels. Having three color channels (RGB) instead of one (gray-scale) gives the attack algorithm more input to distort, thus making it likely to affect the robustness. The fourth metric is the complexity of the data set or how easy the data set is to learn for the neural networks. I believe results would generalize to almost any dataset if it generalized over datasets with differences in all these metrics.

The data sets chosen were MNIST, Sign Language MNIST (SLM), Caltech 101 silhouettes(CS), Rock-Paper-Scissors(RPS), Cifar, and the German Traffic Sign Recognition Benchmark(GTSRB). The datasets and the metrics used when choosing them are listed in table 4.6.

MNIST

The MNIST data set of LeCun and Cortes (2010) is a well known and well-tested data set. It consists of handwritten numbers stored in 28 by 28 pixels gray-scale images and has ten classes, which depicts the first numbers from zero to nine, as seen in figure 4.1. The MNIST data set consists of many samples and is a fairly easy data set for the neural networks to learn. To boost the accuracy of the networks, I added vertical and horizontal augmentation when training, which is applied before each batch. This shifts the images up to 10% of the images' height and width in the vertical and horizontal direction.

Figure 4.1: The different classes from the MNSIT data set.

Sign language MNIST

The SLM dataset[4] depicts 24 of the 26 sign language letters from the English sign alphabet (J and Z require motion). The images are 28 by 28 pixels and in gray-scale. This data set is, like MNIST, fairly easy for convolutional neural networks to learn and with a lot of samples. However, with more classes than MNIST, it is slightly more complex. For this data set, I have added the same augmentation as with MNIST. Ten of the classes can be seen in figure 4.2

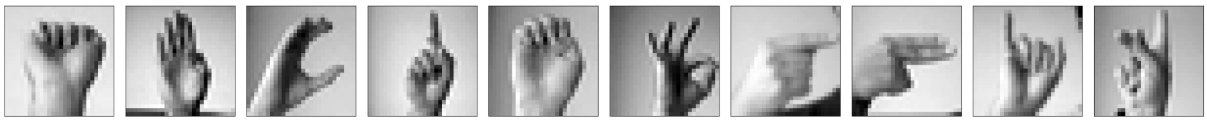


Figure 4.2: Ten classes from the Sign language data set.

Caltech 101 silhouettes

The CS dataset[5] contains the silhouettes from the Caltech 101 dataset of Fei-Fei et al. (2006). In consists of 101 different classes in 28 by 28 pixels wide grayscale images. This data set has only 8671 samples, which gives an average of 86 images per class. Because of this and the similarities between some of the classes, CS is a hard data set to learn for the neural networks. To increase the accuracy, I added augmentation. The augmentation consisted of rotating the images up to 15 degrees in either direction, shifting the images up to 15% in the horizontal and vertical directions, shearing them up to 15%, zooming in up to 15% of the image size and flipping the image horizontally 50% of the time before each batch. Ten of the classes from the Caltech 101 silhouettes data set can be seen in figure 4.3.

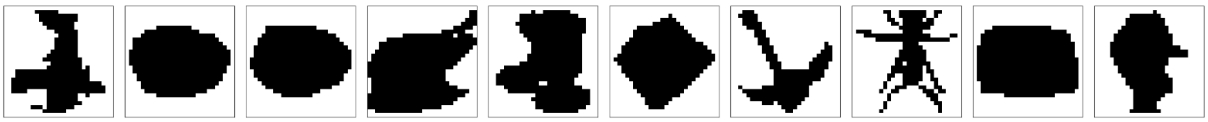


Figure 4.3: Ten classes from the Caltech 101 silhouettes data set.

Rock paper scissor

The RPS data set of de la Bruère-Terreault consists of images depicting the three legal moves from the game "rock paper scissor." The dataset consists of 2188 samples of 300 by 200 pixel RGB images, but the images were resized to 60 by 60 pixels. Due to the few classes, this dataset is less complex than CS. Since the images are larger than SLM, and it has three channels, I expected RPS to be harder for the neural networks to learn than the SLM dataset. The same augmentation as with CS was done on this dataset, in addition to vertical flipping. The classes of the RPS dataset can be seen in figure 4.4

[4]https://www.kaggle.com/datamunge/sign-language-mnist

[5]https://people.cs.umass.edu/ marlin/data.shtml

Figure 4.4: The classes from the RPS data set.

Cifar

The Cifar dataset of Krizhevsky et al. consists of ten classes of various vehicle types and animals. There are 50000 different samples in the dataset, which are 32 by 32 pixel RGB images. I expected this dataset to be harder to learn than RPS as it has more classes. The same augmentation used on CS is used for this dataset. The classes of the Cifar dataset can be seen in figure 4.5

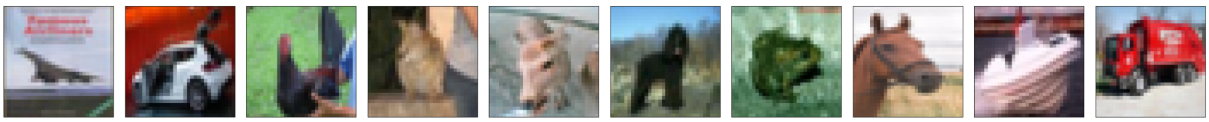


Figure 4.5: The classes from the Cifar data set.

GTSRB

The GTSRB dataset of Stallkamp et al. (2012) consists of different traffic signs. The dataset has 39209 different RGB images of various sizes, which were reshaped to 32 by 32 pixels, and has 43 classes. I expected this dataset to be more complex than Cifar as it has more classes. The augmentation used on this dataset was 5% rotation, 10% zoom, and 10% vertical and horizontal shift. Ten of the GTSRB classes can be seen in figure 4.6.

Figure 4.6: Ten classes from the GTSRB data set.

Data set	Content	Classes	Color Channels	Complexity
MNIST	Numbers	10	1	Low
Sign Language	Sign language letters	24	1	Low/meduim
Caltech 101 silhouettes	Silhouettes	101	1	High
RPS	Hands	3	3	Medium
Cifar	Vehicles and animals	10	3	Medium/high
GTSRB	Signs	43	3	High

Table 4.6: The content, number of classes, number of samples, and expected complexity of the datasets used in the experiments.

4.3.6 Experimental Setup

My experiments were done on The NTNU IDUN HPC cluster[6]. The code for running the experiments can be found on github[7] along with the python libraries used in the code and their version number. The slurm files used to run on the IDUN cluster can also be found in the repository. For my experiments, I used python version 3.7.2. and TensorFlow version 1.13.1.

[6] https://www.hpc.ntnu.no/idun

[7] https://github.com/halvorbmundal/What-causes-robustness-in-neural-networks

Chapter 5

Results

5.1 Accuracy

In figure 5.1, the accuracies of the neural networks with more than 50% accuracy are plotted against the lower bounds. We see that for most of the datasets, the majority of the neural networks achieved over 95% accuracy, which was the threshold for discarding the networks, as described in section 4.2.1. The Cifar and the CS datasets did not have any network with more than 95% accuracy, and as a result, they were not used in the correlation analyses. In the regression analyses, the accuracy was used as one of the independent variables for those datasets.

5.2 Results from exploring RQ1 and RQ2

In order to answer **RQ1** and **RQ2**, the effect of the depth and the activation functions on the neural networks' robustness were tested as described in section 4.2.1 and 4.2.2. To thoroughly examine the effect of these parameters, the lower bound was calculated on three norms. The upper bound was calculated using both the C&W and the HSJA algorithms on three and two norms, respectively, and the success rate was calculated using BIM and MIM as attacking algorithms. In total, ten regression analyses on six data sets, and ten correlation analyses on four data sets were conducted to get a thorough understanding of the effect of the depth and the activation functions on the neural networks' robustness.

5.2.1 Lower bound of the datasets

Only some of the data from the regression analyses using lower bound as the dependant variable are listed in this section. For a complete list see appendix section A.

The effect of the neural networks' depth on the lower bound

From the correlation analyses in table 5.1, we see that the lower bound is negatively correlated with the depth for all datasets and on all norms. Additionally, every coefficient has a statistically significant p-value. However, the correlation coefficient is only around 0.5, which is not high. In the regression analyses in table 5.3 and 5.4, we see that the depth affects the lower bound

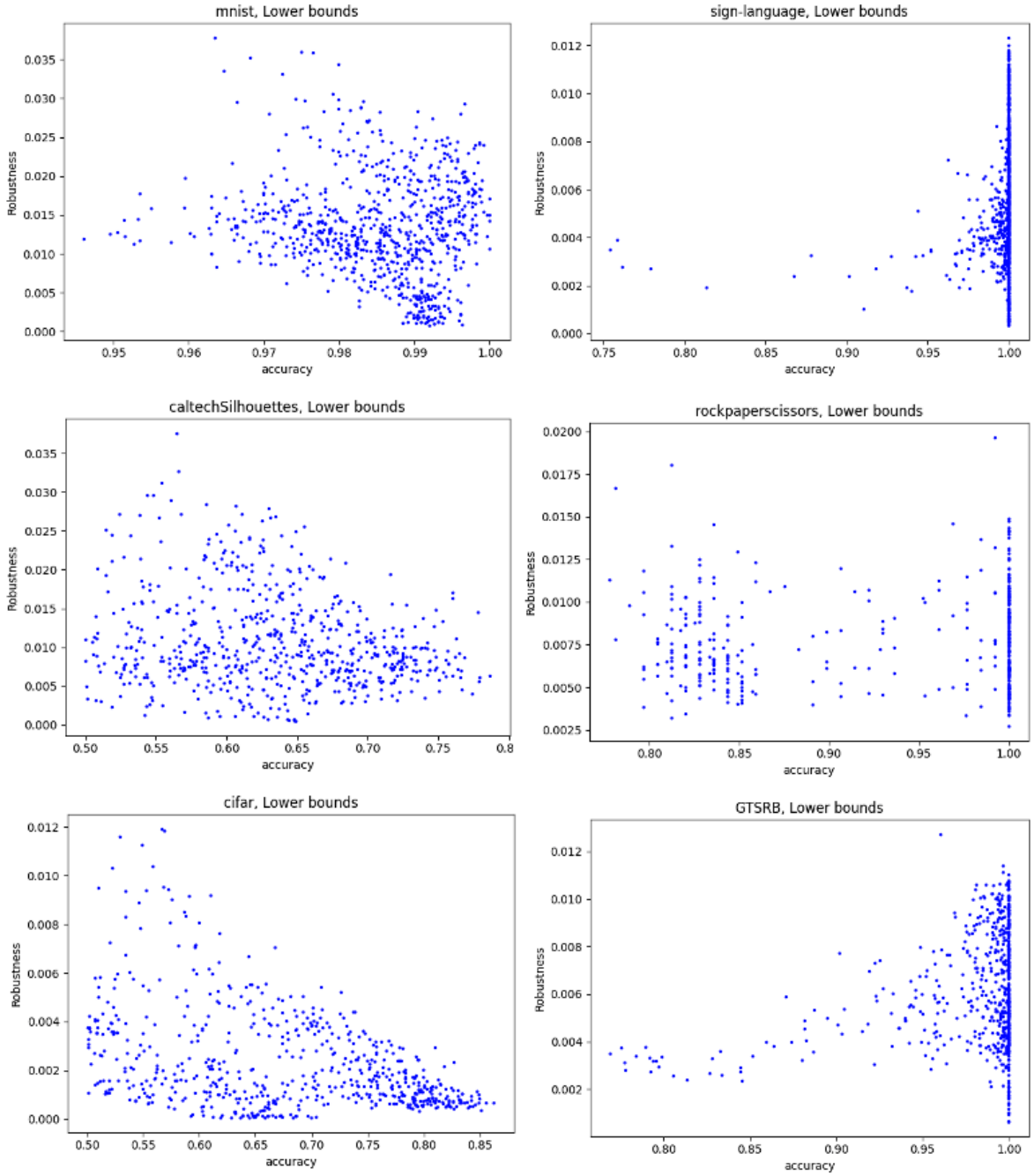


Figure 5.1: Lower bounds of the different datasets plotted against the accuracy.

negatively for all three norms on the CS and Cifar datasets as well, with statistically significant p-values. Because of this, we can reject NH1 and accept H1.

	L_∞		L_2		L_1	
Dataset	Coefficient	P-value	Coefficient	P-value	Coefficient	P-value
MNIST	-0.59152	0.0000	-0.73637	0.0000	-0.66927	0.0000
SL	-0.52772	0.0000	-0.64872	0.0000	-0.58016	0.0000
RSP	-0.69719	0.0000	-0.49804	0.0000	-0.38221	0.0000
GTSRB	-0.77327	0.0000	-0.80504	0.0000	-0.68218	0.0000
Combined	**-0.43186**	**0.0000**	**-0.51825**	**0.0000**	**-0.43018**	**0.0000**

Table 5.1: The coefficient and p-values from linear correlation analyses between the neural networks' depth and the lower bound on the l_∞, L_2 and L_1 norm.

The effect of the neural networks' activation function on the lower bound

Table 5.2 shows the mean of the neural networks when the different activation functions are used and the p-value of the t-test between relu and the other activation functions. We see that the lower bound when using arctan and tanh is much smaller than when using relu, and that the t-test shows the difference is statistically significant. The average lower bound when using sigmoid is slightly higher than when using relu, although the difference is not statistically significant for the L_∞ norm. We cannot reject NH6 from the results as the differences are not statistically significant.

In the regression, relu is represented by the absence of the other activation functions. In table 5.3 and 5.4, we see that sigmoid increase the lower bound when used instead of relu, while arctan and tanh decrease the lower bound.

		L_∞		L_2		L_1	
Activation function	Dataset	Mean	P-value	Mean	P-value	Mean	P-value
Relu	Mnist	0.01807	-	0.09671	-	0.24789	-
	SL	0.00753	-	0.04941	-	0.14497	-
	RPS	0.00744	-	0.06902	-	0.27362	-
	GTSRB	0.00698	-	0.07381	-	0.28952	-
	Combined	**0.01034**	**-**	**0.07161**	**-**	**0.22662**	**-**
Arctan	Mnist	0.00923	0.0000	0.07775	0.0000	0.22619	0.0361
	SL	0.00315	0.0000	0.02929	0.0000	0.10022	0.0000
	RPS	0.00452	0.0000	0.07727	0.0877	0.37030	0.0004
	GTSRB	0.00445	0.0000	0.05406	0.0000	0.19226	0.0000
	Combined	**0.00590**	**0.0000**	**0.05443**	**0.0000**	**0.17688**	**0.0000**
Sigmoid	Mnist	0.01763	0.3306	0.11188	0.0000	0.31530	0.0000
	SL	0.00636	0.0000	0.04775	0.2494	0.15425	0.1276
	RPS	0.01159	0.0000	0.18103	0.0000	0.81428	0.0000
	GTSRB	0.00707	0.6589	0.07594	0.4181	0.27736	0.3843
	Combined	**0.01083**	**0.0673**	**0.08587**	**0.0000**	**0.28372**	**0.0000**
Tanh	Mnist	0.00898	0.0000	0.07527	0.0000	0.21769	0.0029
	SL	0.00318	0.0000	0.02917	0.0000	0.09854	0.0000
	RPS	0.00471	0.0000	0.07792	0.0517	0.35967	0.0008
	GTSRB	0.00426	0.0000	0.05001	0.0000	0.17214	0.0000
	Combined	**0.00575**	**0.0000**	**0.05286**	**0.0000**	**0.16972**	**0.0000**

Table 5.2: The mean lower bound when using the different activation functions on the l_∞, L_2 and L_1 norm. The p-values come from a t-test between the mean lower bound of neural networks using relu and neural networks using the activation function in question.

Caltech 101 silhouettes						
	L_∞		L_2		L_1	
	Coefficient	P-value	Coefficient	P-value	Coefficient	P-value
Bias	0.03412	0.0000	0.18850	0.0000	0.45561	0.0000
Accuracy	-0.01697	0.0000	-0.13209	0.0000	-0.44145	0.0000
Depth	-0.00305	0.0000	-0.01907	0.0000	-0.05162	0.0000
Filter	-0.00001	0.3547	-0.00019	0.0000	-0.00089	0.0000
Arctan	-0.00127	0.0063	0.00849	0.0199	0.03545	0.0039
Sigmoid	0.00253	0.0000	0.04656	0.0000	0.19225	0.0000
Tanh	-0.00172	0.0001	0.00475	0.1770	0.02581	0.0288

Table 5.3: The coefficient and p-values from linear regression of the lower bounds for neural networks trained on the CS set.

Cifar						
	L_∞		L_2		L_1	
	Coefficient	P-value	Coefficient	P-value	Coefficient	P-value
Bias	0.01416	0.0000	0.16034	0.0000	0.57363	0.0000
Accuracy	-0.01221	0.0000	-0.15834	0.0000	-0.62755	0.0000
Depth	-0.00076	0.0000	-0.00895	0.0000	-0.03613	0.0000
Filter	0.00001	0.0000	0.00012	0.0000	0.00050	0.0000
Arctan	-0.00097	0.0000	-0.01105	0.0000	-0.04784	0.0000
Sigmoid	0.00097	0.0000	0.01286	0.0000	0.05894	0.0000
Tanh	-0.00104	0.0000	-0.01219	0.0000	-0.05277	0.0000

Table 5.4: The coefficient and p-values from linear regression of the lower bounds for neural networks trained on the Cifar set.

5.2.2 C&W upper bound

Only some of the data from the regression analyses using the C&W based upper bounds as the dependant variable are listed in this section. For a complete list, see appendix section B.

The effect of the neural networks' depth on the C&W based upper bound

In the correlation analyses in table 5.5 we see that the depth is positively correlated with the C&W upper bound on the L_∞ norm, less so on the L_2 norm and negatively correlated on the L_1 norm. For RPS, the depth is negatively correlated with the C&W upper bound on the L_2 norm. From the table, we see that when we combine the data from the different datasets, the correlation between depth and the C&W upper bound is not statistically significant on the L_1 norm. Because of this, we cannot reject NH2.

For the regression analyses in table 5.7 and 5.8, we see that the depth is negatively correlated with the C&W upper bound for all norms on the Cifar dataset.

	L_∞		L_2		L_1	
Dataset	Coefficient	P-value	Coefficient	P-value	Coefficient	P-value
MNIST	0.27709	0.0000	0.07917	0.0410	-0.30177	0.0000
SL	0.33206	0.0000	0.10822	0.0010	-0.05350	0.1109
RSP	0.08211	0.1019	-0.12942	0.0173	-0.27169	0.0000
GTSRB	0.43972	0.0000	0.15081	0.0003	-0.28360	0.0000
Combined	**0.15268**	**0.0000**	**0.06817**	**0.0007**	**0.01196**	**0.5555**

Table 5.5: The coefficient and p-values from linear correlation analyses between depth and the C&W upper bound on the l_∞, L_2 and L_1 norm.

The effect of the neural networks' activation function on the C&W based upper bound

Table 5.6 shows the mean C&W upper bound of the neural networks when the different activation functions are used and the p-value of the t-test between relu and the other activation functions. We see that the lower bound when using arctan, tanh, and sigmoid is smaller than when using relu, and that the t-test shows the difference is statistically significant. In the regression of CS and Cifar in table 5.7 and 5.8, sigmoid has a positive impact on the C&W upper

bound when compared to relu, while arctan and tanh have a negative impact on the C&W upper bound. Since the values from the regression indicate that the more non-linear activation functions are both have a more and less positive on the C&W upper bound, the NH7 hypothesis cannot be rejected.

		L_∞		L_2		L_1	
Activation function	Dataset	Mean	P-value	Mean	P-value	Mean	P-value
Relu	Mnist	0.08265	-	0.92240	-	6.66316	-
	SL	0.02226	-	0.11828	-	2.57310	-
	RPS	0.02626	-	1.39991	-	28.27362	-
	GTSRB	0.02837	-	0.53838	-	8.68137	-
	Combined	**0.03682**	**-**	**0.65369**	**-**	**10.06971**	**-**
Arctan	Mnist	0.04181	0.0000	0.40631	0.0000	5.31840	0.0000
	SL	0.01080	0.0000	0.03204	0.0000	1.40806	0.0000
	RPS	0.01501	0.0000	0.68434	0.0000	21.42987	0.0000
	GTSRB	0.01644	0.0000	0.18714	0.0000	4.57143	0.0000
	Combined	**0.02375**	**0.0000**	**0.24481**	**0.0000**	**5.19733**	**0.0000**
Sigmoid	Mnist	0.06561	0.0000	0.72780	0.0000	6.24042	0.0318
	SL	0.01611	0.0000	0.07308	0.0000	2.12608	0.0000
	RPS	0.02253	0.0000	1.30038	0.2437	28.23207	0.9688
	GTSRB	0.02224	0.0000	0.35017	0.0000	7.12563	0.0000
	Combined	**0.03260**	**0.0011**	**0.44413**	**0.0000**	**7.31726**	**0.0000**
Tanh	Mnist	0.04191	0.0000	0.45300	0.0000	5.27453	0.0000
	SL	0.01088	0.0000	0.03286	0.0000	1.39086	0.0000
	RPS	0.01615	0.0000	0.73164	0.0000	22.84184	0.0000
	GTSRB	0.01544	0.0000	0.16724	0.0000	4.38462	0.0000
	Combined	**0.02343**	**0.0000**	**0.26180**	**0.0000**	**5.20923**	**0.0000**

Table 5.6: The mean C&W upper bound when using the different activation functions on the l_∞, L_2 and L_1 norm. The p-values come from a t-test between the mean C&W upper bound of neural networks using relu and neural networks using the activation function in question.

Caltech 101 silhouettes						
	L_∞		L_2		L_1	
	Coefficient	P-value	Coefficient	P-value	Coefficient	P-value
Bias	0.08467	0.0000	1.40068	0.0000	11.02496	0.0000
Accuracy	-0.07539	0.0000	-2.25382	0.0000	-11.77071	0.0000
Depth	0.00111	0.1145	0.01518	0.4178	-0.54151	0.0000
Filter	0.00006	0.1648	0.00157	0.1992	-0.00417	0.5207
Arctan	-0.01862	0.0000	-0.37565	0.0000	-0.22600	0.6328
Sigmoid	0.00505	0.0289	0.23417	0.0002	1.22411	0.0002
Tanh	-0.01847	0.0000	-0.31045	0.0003	-0.40771	0.3706

Table 5.7: The coefficient and p-values from linear regression of the C&W upper bounds for neural networks trained on the CS set.

Cifar						
	L_∞		L_2		L_1	
	Coefficient	P-value	Coefficient	P-value	Coefficient	P-value
Bias	0.01982	0.0000	0.50961	0.0000	13.19607	0.0000
Accuracy	-0.01746	0.0000	-0.53687	0.0000	-12.61983	0.0000
Depth	-0.00042	0.0000	-0.01927	0.0000	-0.72953	0.0000
Filter	0.00001	0.0001	0.00063	0.0000	0.02519	0.0032
Arctan	-0.00213	0.0000	-0.04941	0.0000	-0.07225	0.9036
Sigmoid	0.00190	0.0000	0.05023	0.0000	1.04111	0.0059
Tanh	-0.00212	0.0000	-0.04943	0.0000	-0.58483	0.3150

Table 5.8: The coefficient and p-values from linear regression of the C&W upper bounds for neural networks trained on the Cifar set.

5.2.3 HSJA upper bound

Only some of the data from the regression analyses using the HSJA upper bound as the dependant variable are listed in this section. For a complete list see appendix section C.

The effect of the neural networks' depth on the HSJA upper bound

We see from table 5.9 that the neural networks' depth is positively correlated with the HSJA upper bound on the L_∞ norm. On the L_2 norm, the only statistically significant coefficient shows that the depth is negatively correlated with the HSJA upper bound. From the table, we see that when we combine the data from the different datasets, the correlation between depth and the HSJA upper bound is not statistically significant on the L_2 norm. Because of this, we cannot reject NH3.

From the regression analysis in table 5.11 and 5.12, we see that the depth has a negative impact on the CS and Cifar data sets on both the L_∞ norm and the L_2 norm.

	L_∞		L_2	
Dataset	Coefficient	P-value	Coefficient	P-value
MNIST	0.28794	0.0000	-0.04949	0.1096
SL	0.35431	0.0000	0.03959	0.1893
RSP	-0.05886	0.2856	-0.21808	0.0001
GTSRB	0.25217	0.0000	0.00775	0.8462
Combined	**0.16425**	**0.0000**	**0.00768**	**0.6698**

Table 5.9: The coefficient and p-values from linear correlation analyses between depth and the HSJA upper bound on the l_∞, L_2 and L_1 norm.

The effect of the neural networks' activation function on the HSJA upper bound

Table 5.10 shows the mean HSJA upper bound of the neural networks when the different activation functions are used and the p-value of the t-test between relu and the other activation functions. We that the lower bound when using arctan, tanh, and sigmoid is smaller than when using relu, and that the t-test shows the difference is statistically significant. In the regression of Cifar in table 5.11 and 5.12, sigmoid has a positive impact on the HSJA upper bound when compared to relu, while arctan and tanh have a negative impact on the HSJA upper bound. Since the

values from the regression indicate that the more non-linear activation functions are both have a more and less positive on the HSJA upper bound, the NH8 hypothesis cannot be rejected.

		L_∞		L_2	
Activation function	Dataset	Mean	P-value	Mean	P-value
Relu	Mnist	0.09418	-	1.10281	-
	SL	0.02385	-	0.14003	-
	RPS	0.07576	-	2.61424	-
	GTSRB	0.04270	-	0.62663	-
	Combined	**0.05573**	**-**	**0.95679**	**-**
Arctan	Mnist	0.05603	0.0000	0.57620	0.0000
	SL	0.01192	0.0000	0.05079	0.0000
	RPS	0.06592	0.0715	1.55991	0.0000
	GTSRB	0.03650	0.1463	0.36724	0.0000
	Combined	**0.03633**	**0.0000**	**0.39351**	**0.0000**
Sigmoid	Mnist	0.07437	0.0000	0.89403	0.0000
	SL	0.01744	0.0000	0.10446	0.0000
	RPS	0.10946	0.0000	3.17473	0.0057
	GTSRB	0.03299	0.0008	0.53447	0.0004
	Combined	**0.04789**	**0.0000**	**0.72429**	**0.0000**
Tanh	Mnist	0.05441	0.0000	0.54569	0.0000
	SL	0.01166	0.0000	0.04851	0.0000
	RPS	0.06467	0.0271	1.60101	0.0000
	GTSRB	0.03321	0.0171	0.30962	0.0000
	Combined	**0.03478**	**0.0000**	**0.37187**	**0.0000**

Table 5.10: The mean HSJA upper bound when using the different activation functions on the l_∞, L_2 and L_1 norm. The p-values come from a t-test between the mean HSJA upper bound of neural networks using relu and neural networks using the activation function in question.

Caltech 101 silhouettes				
	L_∞		L_2	
	Coefficient	P-value	Coefficient	P-value
Bias	0.08220	0.0000	1.40228	0.0000
Accuracy	-0.04893	0.0009	-1.16000	0.0002
Depth	-0.00227	0.0001	-0.13968	0.0000
Filter	0.00020	0.0000	0.00127	0.1685
Arctan	-0.00694	0.0135	-0.01012	0.8643
Sigmoid	-0.02230	0.0000	-0.14500	0.0004
Tanh	-0.00999	0.0002	-0.11030	0.0532

Table 5.11: The coefficient and p-values from linear regression of the HSJA upper bound for neural networks trained on the CS set.

Cifar				
	L_∞		L_2	
	Coefficient	P-value	Coefficient	P-value
Bias	0.02112	0.0000	0.69728	0.0000
Accuracy	-0.01499	0.0388	-0.69512	0.0000
Depth	-0.00021	0.5739	-0.03042	0.0000
Filter	0.00002	0.4564	0.00060	0.0062
Arctan	-0.00189	0.2730	-0.05714	0.0002
Sigmoid	0.00321	0.0033	0.06643	0.0000
Tanh	-0.00170	0.3122	-0.05696	0.0001

Table 5.12: The coefficient and p-values from linear regression of the HSJA upper bound for neural networks trained on the Cifar set.

5.2.4 BIM success rate

Only some of the data from the analyses regression using the BIM success rate as the dependant variable are listed in this section. For a complete list see appendix section D.

The effect of depth on the BIM success rate

From table 5.13, we see that the depth is positively correlated with the BIM success rate and has a statistically significant p-value for all the datasets in the correlation analyses, although the coefficient is not high. However, in the regression analyses listed in table 5.15, we see that for the CS and Cifar datasets, the depth of the neural network is negatively correlated with the BIM success rate, with statistically significant p-values. Because of this, we cannot reject NH4 as the different datasets behave differently.

Dataset	Coefficient	p-value
MNIST	0.3508	0.0000
SLM	0.4275	0.0000
RPS	0.1934	0.0030
GTSRB	0.3594	0.0000
Combined	**0.34554**	**0.0000**

Table 5.13: The coefficient and p-values from linear correlation analyses between the number of depth in the neural networks and their BIM success rate.

The effect of the activation functions on the BIM success rate

Table 5.14 shows the mean BIM success rate of the neural networks when the different activation functions are used and the p-value of the t-test between relu and the other activation functions. We that the lower bound when using arctan, tanh, and sigmoid is smaller than when using relu, and that the t-test shows the difference is statistically significant. In the regression of Cifar in table 5.15, sigmoid has a positive impact on the BIM success rate when compared to relu, while arctan and tanh have a negative impact. Since the values from the regression indicate that the more non-linear activation functions are both have a more and less positive on the BIM success rate, the NH9 hypothesis cannot be rejected.

	Dataset	Relu	Arctan	Sigmoid	Tanh
Coef	MNIST	0.8387	0.4240	0.7558	0.4075
	SLM	0.6514	0.2350	0.3934	0.2024
	RPS	0.5301	0.0314	0.4000	0.1797
	GTSRB	0.5643	0.3377	0.5664	0.3035
	Combined	**0.65554**	**0.34059**	**0.58246**	**0.31928**
P-value	MNIST	-	0.0000	0.0000	0.0000
	SLM	-	0.0000	0.0000	0.0000
	RPS	-	0.0000	0.0107	0.0000
	GTSRB	-	0.0000	0.9356	0.0000
	Combined	**-**	**0.0000**	**0.0000**	**0.0000**

Table 5.14: The mean BIM success rate when using the different activation functions on the l_∞, L_2 and L_1 norm. The p-values come from a t-test between the mean BIM success rate of neural networks using relu and neural networks using the activation function in question.

	Dataset	Bias	Original accuracy	Depth	Filters	Arctan	Sigmoid	Tanh
Coef	CS	0.6274	-0.2419	-0.0142	0.0009	-0.1991	-0.2229	-0.1952
	Cifar	0.5918	-0.4990	-0.0275	0.0005	-0.1414	0.2216	-0.1636
P-value	CS	0.0000	0.0328	0.0010	0.0009	0.0000	0.0000	0.0000
	Cifar	0.0000	0.0007	0.0001	0.2313	0.0000	0.0000	0.0000

Table 5.15: The coefficient and p-values from linear regression of the BIM success rate for neural networks trained on the cifar and CS datasets.

5.2.5 MIM success rate

Only some of the data from the regression analyses using the MIM success rate as the dependant variable are listed in this section. For a complete list see appendix section E.

The effect of depth on the MIM success rate

From table 5.16, we see that the depth is positively correlated with the MIM success rate and has a statistically significant p-value for all the datasets in the correlation analyses. However, in the regression analyses listed in table 5.18, we see that for the Cifar dataset, the depth of the neural network is negatively correlated with the MIM success rate, with a statistically significant p-value. Because of this, we cannot reject NH5 as the different datasets behave differently.

Dataset	Coefficient	p-value
MNIST	0.3647	0.0000
SLM	0.3712	0.0000
RPS	0.2293	0.0004
GTSRB	0.2450	0.0000

Table 5.16: The coefficient and p-values from linear correlation analyses between the number of depth in the neural networks and their MIM success rate.

The effect of the activation functions on the MIM success rate

Table 5.17 shows the mean MIM success rate of the neural networks when the different activation functions are used and the p-value of the t-test between relu and the other activation functions. We that the lower bound when using arctan, tanh, and sigmoid is smaller than when using relu, and that the t-test shows the difference is statistically significant. In the regression of Cifar in table 5.18, sigmoid has a positive impact on the MIM success rate when compared to relu, while arctan and tanh have a negative impact on the MIM success rate. Since the values from the regression indicate that the more non-linear activation functions are both have a more and less positive on the MIM success rate, the NH8 hypothesis cannot be rejected.

	Dataset	Relu	Arctan	Sigmoid	Tanh
Coef	MNIST	0.8560	0.4378	0.7736	0.4355
	SLM	0.6217	0.2820	0.4778	0.3163
	RPS	0.6284	0.0924	0.4185	0.2623
	GTSRB	0.6324	0.4334	0.6043	0.4072
	Combined	**0.67545**	**0.35577**	**0.60079**	**0.37445**
P-value	MNIST	-	0.0000	0.0000	0.0000
	SLM	-	0.0000	0.0000	0.0000
	RPS	-	0.0000	0.0000	0.0000
	GTSRB	-	0.0000	0.1607	0.0000
	Combined	**-**	**0.0000**	**0.0000**	**0.0000**

Table 5.17: The mean MIM success rate when using the different activation functions on the l_∞, L_2 and L_1 norm. The p-values come from a t-test between the mean MIM success rate of neural networks using relu and neural networks using the activation function in question.

	Dataset	Bias	Original accuracy	Depth	Filters	Arctan	Sigmoid	Tanh
Coef	CS	0.6289	-0.2287	-0.0013	0.0008	-0.2062	-0.1959	-0.2042
	Cifar	0.7371	-0.5117	-0.0328	0.0011	-0.1470	0.1414	-0.1650
P-value	CS	0.0000	0.0153	0.7091	0.0003	0.0000	0.0000	0.0000
	Cifar	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

Table 5.18: The coefficient and p-values from linear regression of the MIM success rate for neural networks trained on the cifar and CS datasets.

5.2.6 Summary of results concerning RQ1 and RQ2

The effect of the neural networks' depth on their robustness

The depth did not affect the lower bound any differently when considering the distance norms used, while both the C&W upper bound and the HSJA upper bound had a more negative correlation with the depth when the norm decreased. Only the lower bound and the C&W upper bound were used to measure the L_1 robustness, and in both cases, there was a negative correlation between the measured robustness and the depth. For the L_2 norm, the lower bound was negatively correlated with the depth, the BIM success rate did not have statistically significant p-values,

and the C&W upper bound was positively correlated with the depth. On the L_∞ norm, the C&W upper bound, the BIM success rate, and the MIM success rate was positively correlated with the depth for all datasets except cifar and CS, while the lower bound was negatively correlated for all datasets. In table 5.19 we see that of the null hypotheses concerning the hyperparameters effect on the robustness from section 4.2.1, only NH1 could safely be rejected.

Hypothesis	Rejected
NH1	Yes
NH2	No
NH3	No
NH4	No
NH5	No

Table 5.19: The hypotheses concerning the effect of the depth of the neural networks on their robustness.

The effect of the neural networks' activation function on their robustness

From the results, it is clear that neural networks using arctan and tanh as activation function has lower robustness for all estimation methods than those using relu, and that neural networks using sigmoid have robustness much closer to those using relu. When using the sigmoid function, the neural networks had higher lower bounds than when using relu. However, this was not statistically significant on the L_∞ norm.

None of the alternative hypotheses concerning the activation functions could be accepted, although H7, H8, H9, and H10 could be accepted if the results from the regression of the Cifar and CS datasets are removed. If so, then relu has a positive effect on the HJSA upper bound, C&W upper bound, BIM success rate, and MIM success rate when compared to the non-linear activation functions.

Hypothesis	Rejected
NH6	No
NH7	No
NH8	No
NH9	No
NH10	No

Table 5.20: The hypotheses concerning the effect of the activation functions of the neural networks on their robustness.

5.3 Results from exploring RQ3

To answer **RQ3**, the effect of the depth and the number of filters, on the neural networks' robustness were tested as described in section 4.2.3. As the effect of the depth on the robustness was covered when testing **RQ1** and **RQ2**, only the number of filters considered was considered in the following section. As with **RQ1** and **RQ2**, ten regression analyses on six data sets, and ten correlation analyses on four data sets were conducted.

5.3.1 The effect of the neural networks' width on the lower bound

From the correlation analyses in table 5.21, we see that the correlation coefficients with statistical significance are both positive and negative, but have high p-values. All the coefficients from the regression analyses listed in table 5.3 and 5.4. When all the datasets are combined, we see that the coefficients on the L_2 norm and the L_1 norm are statistically significant, but also very low. We see that in some cases, such as with MNIST, the correlation gives high p-values, while the regression gives low p-values. This indicates that we must adjust for the other variables used in the regression. From the results of the regression analyses, we see that the number of filters affects the lower bound both negatively and positively. The NH11 hypothesis cannot be rejected from the results as the correlations on the L_∞ norm is not significant, and the regression coefficients are both positive and negative.

	L_∞		L_2		L_1	
Dataset	Coefficient	P-value	Coefficient	P-value	Coefficient	P-value
MNIST	0.04947	0.0673	-0.01701	0.5339	-0.03326	0.2257
SL	0.05510	0.0363	-0.02665	0.3119	-0.03782	0.1532
RSP	0.17801	0.0003	-0.18513	0.0005	-0.25039	0.0000
GTSRB	0.10571	0.0068	0.00138	0.9719	0.02488	0.5266
Combined	**0.01592**	**0.3222**	**-0.09754**	**0.0000**	**-0.12692**	**0.0000**

Table 5.21: The coefficient and p-values from linear correlation analyses between the number of filters and the lower bound on the l_∞, L_2 and L_1 norm.

5.3.2 The effect of the neural networks' width on the C&W upper bound

From the correlation analyses in table 5.22, we see that the C&W upper bound is positively correlated with the number of filters for all datasets and all norms. When we combine the results from the datasets, the coefficients for the L_2 and L_1 norm change sign. This could be because the correlation is weak. Because of this, we cannot conclude that the width correlates with the C&W upper bound, and NH12 is not rejected. We also see in table 5.7 and 5.8 that the number of filters affects the C&W upper bound positively on the Cifar and CS datasets with the exception of CS on the L_1 norm.

	L_∞		L_2		L_1	
Dataset	Coefficient	P-value	Coefficient	P-value	Coefficient	P-value
MNIST	0.18628	0.0000	0.23345	0.0000	0.19848	0.0000
SL	0.28349	0.0000	0.32064	0.0000	0.31573	0.0000
RSP	0.22419	0.0000	0.14691	0.0068	0.02119	0.7044
GTSRB	0.29060	0.0000	0.36014	0.0000	0.34968	0.0000
Combined	**0.10059**	**0.0000**	**-0.04157**	**0.0379**	**-0.17613**	**0.0000**

Table 5.22: The coefficient and p-values from linear correlation analyses between the number of filters and the C&W upper bound on the l_∞, L_2 and L_1 norm.

5.3.3 The effect of the neural networks' width on the HSJA upper bound

From the correlation analyses in table 5.23, we see that the HSJA upper bound is positively correlated with the number of filters for all datasets and all norms except for the RPS dataset.

When we combine the correlations, we see that the p-value of the $L\infty$ norm is not statistically significant, and thus we cannot reject NH13. We also see that the statistically significant coefficient of the L_2 norm is very low. In table 5.11 and 5.12, we see that the number of filters affects the HSJA upper bound positively on the Cifar and CS datasets as well.

	L_∞		L_2	
Dataset	Coefficient	P-value	Coefficient	P-value
MNIST	0.17374	0.0000	0.18153	0.0000
SL	0.29375	0.0000	0.17561	0.0000
RSP	-0.26031	0.0000	0.01767	0.7551
GTSRB	0.22128	0.0000	0.37827	0.0000
Combined	**-0.00194**	**0.9123**	**-0.08724**	**0.0000**

Table 5.23: The coefficient and p-values from linear correlation analyses between the number of filters and the HSJA upper bound on the l_∞, L_2 and L_1 norm.

5.3.4 The effect of the neural networks' width on the BIM success rate

From table 5.24, we see that there is a positive correlation between the number of filters and the BIM success rate. From the regression analyses listed in table 5.15, we see that the number of filters has a positive effect on the CS and Cifar datasets as well. Because of this, we can reject NH14 and accept H14. However, we see that the coefficient is also low, which inticates a weak correlation.

Dataset	Coefficient	P-value
MNIST	0.1149	0.0001
SLM	0.2794	0.0000
RPS	0.2620	0.0001
GTSRB	0.2447	0.0000
Combined	**0.15110**	**0.0000**

Table 5.24: The coefficient and p-values from linear correlation analyses between the number of filters and the BIM success rate for neural networks trained on various datsets.

5.3.5 The effect of the neural networks' width on the MIM success rate

From table 5.25, we see that there is a positive correlation between the number of filters and the MIM success rate. From the regression analyses listed in table 5.18, we see that the number of filters has a positive effect on the CS and Cifar datasets as well. Because of this, we can reject NH14 and accept H14. However, we see that the coefficient is also low, which indicates a weak correlation.

Dataset	Coefficient	P-value
MNIST	0.0984	0.0111
SLM	0.0777	0.0304
RPS	0.2336	0.0003
GTSRB	0.0698	0.0979
Combined	**0.08985**	**0.0000**

Table 5.25: The coefficient and p-values from linear correlation analyses between the number of filters and the MIM success rate for neural networks trained on various datsets.

5.3.6 Summary of results concerning RQ3

In table 5.26, we see that there is only the NH14 and NH15 hypotheses from section 4.2.3 which were rejected. For the other hypotheses, the p-values were too high, or the correlation differed between the norms. We also see that for all the coefficients with statistical significance, the correlation is low.

Hypothesis	Rejected
NH11	No
NH12	No
NH13	No
NH14	Yes
NH15	Yes

Table 5.26: The hypotheses concerning the effect of the width of the neural networks on their robustness.

5.4 Results from exploring RQ4

This experiment was conducted to answer **RQ4** to see if it is possible to push the decision boundary of the neural networks. In table 5.27, we see the average lower bounds for the neural networks trained on natural examples, and in table 5.28 and 5.29 we see the average lower bounds for the same networks trained on adversarial examples. The ϵ is set to roughly 5 and 10 times the mean lower bounds measured for the neural networks without adversarial training. For the largest epsilons, Cifar and CS are not shown as the neural networks did not converge during the training. From the tables, we see a substantial increase in the average lower bound when the neural networks are trained on adversarial neural networks. We also see that the accuracy on natural examples, the examples not generated by an adversarial attack algorithm, stays roughly the same when the neural networks are trained against adversarial examples. From table 5.30, we see that the ϵ used when training neural networks on the BIM attack is strongly correlated with the lower bound for all datasets. Additionally, the coefficients have very low p-values. Because of this NH16 can be rejected and H16 accepted.

Dataset	Lower bound	Accuracy
MNIST	0.0183	99.1%
SLM	0.0062	100.0%
CS	0.0096	68.6%
RPS	0.0078	100.0%
Cifar	0.0017	76.2%
GTSRB	0.0068	99.5%

Table 5.27: The avrage lower bound and accuracy for the nerual networks without adversarial training.

Dataset	ϵ	Average lower bound	Robustness increase	Average natural accuracy	Average adversarial accuracy
MNIST	0.1	0.0761	417%	98.8%	96.8%
SLM	0.03	0.0187	302%	100.0%	99.6%
CS	0.05	0.0440	460%	67.4%	55.3%
RPS	0.04	0.0264	337%	96.3%	89.2%
Cifar	0.01	0.0080	465%	62.5%	45.3%
GTSRB	0.05	0.0283	414%	99.2%	90.1%

Table 5.28: The ϵ used for the adversarial training and the average lower bound and accuracy for the nerual networks.

Dataset	ϵ	Average lower bound	Robustness increase	Average natural accuracy	Average adversarial accuracy
MNIST	0.2	0.1354	707%	98.6%	94.6%
SLM	0.06	0.0223	360%	99.9%	93.0%
RPS	0.08	0.0422	671%	97.9%	88.9%
GTSRB	0.1	0.0379	552%	98.7%	78.7%

Table 5.29: The ϵ used for the adversarial training and the average lower bound and accuracy for the nerual networks.

Dataset	Coefficient	P-value
MNIST	0.9651	1.456e-39
SL	0.7203	9.816e-13
CS	0.9462	1.061e-21
RSP	0.9391	2.952e-26
Cifar	0.6179	1.276e-04
GTSRB	0.7847	5.567e-16

Table 5.30: The coefficients and p-values of the correlation between the ϵ used when training neural networks and the neural networks' lower bounds.

Chapter 6

Discussion

6.1 Changing the hyperparameters is not enough to create robust neural networks

From the results, we see that the hyperparameters alone are not enough to create sufficiently robust neural networks. If we look at GTSRB and the L_∞ norm as an example, then according to the coefficients from the lower bound regression, the largest lower bound within the hyperparameters used in the experiment is 0.010, a bit less than twice the average lower bound of all the networks on the GTSRB dataset. The lower bound regression is listed in the appendix table A.4 and the average lower bounds for the neural networks are listed in the appendix, table A.5. The highest C&W upper bound we can get using the coefficients and the hyperparameters from the regression is 0.049, slightly higher than twice the mean C&W upper bound. The C&W regression is listed in the appendix, table B.4. We see the same for the other datasets with the maximum lower and upper bounds from optimal hyperparameters being close to twice the mean bounds for the neural networks. It should be noted that the parameters used to maximize the lower bound are not the same as those used to maximize the upper bound. The hyperparameters that maximize both the upper and lower bounds would yield much lower bounds. In figure 6.1 we see adversarial attacks on the GTSRB set using the BIM attack with ϵ equal to 0.01 and 0.05 on the L_∞ norm. As we can see, the images are quite similar, and a self-driving car fooled by the $\epsilon = 0.05$ attack would not be considered safe. This is a bit better on the data sets that have inherently more robust neural networks such as MNIST, where the neural networks have a maximum upper bound of 0.111, given the hyperparameters used in the experiment and the coefficients from the regression listed in appendix table B.1. However, it is also worse for the data sets that have less robust neural networks such as Cifar and Sign language.

The results of Burkard and Lagesse (2019) show that the depth, width, and activation functions of the neural networks have a low impact on the neural networks' robustness. This is in line with my results, making it likely that these hyperparameters are unimportant for the robustness of neural networks.

6.2 Implication of the results of RQ1 and RQ2

To answer RQ1 and RQ2, the analyses in section 5.2 were conducted to determine how the depth and the activation function of the neural networks affect their robustness. If neural net-

Figure 6.1: Adversarial examples created using the BIM attack with ϵ equal to 0.01 and 0.05 on the L_∞ norm on five images from the GTSRB set. The upper row contains the original images, while the middle row contains the adversarial images with L_∞ distance 0.01, and the bottom row contains the adversarial images with L_∞ distance 0.05.

works are vulnerable to adversarial examples because they are too linear, making the neural networks deeper and using a more non-linear activation function should increase the robustness of the neural networks. The opposite should happen if the neural networks are vulnerable to adversarial examples because they are too non-linear.

6.2.1 Depth

From the results, we see there were only we see that only the NH1 hypothesis could safely be rejected. Thus, it cannot be concluded that the depth of neural networks correlates with their robustness. This aligns with the results of Su et al. (2018), who find that neural networks within the same architectural family have similar robustness regardless of depth. Since the best case hyperparameters did not result in sufficient robustness, and the result does not show correlation, it seems unlikely that the depth of neural networks affects their robustness to more than an insignificant extent.

6.2.2 Activation functions

From the results from section 5.2, none of the null hypotheses could be rejected unless the regression analyses of Cifar and CS were discarded. If they are discarded, the results indicate that using relu as activation function produces neural networks more robust than if the more non-linear activation functions are used for all estimation methods except for the lower bound. In the regression of Cifar and CS, the accuracy of the neural networks were used as an independent

variable to controll for the accuracy's effect on the robustness. Since the accuracy is correlated with the activation functions, there is multicollinearity in the regression analyses, which can affect the coefficients. Because of this, it is probable that the regression of Cifar and CS cannot be trusted. If the regressions are discarded, then the non-linearity hypothesis from 2.2.1 is more likely than the linearity hypothesis. However, the difference between the effect of relu and sigmoid on the robustness estimations is small. As discussed in section 6.1, the best case effect of all the hyperparameters is small. Also, the lower bound increased when the neural networks had sigmoid as activation function as opposed to relu. Because of this, the non-linearity hypothesis seems unlikely as well.

6.3 Implication of the results of RQ3

The analyses in section 5.3 were conducted to answer RQ3 and determine how the width affects the neural networks' robustness. If the neural networks' vulnerability stems from overfitting, increasing the depth and width should reduce the networks' robustness.

From the results, we see that only NH14 and NH15 could be rejected, and we cannot conclude that the width of neural networks affects their robustness. The results of Madry et al. (2017) showed that the width neural networks were positively correlated with their success rate against the BIM attack. This is in line with my findings. Still, I find that the correlation is weak, and I am unable to reproduce the correlation with the upper and lower bound robustness estimations.

The result of Su et al. (2018) show that the size of the neural networks model does not affect the robustness much. This fits better with my results, as the results with statistical significance show that the correlation is low. This, combined with the result discussed in section 6.1, that the best-case robustness from the hyperparameters is still low, makes it unlikely that the width of the neural networks affects their robustness much.

The results of Burkard and Lagesse (2019) show that the pool size of the max-pooling and dropout increased the robustness of neural networks. These parameters are often used to reduce overfitting of neural networks. However, if neural networks' vulnerability to adversarial examples stems from overfitting, then the width and depth should affect the robustness. Both my results and the results of Burkard and Lagesse (2019) show that this is not the case. Thus the overfitting hypothesis seems unlikely.

6.4 Implication of the results of RQ4

The experiment in section 5.4 was conducted to answer **RQ4**, to see how the training input affects the neural networks' robustness. From the results of the experiment, we see that the lower bound increases considerably on all datasets when the neural networks are trained on adversarial examples created by the BIM attack algorithm. For the GTSRB, the lower bound gets increased to 0.0379, which is almost four times larger than the lower bound of the best possible combination of hyperparameters. We also see that the ϵ of the attack is strongly correlated with lower bound, with statistically significant p-values. Because of this, the decision boundary hypothesis seems to be the most likely hypothesis.

6.5 How to use the decicion boundary hypothesis to create more robust neural networks

From the experiment, we see that we can make the neural networks more robust against adversarial examples by training on examples created by the BIM attack. Madry et al. (2017) have already shown that this method improves the neural networks' robustness against the FGSM, BIM, and C&W attacks. However, as described in section , it is not enough to prove robustness against existing attacks to prove robustness against all attacks. By showing that a certified lower bound of the neural networks increases when training on the BIM attack, I prove that this method is effective against any attack on the L_∞ norm.

Since neural networks' vulnerability to adversarial examples most likely is caused by the training input being to close to the decision boundary, training on any adversarial attack would increase the robustness, provided the attack is sufficiently strong. A downside with using the BIM attack is that it slows down training of the neural networks significantly. In my experiment, I used 20 steps of BIM, which required the gradients of the neural networks to be calculated 20 times before every training step. By using faster attacks, one could reduce the training time required to make robust neural networks.

We also see that training on adversarial examples creates a more difficult problem for the neural networks to learn. When the ϵ of the attack was increased, the neural networks were not able to converge on the Cifar and CS datasets, and the accuracy on the other datasets decreased. This means that we need stronger neural networks to ensure they converge when training on neural networks.

6.6 Threats to validity

6.6.1 Internal threats

In my experiments, I have used multiple datasets and methods to collect evidence for my conclusions. For the experiments concerning RQ1, RQ2, and RQ3, I conducted ten different regression analyses on six different datasets and ten correlation analyses on four datasets. The analyses gave statistically significant values for most of the analyses. However, since some of the analyses gave different results, I believe there a bias for the hyperparameters within some of the robustness estimation techniques. I do not consider this to be a problem as the main reason for using multiple robustness estimation methods was to detect bias, which is accounted for in my conclusion.

For the experiments concerning RQ4, I only used the lower bound as the robustness estimation technique. Since I used the same hyperparameters for all tests, I do not believe there is any bias. This belief is strengthened as the same results as I had were observed with a different robustness estimation technique by Madry et al. (2017). All the correlation analyses gave statistically significant p-values, which makes me certain there is enough evidence to support the claim that the decision boundary hypothesis is the most likely hypothesis to be true.

When testing RQ1, RQ2, and RQ3, the average estimated robustness for ten images was used to determine the robustness for each neural network in all estimation methods. Although this is too few examples to capture the whole dataset's robustness correctly, I expect the error was evened out over the multiple neural networks and that the correct trends were visible in the regression. Additionally, one might argue that estimated robustness for all ten images should

be used, and not just the average. In hindsight, this might have been better. However, I believe the trends among the hyperparameters would have been the same as I used a large number of neural networks.

6.6.2 External threats

For all my experiments, I have used six different datasets to ensure generalizability. Since my results repeated across the datasets, they will most likely generalize to other datasets as well. However, in my experiments, I only tested image sets. There is a possibility that other types of datasets, such as audio sets behave differently. On the other hand, this is unlikely if the decision boundary hypothesis is correct.

For my experiments, I have also only used CNNs and not other neural network architectures such as VGGNets or ResNets. It is possible that my results do not generalize to other neural network architectures.

Chapter 7

Conclusion and future work

7.1 Conclusion

To utilize the excellent performance of neural networks, we must be sure they are reliable and trustworthy. This is only possible as long as we know where adversarial examples stem from and that the neural networks we use are robust. In this paper, I have presented and tested four hypotheses as to why neural networks are vulnerable to adversarial examples.

The first hypothesis was that neural networks are vulnerable to adversarial examples because they are too linear. If this hypothesis is true, then making the neural networks more non-linear by either using a more non-linear activation function or applying more layers of the non-linear function should increase the robustness. From my results, it is difficult to conclude how the depth affects the robustness, as the results differ depending on the estimation method and L_p norm. However, we see that the arctan and tanh give much lower robustness than relu, even though they are more non-linear. Because of this, the linearity hypothesis from RQ1 seems unlikely.

The second hypothesis was that neural networks are vulnerable to adversarial examples because they are too non-linear. If this hypothesis is correct, making the neural networks more non-linear by either using a more non-linear activation function or applying more layers of the non-linear function should decrease the robustness. Although we see that arctan and tanh decrease the robustness compared to relu, we also see that sigmoid and relu give similar robustness, even though sigmoid is more non-linear than relu. When we look at the best-case scenario from the regression, we also see that there is not much robustness to be gained by changing the hyperparameters. This makes it unlikely that the non-linearity hypothesis from RQ2 is correct.

The third hypothesis was that neural networks are vulnerable to adversarial examples because they overfit. If this is true, increasing the neural networks' capacity too much, by making them too wide and deep should decrease their robustness. Since there is not much robustness to be gained by changing the hyperparameters, this hypothesis seems unlikely as well. Additionally, I hypothesize that the lower bound of CNN-cert gets looser on wide neural networks. If my hypothesis is true, the overfitting hypothesis is even more unlikely, as the width increased all the robustness estimations except for the lower bounds.

The fourth hypothesis was that neural networks are vulnerable to adversarial examples because the decision boundary is left to close to the training examples. In my results, I show that training on adversarial examples greatly increases the lower bound. I also show that the increase in the lower bound is strongly correlated with the ϵ of the attack used when training the neural

networks. This strengthens the decision boundary hypothesis and makes it the hypothesis most likely to be true. From the results regarding the decision boundary hypothesis, I also show that training on the BIM attack makes the neural networks robust to any attack, as the lower bound is an attack agnostic metric.

7.2 Future work

From my results, I think exploring the decision boundary and how it can be pushed in the right direction is the most interesting path ahead. I used 20 steps of the BIM attack when training on the adversarial examples, which lead to the neural networks taking much longer time to train. However, if only one step is used, the BIM attack turns in to the FGSM attack, which has been shown to create non-robust networks when used in training. It would be interesting to see what the optimal number of steps is, or if there is another attack that trains the neural networks against adversarial examples faster. Additionally, the BIM attack only uses the L_∞ norm. It would be interesting to see if it was possible to train neural networks that are robust on all norms.

Bibliography

, 1993. Regression with dummy variables. URL: https://methods.sagepub.com/book/regression-with-dummy-variables, doi:10.4135/9781412985628.

Athalye, A., Engstrom, L., Ilyas, A., Kwok, K., 2017. Synthesizing robust adversarial examples. arXiv:1707.07397.

Bojarski, M., Testa, D.D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., Zieba, K., 2016. End to end learning for self-driving cars. arXiv:1604.07316.

Boopathy, A., Weng, T.W., Chen, P.Y., Liu, S., Daniel, L., 2018. Cnn-cert: An efficient framework for certifying robustness of convolutional neural networks. arXiv:1811.12395.

Brendel, W., Rauber, J., Bethge, M., 2017. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. arXiv:1712.04248.

de la Bruère-Terreault, J., . URL: https://www.kaggle.com/drgfreeman/rockpaperscissors.

Burkard, C., Lagesse, B., 2019. Can intelligent hyperparameter selection improve resistance to adversarial examples? arXiv:1902.05586.

Carlini, N., Wagner, D., 2017. Towards evaluating the robustness of neural networks, in: 2017 IEEE Symposium on Security and Privacy (SP), pp. 39–57. doi:10.1109/SP.2017.49.

Chen, J., Jordan, M.I., Wainwright, M.J., 2019. Hopskipjumpattack: A query-efficient decision-based attack. arXiv:1904.02144.

Chen, P.Y., Sharma, Y., Zhang, H., Yi, J., Hsieh, C.J., 2017. Ead: Elastic-net attacks to deep neural networks via adversarial examples. arXiv:1709.04114.

Ciodaro, T., Deva, D., de Seixas, J.M., Damazio, D., 2012. Online particle detection with neural networks based on topological calorimetry information. Journal of Physics: Conference Series 368, 012030. URL: https://doi.org/10.1088%2F1742-6596%2F368%2F1%2F012030, doi:10.1088/1742-6596/368/1/012030.

Dahl, G.E., Stokes, J.W., Deng, L., Yu, D., 2013. Large-scale malware classification using random projections and neural networks, in: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 3422–3426.

Dong, Y., Liao, F., Pang, T., Su, H., Zhu, J., Hu, X., Li, J., 2018. Boosting adversarial attacks with momentum, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.

Esteva, A., Kuprel, B., Novoa, R.A., Ko, J., Swetter, S.M., Blau, H.M., Thrun, S., 2017. Dermatologist-level classification of skin cancer with deep neural networks. Nature 542, 115–118. URL: https://doi.org/10.1038/nature21056, doi:10.1038/nature21056.

Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T., Song, D., 2017. Robust physical-world attacks on deep learning models. arXiv:1707.08945.

Fei-Fei, L., Fergus, R., Perona, P., 2006. One-shot learning of object categories. IEEE Trans. Pattern Anal. Mach. Intell. 28, 594–611. URL: https://doi.org/10.1109/TPAMI.2006.79, doi:10.1109/TPAMI.2006.79.

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. MIT Press. http://www.deeplearningbook.org.

Goodfellow, I.J., Shlens, J., Szegedy, C., 2014. Explaining and harnessing adversarial examples. arXiv:1412.6572.

Graves, A., rahman Mohamed, A., Hinton, G., 2013. Speech recognition with deep recurrent neural networks. arXiv:1303.5778.

Grosse, K., Papernot, N., Manoharan, P., Backes, M., McDaniel, P., 2016. Adversarial perturbations against deep neural networks for malware classification. arXiv:1606.04435.

Gu, S., Rigazio, L., 2014. Towards deep neural network architectures robust to adversarial examples. arXiv:1412.5068.

He, K., Zhang, X., Ren, S., Sun, J., 2015a. Deep residual learning for image recognition. arXiv:1512.03385.

He, K., Zhang, X., Ren, S., Sun, J., 2015b. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. arXiv:1502.01852.

Helmstaedter, M., Briggman, K.L., Turaga, S.C., Jain, V., Seung, H.S., Denk, W., 2013. Connectomic reconstruction of the inner plexiform layer in the mouse retina. Nature 500, 168–174. URL: https://doi.org/10.1038/nature12346, doi:10.1038/nature12346.

Hinton, G., Vinyals, O., Dean, J., 2015. Distilling the knowledge in a neural network. arXiv:1503.02531.

Julian, K.D., Lopez, J., Brush, J.S., Owen, M.P., Kochenderfer, M.J., 2016. Policy compression for aircraft collision avoidance systems, in: 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC), pp. 1–10. doi:10.1109/DASC.2016.7778091.

Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J., 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. CoRR abs/1702.01135. URL: http://arxiv.org/abs/1702.01135, arXiv:1702.01135.

Kolter, J.Z., Wong, E., 2017. Provable defenses against adversarial examples via the convex outer adversarial polytope. CoRR abs/1711.00851. URL: http://arxiv.org/abs/1711.00851, arXiv:1711.00851.

Krizhevsky, A., Nair, V., Hinton, G., . Cifar-10 (canadian institute for advanced research) URL: http://www.cs.toronto.edu/~kriz/cifar.html.

Kurakin, A., Goodfellow, I., Bengio, S., 2016a. Adversarial examples in the physical world. arXiv:1607.02533.

Kurakin, A., Goodfellow, I., Bengio, S., 2016b. Adversarial machine learning at scale. arXiv:1611.01236.

Lecun, Y., Bottou, L., Bengio, Y., Haffner, P., 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE 86, 2278 – 2324. doi:10.1109/5.726791.

LeCun, Y., Cortes, C., 2010. MNIST handwritten digit database URL: http://yann.lecun.com/exdb/mnist/.

Levin, S., Wong, J.C., 2018. Self-driving uber kills arizona woman in first fatal crash involving pedestrian. URL: https://www.theguardian.com/technology/2018/mar/19/uber-self-driving-car-kills-woman-arizona-tempe.

Liu, Y., Chen, X., Liu, C., Song, D., 2016. Delving into transferable adversarial examples and black-box attacks. arXiv:1611.02770.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A., 2017. Towards deep learning models resistant to adversarial attacks. arXiv:1706.06083.

Moosavi-Dezfooli, S.M., Fawzi, A., Frossard, P., 2015. Deepfool: a simple and accurate method to fool deep neural networks. arXiv:1511.04599.

Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A., 2015a. The limitations of deep learning in adversarial settings. arXiv:1511.07528.

Papernot, N., McDaniel, P., Wu, X., Jha, S., Swami, A., 2015b. Distillation as a defense to adversarial perturbations against deep neural networks. arXiv:1511.04508.

Papernot, N., McDaniel, P.D., 2016. On the effectiveness of defensive distillation. CoRR abs/1607.05113. URL: http://arxiv.org/abs/1607.05113, arXiv:1607.05113.

Papernot, N., McDaniel, P.D., 2017. Extending defensive distillation. CoRR abs/1705.05264. URL: http://arxiv.org/abs/1705.05264, arXiv:1705.05264.

Papernot, N., McDaniel, P.D., Wu, X., Jha, S., Swami, A., 2015c. Distillation as a defense to adversarial perturbations against deep neural networks. CoRR abs/1511.04508. URL: http://arxiv.org/abs/1511.04508, arXiv:1511.04508.

Raghunathan, A., Steinhardt, J., Liang, P., 2018. Certified defenses against adversarial examples. CoRR abs/1801.09344. URL: http://arxiv.org/abs/1801.09344, arXiv:1801.09344.

Robertson, G., Lehmann, E.D., Sandham, W., Hamilton, D., 2011. Blood glucose prediction using artificial neural networks trained with the aida diabetes simulator: A proof-of-concept pilot study. JECE 2011, 2:2–2:2. URL: http://dx.doi.org/10.1155/2011/681786, doi:10.1155/2011/681786.

Rosenblatt, F., 1958. The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review , 65–386.

Rozsa, A., Gunther, M., Boult, T.E., 2016. Towards robust deep neural networks with bang. arXiv:1612.00138.

Shaham, U., Yamada, Y., Negahban, S., 2018. Understanding adversarial training: Increasing local stability of supervised models through robust optimization. Neurocomputing 307, 195–204. URL: http://dx.doi.org/10.1016/j.neucom.2018.04.027, doi:10.1016/j.neucom.2018.04.027.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., Hassabis, D., 2017. Mastering the game of go without human knowledge. Nature 550, 354–359. URL: https://doi.org/10.1038/nature24270, doi:10.1038/nature24270.

Simonyan, K., Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556.

SINGH, G., GEHR, T., MIRMAN, M., PÜSCHEL, M., VECHEV, M., 2018. Fast and effective robustness certification. URL: https://www.sri.inf.ethz.ch/publications/singh2018effective.

Singh, G., Gehr, T., Püschel, M., Vechev, M.T., 2019a. Boosting robustness certification of neural networks.

Singh, G., GEHR, T., PÜSCHEL, M., VECHEV, M., 2019b. An abstract domain for certifying neural networks. URL: https://eth-sri.github.io/publications/singh2019domain.

Stallkamp, J., Schlipsing, M., Salmen, J., Igel, C., 2012. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. Neural Networks , –URL: http://www.sciencedirect.com/science/article/pii/S0893608012000457, doi:10.1016/j.neunet.2012.02.016.

Su, D., Zhang, H., Chen, H., Yi, J., Chen, P.Y., Gao, Y., 2018. Is robustness the cost of accuracy? – a comprehensive study on the robustness of 18 deep image classification models. arXiv:1808.01688.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., 2014a. Going deeper with convolutions. arXiv:1409.4842.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S.E., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., 2014b. Going deeper with convolutions. CoRR abs/1409.4842. URL: http://arxiv.org/abs/1409.4842, arXiv:1409.4842.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R., 2013. Intriguing properties of neural networks. arXiv:1312.6199.

Taigman, Y., Yang, M., Ranzato, M., Wolf, L., 2014. Deepface: Closing the gap to human-level performance in face verification, in: 2014 IEEE Conference on Computer Vision and Pattern Recognition, pp. 1701–1708.

Tanay, T., Griffin, L., 2016. A boundary tilting persepective on the phenomenon of adversarial examples. arXiv:1608.07690.

Vargas, D.V., Kotyan, S., 2019. Robustness assessment for adversarial machine learning: Problems, solutions and a survey of current neural networks and defenses. arXiv:1906.06026.

Weng, T.W., Zhang, H., Chen, H., Song, Z., Hsieh, C.J., Boning, D., Dhillon, I.S., Daniel, L., 2018a. Towards fast computation of certified robustness for relu networks. arXiv:1804.09699.

Weng, T.W., Zhang, H., Chen, P.Y., Yi, J., Su, D., Gao, Y., Hsieh, C.J., Daniel, L., 2018b. Evaluating the robustness of neural networks: An extreme value theory approach. arXiv:1801.10578.

Yadron, D., Tynan, D., 2016. Tesla driver dies in first fatal crash while using autopilot mode. URL: https://www.theguardian.com/technology/2016/jun/30/tesla-autopilot-death-self-driving-car-elon-musk.

Zhang, H., Weng, T., Chen, P., Hsieh, C., Daniel, L., 2018. Efficient neural network robustness certification with general activation functions. CoRR abs/1811.00866. URL: http://arxiv.org/abs/1811.00866, arXiv:1811.00866.

Zhang, J., Li, C., 2019. Adversarial examples: Opportunities and challenges. IEEE Transactions on Neural Networks and Learning Systems , 1–16URL: http://dx.doi.org/10.1109/TNNLS.2019.2933524, doi:10.1109/tnnls.2019.2933524.

Appendix A

Lower bounds

A.1 Regression

MNIST						
	L_∞		L_2		L_1	
	Coefficient	P-value	Coefficient	P-value	Coefficient	P-value
Bias	0.02998	0.0000	0.13804	0.0000	0.24346	0.0000
Depth	-0.00292	0.0000	-0.02239	0.0000	-0.06479	0.0000
Filter	0.00001	0.0269	-0.00007	0.0458	-0.00035	0.0039
Arctan	-0.00880	0.0000	-0.01911	0.0000	-0.02216	0.0004
Sigmoid	-0.00084	0.0013	0.01143	0.0000	0.05588	0.0000
Tanh	-0.00904	0.0000	-0.02133	0.0000	-0.03056	0.0000

Table A.1: The coefficient and p-values from linear regression of the lower bounds for neural networks trained on the MNIST set.

Sign Language						
	L_∞		L_2		L_1	
	Coefficient	P-value	Coefficient	P-value	Coefficient	P-value
Bias	0.01054	0.0000	0.05369	0.0000	0.09177	0.0000
Depth	-0.00099	0.0000	-0.00869	0.0000	-0.02831	0.0000
Filter	-0.00000	0.7005	-0.00006	0.0000	-0.00023	0.0000
Arctan	-0.00437	0.0000	-0.02012	0.0000	-0.04583	0.0000
Sigmoid	-0.00144	0.0000	-0.00421	0.0000	0.00079	0.7785
Tanh	-0.00436	0.0000	-0.02016	0.0000	-0.04638	0.0000

Table A.2: The coefficient and p-values from linear regression of the lower bounds for neural networks trained on the SL set.

Rock paper scissor						
	L_∞		L_2		L_1	
	Coefficient	P-value	Coefficient	P-value	Coefficient	P-value
Bias	0.01042	0.0000	0.07017	0.0000	0.00185	0.9554
Depth	-0.00112	0.0000	-0.01291	0.0000	-0.04846	0.0000
Filter	0.00002	0.0069	-0.00023	0.0097	-0.00114	0.0426
Arctan	-0.00076	0.0282	0.02118	0.0000	0.11418	0.0000
Sigmoid	0.00357	0.0000	0.09613	0.0000	0.45056	0.0000
Tanh	-0.00068	0.0418	0.02077	0.0000	0.10071	0.0000

Table A.3: The coefficient and p-values from linear regression of the lower bounds for neural networks trained on the RPS set.

GTSRB						
	L_∞		L_2		L_1	
	Coefficient	P-value	Coefficient	P-value	Coefficient	P-value
Bias	0.01144	0.0000	0.08843	0.0000	0.18574	0.0000
Depth	-0.00122	0.0000	-0.01544	0.0000	-0.06464	0.0000
Filter	-0.00000	0.7822	-0.00004	0.0587	0.00016	0.1458
Arctan	-0.00207	0.0000	-0.00965	0.0000	-0.03502	0.0000
Sigmoid	-0.00015	0.1655	0.00417	0.0005	0.01486	0.0171
Tanh	-0.00234	0.0000	-0.01262	0.0000	-0.04361	0.0000

Table A.4: The coefficient and p-values from linear regression of the lower bounds for neural networks trained on the GTSRB set.

Mean of lower bounds			
Dataset	L_i	L_2	L_1
MNIST	0.0134	0.0900	0.2504
SLM	0.0051	0.0390	0.1250
Caltech Silhouettes	0.0105	0.0747	0.2301
RPS	0.0074	0.1628	0.8044
Cifar	0.0023	0.0276	0.1037
GTSRB	0.0060	0.0700	0.2697

Table A.5: The mean lower bounds for the neural networks on the different data sets, calculated on the L_∞, L_2, and L_1 norm.

Appendix B

C&W upper bounds

B.1 Regression

MNIST						
	L_∞		L_2		L_1	
	Coefficient	P-value	Coefficient	P-value	Coefficient	P-value
Bias	0.03124	0.0000	0.10021	0.1403	4.21098	0.0000
Depth	0.00600	0.0000	0.04292	0.0000	-0.29424	0.0000
Filter	0.00025	0.0000	0.00486	0.0000	0.01405	0.0000
Arctan	-0.04083	0.0000	-0.52718	0.0000	-1.34476	0.0000
Sigmoid	-0.01644	0.0000	-0.16648	0.0000	-0.40110	0.0031
Tanh	-0.04082	0.0000	-0.47321	0.0000	-1.38986	0.0000

Table B.1: The coefficient and p-values from linear regression of the C&W upper bounds for neural networks trained on the MNIST set.

Sign Language						
	L_∞		L_2		L_1	
	Coefficient	P-value	Coefficient	P-value	Coefficient	P-value
Bias	0.00468	0.0000	0.00960	0.0963	1.22133	0.0000
Depth	0.00210	0.0000	0.00645	0.0000	-0.00231	0.8786
Filter	0.00007	0.0000	0.00062	0.0000	0.00888	0.0000
Arctan	-0.01069	0.0000	-0.07930	0.0000	-1.07398	0.0000
Sigmoid	-0.00511	0.0000	-0.03973	0.0000	-0.39414	0.0000
Tanh	-0.01055	0.0000	-0.07860	0.0000	-1.07641	0.0000

Table B.2: The coefficient and p-values from linear regression of the C&W upper bounds for neural networks trained on the SL set.

Rock paper scissor						
	L_∞		L_2		L_1	
	Coefficient	P-value	Coefficient	P-value	Coefficient	P-value
Bias	0.00492	0.0002	0.08938	0.5319	19.90521	0.0000
Depth	0.00281	0.0000	0.08835	0.0002	-0.53545	0.1045
Filter	0.00010	0.0000	0.00766	0.0002	0.03510	0.3022
Arctan	-0.01610	0.0000	-0.89396	0.0000	-6.82215	0.0000
Sigmoid	-0.00219	0.0016	-0.09114	0.2164	-1.32588	0.2003
Tanh	-0.01467	0.0000	-0.83660	0.0000	-5.43824	0.0000

Table B.3: The coefficient and p-values from linear regression of the C&W upper bounds for neural networks trained on the RPS set.

GTSRB						
	L_∞		L_2		L_1	
	Coefficient	P-value	Coefficient	P-value	Coefficient	P-value
Bias	-0.00483	0.0000	-0.16585	0.0000	4.95038	0.0000
Depth	0.00386	0.0000	0.04685	0.0000	-0.32573	0.0000
Filter	0.00011	0.0000	0.00334	0.0000	0.03053	0.0000
Arctan	-0.00992	0.0000	-0.27345	0.0000	-3.09511	0.0000
Sigmoid	-0.00334	0.0000	-0.12256	0.0000	-1.08221	0.0000
Tanh	-0.01016	0.0000	-0.27623	0.0000	-3.16962	0.0000

Table B.4: The coefficient and p-values from linear regression of the C&W upper bounds for neural networks trained on the GTSRB set.

Mean of upper bounds			
Dataset	L_i	L_2	L_1
MNIST	0.0580	0.6270	5.8632
SLM	0.0157	0.0693	1.9621
Caltech Silhouettes	0.0539	0.7263	5.3560
RPS	0.0232	1.1427	26.0387
Cifar	0.0071	0.0656	3.1562
GTSRB	0.0234	0.3897	7.1474

Table B.5: The mean C&W upper bound for the neural networks on the different data sets, calculated on the L_∞, L_2, and L_1 norm.

Appendix C

HSJA upper bounds

C.1 Regression

MNIST				
	L_∞		L_2	
	Coefficient	P-value	Coefficient	P-value
Bias	0.03712	0.0000	0.47536	0.0000
Depth	0.00640	0.0000	-0.00050	0.9333
Filter	0.00029	0.0000	0.00368	0.0000
Arctan	-0.03830	0.0000	-0.53028	0.0000
Sigmoid	-0.01827	0.0000	-0.20074	0.0000
Tanh	-0.03998	0.0000	-0.56118	0.0000

Table C.1: The coefficient and p-values from linear regression of the HSJA upper bound for neural networks trained on the MNIST set.

Sign Language				
	L_∞		L_2	
	Coefficient	P-value	Coefficient	P-value
Bias	0.00376	0.0000	0.04878	0.0000
Depth	0.00250	0.0000	0.00360	0.0001
Filter	0.00011	0.0000	0.00063	0.0000
Arctan	-0.01109	0.0000	-0.08931	0.0000
Sigmoid	-0.00490	0.0000	-0.03359	0.0000
Tanh	-0.01142	0.0000	-0.09161	0.0000

Table C.2: The coefficient and p-values from linear regression of the HSJA upper bound for neural networks trained on the SL set.

Rock paper scissor				
	L_∞		L_2	
	Coefficient	P-value	Coefficient	P-value
Bias	0.02047	0.0175	0.46465	0.1830
Depth	0.00507	0.0003	0.04268	0.4522
Filter	-0.00037	0.0008	0.00777	0.1130
Arctan	-0.02763	0.0000	-1.29312	0.0000
Sigmoid	0.02964	0.0000	0.40161	0.0264
Tanh	-0.02855	0.0000	-1.23325	0.0000

Table C.3: The coefficient and p-values from linear regression of the HSJA upper bound for neural networks trained on the RPS set.

GTSRB				
	L_∞		L_2	
	Coefficient	P-value	Coefficient	P-value
Bias	-0.01832	0.0024	0.03721	0.4274
Depth	0.00649	0.0000	0.01999	0.0023
Filter	0.00032	0.0000	0.00459	0.0000
Arctan	-0.00230	0.5253	-0.19193	0.0000
Sigmoid	-0.00498	0.0721	-0.05110	0.0179
Tanh	-0.00448	0.1958	-0.24230	0.0000

Table C.4: The coefficient and p-values from linear regression of the HSJA upper bound for neural networks trained on the GTSRB set.

Mean of HSJA upper bounds		
Dataset	L_i	L_2
MNIST	0.0698	0.7813
SLM	0.0166	0.0860
CS	0.0568	0.6669
RPS	0.0797	2.4484
Cifar	0.0115	0.1091
GTSRB	0.0376	0.5112

Table C.5: The mean HSJA upper bound scores for the neural networks on the different data sets, calculated on the L_∞, L_2, and L_1 norm.

Appendix D

BIM success rate

D.1 Regression

	Dataset	Bias	Original accuracy	Depth	Filters	Arctan	Sigmoid	Tanh
Coef	MNIST	0.1756	-	0.0860	0.0012	-0.4525	-0.0901	-0.4501
	SLM	-0.0768	-	0.1131	0.0026	-0.4128	-0.2309	-0.4496
	CS	0.6274	-0.2419	-0.0142	0.0009	-0.1991	-0.2229	-0.1952
	RPS	-0.4615	-	0.1290	0.0044	-0.5862	-0.1607	-0.5302
	Cifar	0.5918	-0.4990	-0.0275	0.0005	-0.1414	0.2216	-0.1636
	GTSRB	-0.3562	-	0.0928	0.0026	-0.1970	0.0401	-0.2097
P-value	MNIST	0.0000	-	0.0000	0.0000	0.0000	0.0000	0.0000
	SLM	0.0136	-	0.0000	0.0000	0.0000	0.0000	0.0000
	CS	0.0000	0.0328	0.0010	0.0009	0.0000	0.0000	0.0000
	RPS	0.0002	-	0.0000	0.0000	0.0000	0.0032	0.0000
	Cifar	0.0000	0.0007	0.0001	0.2313	0.0000	0.0000	0.0000
	GTSRB	0.0000	-	0.0000	0.0000	0.0000	0.0161	0.0000

Table D.1: The coefficient and p-values from linear regression of the BIM success rate for neural networks trained on various datsets.

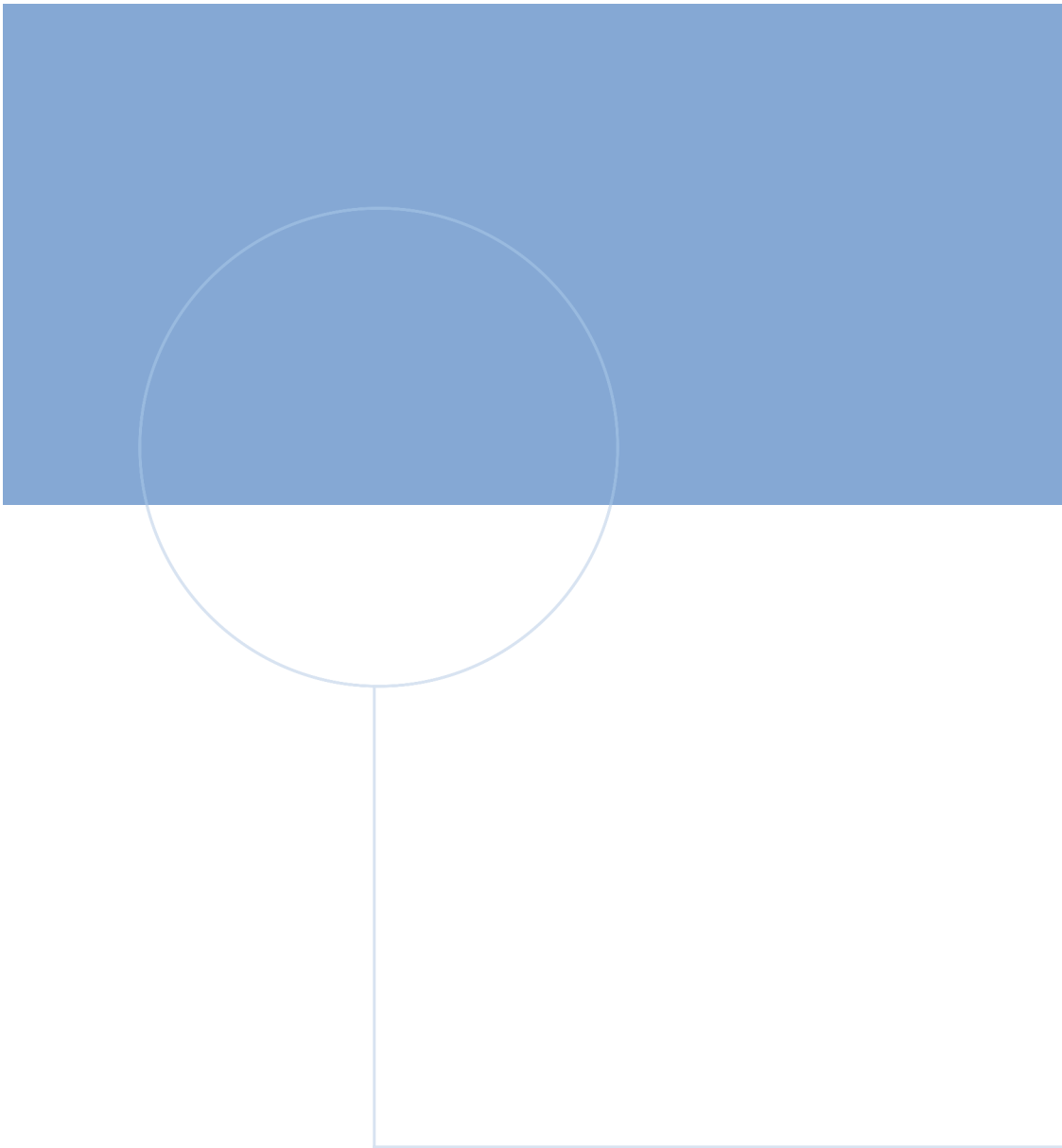
Appendix E

MIM success rate

E.1 Regression

	Dataset	Bias	Original accuracy	Depth	Filters	Arctan	Sigmoid	Tanh
Coef	MNIST	0.2172	-	0.0948	0.0009	-0.4407	-0.0958	-0.4378
	SLM	0.0981	-	0.0836	0.0003	-0.3259	-0.1215	-0.3110
	CS	0.6289	-0.2287	-0.0013	0.0008	-0.2062	-0.1959	-0.2042
	RPS	-0.2747	-	0.1463	0.0027	-0.7775	-0.2747	-0.6925
	Cifar	0.7371	-0.5117	-0.0328	0.0011	-0.1470	0.1414	-0.1650
	GTSRB	0.3428	-	0.0507	0.0002	-0.1963	-0.0139	-0.2100
P-value	MNIST	0.0000	-	0.0000	0.0008	0.0000	0.0000	0.0000
	SLM	0.0153	-	0.0000	0.3695	0.0000	0.0000	0.0000
	CS	0.0000	0.0153	0.7091	0.0003	0.0000	0.0000	0.0000
	RPS	0.0008	-	0.0000	0.0000	0.0000	0.0000	0.0000
	Cifar	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	GTSRB	0.0000	-	0.0000	0.5367	0.0000	0.4830	0.0000

Table E.1: The coefficient and p-values from linear regression of the MIM success rate for neural networks trained on various datsets.



NTNU

Norwegian University of
Science and Technology