Master Thesis
Computing Science


Radboud University


Info Support B.V.

# Evaluating Adversarial Attack Detectors using Formal Verification Methods

*Author:*
Reinier Joosse
R.Joosse@student.ru.nl
Student number: s4698649

*University supervisor,*
*First assessor:*
dr. N. Jansen
n.jansen@science.ru.nl

*Second assessor:*
dr. D. Strüber
d.strueber@cs.ru.nl

*External supervisors:*
E. Stoelinga
Emiel.Stoelinga@infosupport.com

L. van Neutegem
Leo.vanNeutegem@infosupport.com

*External process supervisor:*
H. Nieboer
Harry.Nieboer@infosupport.com

June 7, 2021

**Abstract**

Neural networks are known to be vulnerable to so-called *adversarial attacks*: inputs that are deliberately slightly modified to make the networks misclassify them. Several defenses against these attacks exist. It is hard, however, to know if these defenses are effective against unseen types of attacks. One can test whether an attack detector detects old attacks, but an attacker can always come up with attacks that the detector was not trained against. To prevent this historical bias when testing a detector, we propose two new metrics that estimate the quality of attack detectors independently of which attack method is used. The new metrics are shown to be computable using SMT solving techniques. The usefulness of the new metrics is demonstrated in experiments where their results are compared to the results of traditional test metrics.

**Acknowledgments**

I would like to thank my university supervisor Nils Jansen for the useful tips and feedback he provided during this thesis project. I would also like to thank Emiel Stoelinga, Leo van Neutegem and Harry Nieboer for their guidance. They supported me not only with the content of the thesis, but also with all other aspects that surrounded the project. I would like to thank the company Info Support for providing this opportunity. Finally, I would like to thank my fellow graduation students at Info Support and my other Info Support colleagues. The contact with them made the thesis period enjoyable despite the fact that most work was done from home due to the pandemic.

# Contents


**1 Introduction** **2**

**2 Preliminaries** **5**

**3 Related Work** **8**
3.1 Adversarial examples . . . . . 8
3.2 SafetyNet . . . . . 9
3.3 Metzen et al. . . . . 10

**4 Problem Statement** **11**
4.1 Detector quality metrics . . . . . 11
4.2 Motivation . . . . . 13
4.3 Research question . . . . . 14

**5 Method** **15**
5.1 Experimental setup . . . . . 15
5.2 Noisy MNIST . . . . . 19
5.3 Marabou . . . . . 19
5.4 SMT encoding of the queries . . . . . 21

**6 Results** **27**
6.1 General results . . . . . 27
6.2 Confusion matrix and SMT scores do not always agree . . . . . 28
6.3 False Negatives metrics negatively correlate . . . . . 28
6.4 False Positives metrics weakly positively correlate . . . . . 28

**7 Discussion of the Results** **32**
7.1 Added value of the new metrics . . . . . 32
7.2 Additional insights . . . . . 33
7.3 Limitations and future work . . . . . 34

**A Results Data** **39**

# Chapter 1

# Introduction

In the past decade, new machine learning techniques have made it possible for machines to process increasingly complex data. For example, machine learning is currently used for recognizing the content of images and for distinguishing spoken words in audio recordings. As the number of applications increases, it becomes increasingly important for these systems to be reliable. For example, if a traffic sign recognition system in a self-driving car does not recognize a red traffic light, the consequences could be severe.

In previous research, it was found that neural networks (an important machine learning method) are vulnerable to so-called *adversarial attacks* [24]. An adversarial attack is an input for a neural network that has been deliberately modified in such a way that the network fails to recognize it, while a human does not see any difference (or almost none). For example, in a picture of a stop sign, a few pixels are changed such that the network thinks it is a 30 km/h sign. Another example is shown in Figure 1.1, taken from Athalye et al. [2], who 3D-printed a turtle with an adversarial color pattern which was classified as a rifle almost all of the time by an ImageNet classifier.

To prevent such attacks, several solutions have been proposed in recent literature.

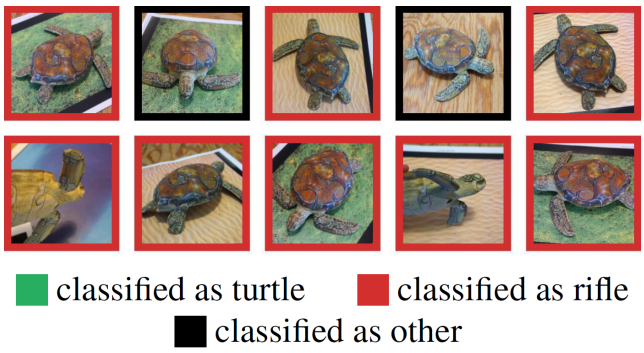



Figure 1.1: Pictures of an "adversarial" turtle classified as rifles. Taken from [2].

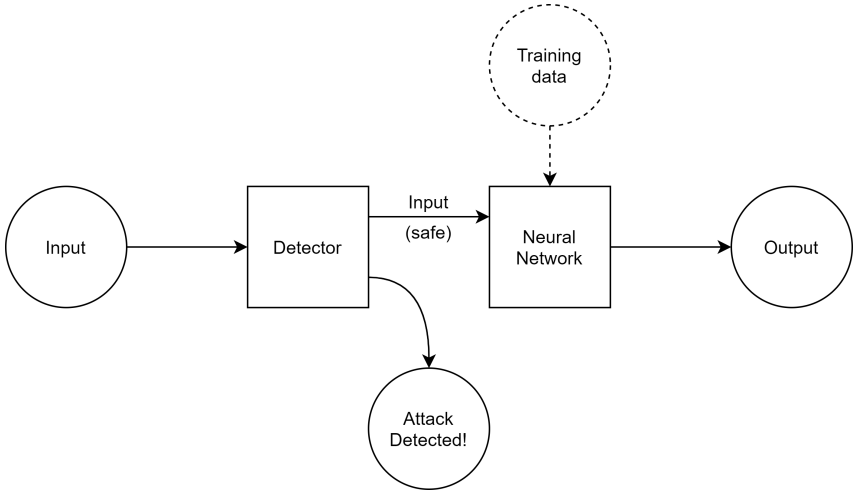



Figure 1.2: Normal operation of an adversarial attack detector. The detector analyzes the input for a neural network to check if it is an adversarial attack. If it does not detect an attack, the input is given to the network as usual. The detector may or may not use information about the structure or operation of the neural network.

These solutions usually fall into one of two categories:

1. Methods that *change the neural network itself* to make it more robust against adversarial attacks.

2. Methods that try to *detect* whether a given input is an adversarial attack. The input can then be ignored or rejected. Figure 1.2 shows the context in which an attack detector operates.

In this thesis, we only discuss techniques from the second category: adversarial attack detectors. Concretely, we try to *measure the effectiveness* of several known adversarial attack detectors.

A typical way to evaluate the effectiveness of an adversarial attack detector is to evaluate its performance on a test set. Such a test set contains both attacked inputs and normal inputs. If such a test set is available, it can be checked how many of the attacks are detected by the detector, and whether it raises any false alarms about normal examples. This approach depends on the quality of the test set, however. If the test set only contains attacks generated with one specific attack method, the test result may indicate that the detector is very effective. But when it is attacked using a different attack, it may suddenly perform much worse.

Instead of the traditional test set performance, we propose two new quality metrics for adversarial attack detectors based on Satisfiability Modulo Theories (SMT) solving. SMT solvers are programs that automatically formally prove properties [19]. For example, if an SMT solver concludes that there does not exist any undetected adversarial example close to an original image, then we can be sure that this conclu-

sion is correct.[1] That way, we can be sure that a detector detects all close adversarial examples regardless of which attack technique is used.

This thesis proposes two quality metrics of adversarial attack detectors can be computed using SMT solving; calculates those metrics, as well as traditional test set metrics, for several different adversarial attack detection techniques; and compares and analyzes the results. To our knowledge, we are the first to assess the quality of adversarial attack detectors using SMT solving techniques.

The contributions of this thesis include the following:

1. Two new quality metrics for adversarial attack detectors are proposed: the fraction of test samples that have close false positives, and the fraction of test samples that have close false negatives.

2. The new quality metrics are shown to be computable using SMT solving techniques.

3. Experiments show that these metrics provide useful information about the quality of adversarial attack detectors that is not obtained by using only the test set accuracy. For instance, one detector's test set accuracy was 0.9999, yet the SMT solver found undetected adversarial examples requiring no more than 0.2% change per pixel for 100% of tested normal inputs. More generally, as the test set accuracy improved for different detectors, the detectors' quality worsened according to one SMT metric. This result suggests that the detectors with high test set accuracies overfit on the particular adversarial attack used to generate the test set. Evidently, the new SMT metrics effectively estimate a detector's quality independent of which attack technique is used, in contrast to traditional test set metrics.

The remainder of this thesis is structured as follows. Before diving deeper into the contents of this thesis, some notations and definitions are given in Chapter 2. Subsequently, we will give an overview of related literature in Chapter 3. Chapter 4 presents the problem statement that will be answered using the method described in Chapter 5. The results can be found in Chapter 6. We conclude in Chapter 7.

[1] Except, of course, if the code of the SMT solver itself or the asked query contains bugs or logical errors.

# Chapter 2

# Preliminaries

This chapter lists some notations and definitions that will be necessary to understand the following chapters in this thesis.

Throughout this thesis, we will denote scalar variables with plain characters, such as $x$. Column vectors will be represented with bold face characters: $\mathbf{x}$. Matrices will be written with capital bold face characters: $\mathbf{X}$. The notation $\mathbf{X}[i]$ refers to the $i$-th row of matrix $\mathbf{X}$.

When a unary function that normally operates on scalars is applied to a vector or matrix, this means it is applied to every element in that vector or matrix, unless stated otherwise. For example, if $\phi(x) = \mathsf{max}(0, x)$, then

$$\phi\left(\begin{pmatrix}-5\\3\end{pmatrix}\right) = \begin{pmatrix}\phi(-5)\\\phi(3)\end{pmatrix} = \begin{pmatrix}0\\3\end{pmatrix}$$

**Definition 1** (Neural Network)**.** An (artificial) *neural network* is a function $f$ that maps an input vector $\mathbf{x}$ to an output vector $f(\mathbf{x})$. The input and output vectors are not necessarily of the same size. A neural network is composed of $n$ layers $f_i$, such that $f(\mathbf{x}) = f_n(f_{n-1}(\dots f_1(\mathbf{x})))$. Each layer works by multiplying a weight matrix $\mathbf{W}_i$ by its input $\mathbf{x}$, adding a bias vector $\mathbf{b}_i$ and passing it through an activation function $\phi_i$, namely: $f_i(\mathbf{x}) = \phi_i(\mathbf{W}_i\mathbf{x} + \mathbf{b}_i)$. The last layer of a neural network ($f_n$) is called the *output layer*. Note that $f_0$ does not exist as the input layer simply consists of the input values.

Let $D_j$ denote the number of dimensions in layer $j$. For example, $D_0$ is the number of pixels in the input image, $D_1$ is the number of nodes in the first hidden layer and $D_n$ is the number of outputs of the neural network.

Note that the arity of the layers may vary: for example, a neural network with three layers ($n = 3$) may have $f_1 : \mathbb{R}^4 \to \mathbb{R}^3$, $f_2 : \mathbb{R}^3 \to \mathbb{R}^4$ and $f_3 : \mathbb{R}^4 \to \mathbb{R}$. This makes $f : \mathbb{R}^4 \to \mathbb{R}$, meaning that the neural network takes a four-dimensional input and returns one value. Different layers may also have different activation functions.

Neural networks can be depicted as in Figure 2.1. In this figure, the neurons are depicted as white circles, and the layers as the vertical arrangements of the neurons.

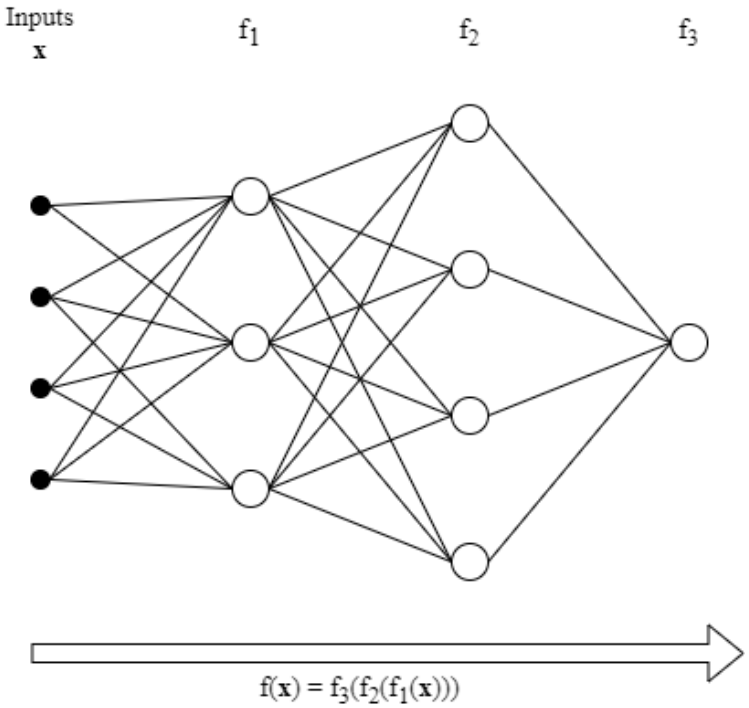



Figure 2.1: A schematic depiction of a neural network.

The activation value of each neuron depends on the activation values of all neurons in the previous layer, depicted by the lines which connect them. The strength of these connections is defined by the weight matrices.

The weights and biases of a neural network are learned in the training phase by training the network on a large amount of data of which the correct outputs are known. Neural networks are (usually) trained using the backpropagation algorithm [22]. Once training is complete, the weights and biases are fixed and the network only performs its learned computation.

**Definition 2** (Linear SVM)**.** A *linear support vector machine* consists of a decision function $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + \mathbf{b}$. To perform binary classification, we say that the SVM's output is 0 ("non-adversarial") if $f(\mathbf{x}) \leq 0$ and 1 ("adversarial") if $f(\mathbf{x}) > 0$.

Similarly to neural networks, the weights $\mathbf{w}$ and biases $\mathbf{b}$ are learned during the training phase of the SVM. After the training phase, they are fixed.

**Definition 3** (Satisfiability Modulo Theories)**.** For the purposes of this thesis, an *SMT formula* consists of two ingredients:

1. A set of variables that can be assigned real values;

2. A conjunction of linear equalities and inequalities over these variables.

For example, the following is an SMT formula:

$$x \leq y + 2 \text{ and } y \geq 5 \text{ and } y < 9$$

An SMT formula is *satisfiable* if and only if there exists an assignment to all its variables such that the constraint evaluates to "true". The formula above is satisfiable; for example, one possible assignment for the formula above is $[x \mapsto 8, y \mapsto 6]$.

For this instance, it is trivial to find a solution, but when the formulas become larger (in the order of thousands of variables or more), it becomes very hard to find a solution by hand. Computer programs that can solve SMT instances are called SMT solvers. An SMT solver takes an SMT formula as input and either proves that is unsatisfiable, or outputs a satisfying assignment. The SMT problem is NP-complete [19], but many instances can nevertheless be solved quickly by SMT solvers.

SMT solvers are very useful, since many problems can be formalized and translated into SMT formulas. One can then use an SMT solver to find a satisfying assignment (or conclude that it is unsatisfiable). A satisfying assignment can be translated back to a solution for the original problem. We use this property in this thesis.

# Chapter 3

# Related Work

The research done in this thesis project builds on the previous work of other researchers. In this chapter, we relate this thesis to relevant previous work. First, we describe some important works related to adversarial examples and verification of neural networks. Subsequently, we describe two adversarial attack detection techniques of which the basic forms were evaluated in this thesis.

## 3.1 Adversarial examples

Adversarial examples have been studied for a long time, also outside the field of neural networks. For example, one of the first applications was the evasion of e-mail spam filters [23]. Adversarial attacks gained attention from the research community when Szegedy et al. [24] discovered that neural networks are susceptible to them. Since this publication, many defenses have been proposed, many of which have subsequently been broken again by other researchers with increasingly powerful attacks [23, 26].

There are several hypotheses to explain why adversarial examples exist, but none is universally accepted yet [23]. Their existence demonstrates that trained neural networks do not use the same reasoning that humans use to make sense of data such as images. Researchers have come up with several different ways to find adversarial examples for a given neural network. A literature survey of attacks and defenses may be found in [23].

With the increased usage of neural networks in everyday life and their potential future applications, security concerns related to neural networks have gained increased interest. This has also partially inspired increased interest in the application of formal verification techniques on neural networks. If applied successfully, these methods would guarantee certain properties of the networks, such as the safety of their decisions or robustness against adversarial attacks.

For example, Huang et al. [12] found a way to perform a layer-by-layer analysis of neural networks with an SMT solver to see if an adversarial example exists within a given distance of a normal input. Also, Katz et al. [13] created an SMT-based

tool called Reluplex which can verify properties of neural networks, including robustness against adversarial examples. Reluplex was later extended and integrated in a tool called Marabou [14]. There are also works that focus on evaluating the robustness of neural networks using Mixed Integer Linear Programming (MILP, a formal verification method like SMT), e.g. [11, 25].

SMT-based tools have already been used to estimate the robustness of neural networks against adversarial examples. Carlini et al. [4] evaluated the adversarial robustness of a neural network after making it more robust using a particular defense technique. They did this by using Reluplex to find the minimally distorted adversarial attack. The intuition is that if the defense is effective, then the attack will need to make more modifications to an original input (e.g. change more pixels) than when the defense is not applied. Hence, better defenses will increase the minimal distortion of an adversarial example. In short, Carlini et al. [4] focused on evaluating a defense that made a neural network more robust. In contrast, this thesis evaluates attack *detection* techniques.

## 3.2 SafetyNet

One method for detecting adversarial attacks on neural networks is SafetyNet, proposed in [17]. The method is claimed to effectively detect both normal adversarial attacks and attacks that actively try to outsmart the detector.

The SafetyNet method is based on the assumption that adversarial examples cause unusual activation patterns of neurons in layers near the end of the network's calculation. SafetyNet performs the usual calculation of the network up to a layer $f_i$. Subsequently, it passes the activation values of that layer to a mechanism that decides whether these activation values are likely to be caused by an adversarial example. Figure 3.1 gives an overview of how SafetyNet works.

To be precise, before the activation values are used as input for the attack detector, they are first converted to (binary) codes. An activation value is converted to a 1 if it passes a threshold; otherwise it becomes a 0. For example, if the threshold value is 0.43, then the activation values $(0.2, 0, 0.6, 3.21, 0.3)$ are converted to the code $(0, 0, 1, 1, 0)$. It is also possible to use $n > 1$ thresholds, which results in an $(n + 1)$-ary code.

The attack detection mechanism is a Support Vector Machine (SVM) with a Radial Basis Function (RBF) kernel. This is simply a classification method, comparable to a neural network. The RBF-SVM is trained on a data set of labeled inputs, which contains codes at layer $f_i$ of both normal and adversarial inputs (labeled with the true class, namely whether the example is adversarial). The adversarial examples in this training data set can be generated using any existing attack method.

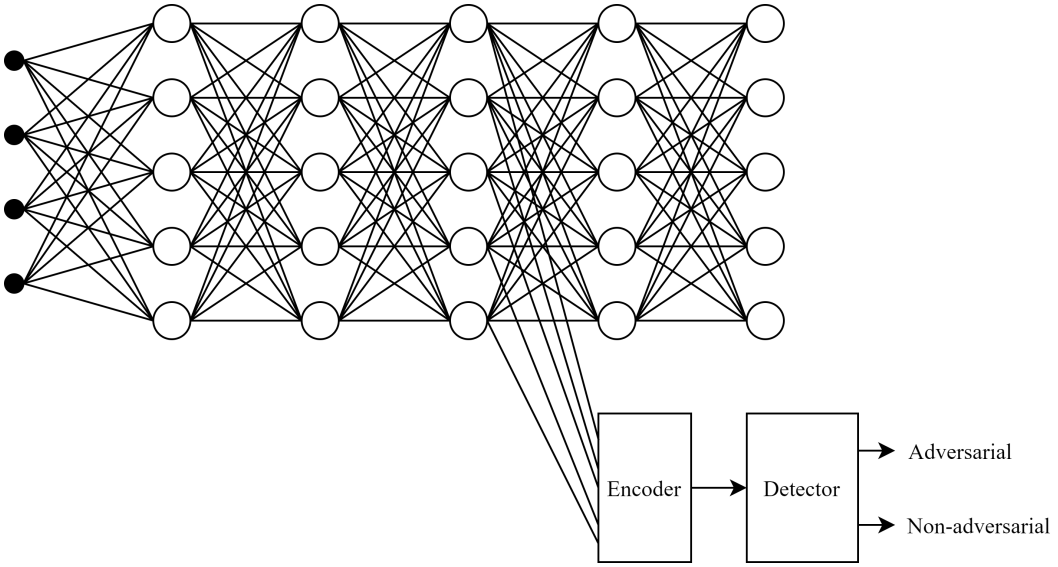



Figure 3.1: SafetyNet.

## 3.3 Metzen et al.

Metzen et al. [18] proposed a similar approach to SafetyNet to detect adversarial examples. Again, the values of the nodes at a specific layer in the network are investigated to see if they were likely caused by an adversarial example. This time, however, these values are not converted to codes; and they are not classified by an SVM, but by a feed-forward neural network. The size and architecture of this detection network can vary based on the original network and based on which layer of the original network is used as input for the detection network.

The paper by Metzen et al. [18] was published before the SafetyNet paper [17]. The authors of the SafetyNet paper found out (experimentally) that their detection approach was more difficult to defeat using standard (gradient-based) attack methods; and that it generalized better to defend against attack methods that it had not been trained against.

# Chapter 4

# Problem Statement

We propose and evaluate two new quality metrics for adversarial attack detectors. In this chapter, we will introduce those metrics and pose the central research question.

## 4.1 Detector quality metrics

Some metrics that indicate the quality of an adversarial attack detector can be calculated without an SMT solver. In particular, as with any binary classifier, we can look at its *confusion matrix* (see Table 4.1). To understand the confusion matrix, note that a binary classifier has two possible outputs, which we will call positive and negative. In our case, a positive outcome will mean that the detector concludes that its input is adversarial; a negative outcome will mean that it concludes that its input is normal.[1] The confusion matrix contains four numbers:

- True positives: the fraction of adversarial examples that are classified as adversarial.
- True negatives: the fraction of normal examples that are classified as normal.
- False positives: the fraction of normal examples that are classified as adversarial.
- False negatives: the fraction of adversarial examples that are classified as normal.

To calculate these fractions, we can gather many normal examples and generate adversarial examples using a known attack method, and see how the detector classifies them. Naturally, for a good detector, the number of true positives and true negatives will be high, and the number of false positives and false negatives will be low. Depending on the application, some of these numbers may be more important

[1] Here, we use the convention from medical tests, where a positive test outcome implies that the patient has the disease.

| | **Actually adversarial** | **Actually normal** |
|---|---|---|
| **Classified as adversarial** | True positives | False positives |
| **Classified as normal** | False negatives | True negatives |

Table 4.1: The confusion matrix for an adversarial attack detector.

| | |
|---|---|
| Accuracy | $\frac{\text{true positives+true negatives}}{\text{true positives+false positives+false negatives+true negatives}}$ |
| False positive rate | $\frac{\text{false positives}}{\text{true negatives+false positives}}$ |
| False negative rate | $\frac{\text{false negatives}}{\text{true positives+false negatives}}$ |

Table 4.2: Different ways to summarize a confusion matrix.

than others. For example, in security-critical applications, it may be acceptable if some normal examples are flagged as adversarial (false positives), but unacceptable if some attacks remain undetected (false negatives).

The values from the confusion matrix can also be combined into one number. For example, the *accuracy* is the number of correctly classified inputs (true positives and true negatives) divided by the total number of inputs. The accuracy is a good choice if the set of test inputs is class-balanced, i.e. there are as many positive as negative examples. If we are mainly interested in the number of false positives, we may also use the *false positive rate*; and for false negatives, there is the *false negative rate*. Table 4.2 lists the definitions of these metrics.

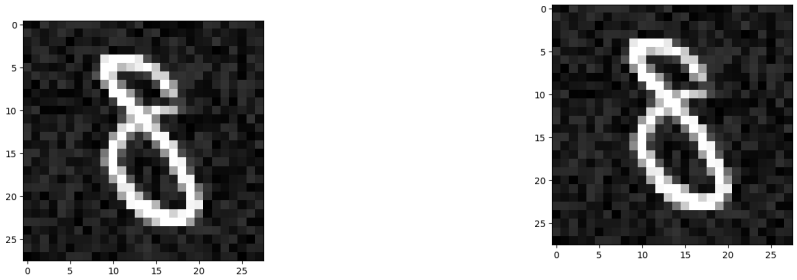
On top of the confusion matrix metrics, we propose two quality metrics that can be calculated using SMT solvers. For these metrics, we assume that we have a test set of normal inputs. The metrics are as follows:

1. The fraction of the test inputs that are close to a false negative. That is: the input is close to another input that is not classified correctly (it is adversarial), but the detector thinks it is non-adversarial.

2. The fraction of the test inputs that are close to a false positive. That is: the input is close to another input that is classified correctly (it is non-adversarial), but the detector thinks that it is adversarial.

Of course, the exact meaning of the word "close" in these metrics must still be defined. Intuitively, an input is close to another input if it is hard to spot the difference between the two. This can be formalized using several different distance metrics, such as the Manhattan distance ($L_1$) or the Euclidean distance ($L_2$).

In this thesis, we use grayscale images as inputs, with pixels values ranging between 0 and 1. To measure the distance between two images, we use the Chebyshev distance ($L_\infty$), and we will say that two images are close if the distance between them is smaller than or equal to 0.002. This means that image A is close to image B if one can obtain image B from image A by adding to or subtracting from each



(a) The original image.

(b) An image 0.002 distance away from the original.

Figure 4.1: An example of a Chebyshev distance of 0.002.

pixel (which ranges from 0 to 1) no more than 0.002. An example of this is shown in Figure 4.1: the difference is barely noticeable.

Our choice for the Chebyshev distance is straightforward to encode in SMT and suffices for the purposes of this thesis. In practice, however, it may be worth investigating other notions of distance. Our choice for the bound of 0.002 is based on the practical observations. First of all, larger bounds quickly lead to longer run times for the SMT solver. Secondly, we observed that there already exists a false negative or false positive within this bound for many of the original inputs; so a bound of 0.002 suffices.

## 4.2 Motivation

One way to think about how the new metrics work, is that they measure the sparsity of the false positives and false negatives. If there are few false positives (resp. false negatives), then few normal inputs will be close to a false positive (resp. false negative). A good detector will have few false positives and false negatives; therefore, calculating the two metrics for that detector should result in low numbers.

The two metrics described above are closely related to the amounts of false negatives and false positives in the confusion matrix, but there is an important difference. Namely, we hypothesize that the objectivity of the values in the confusion matrix suffers from a problem that the new metrics do not suffer from, as we will see now.

As mentioned before, to calculate the values in the confusion matrix, we evaluate a detector on an existing test set with both normal examples and adversarial examples. The problem with this approach is that it depends on the way the adversarial examples in the test set are generated. Researchers usually generate the adversarial examples using one attack technique. However, a detector may be very good at detecting adversarial examples generated using one attack technique, but worse when a different attack is used. Hence, a detector that performs very well on

a test set generated with the best currently known attack methods may fail when a new attack technique is invented.[2]

This problem was already observed by Carlini et al. [4], who tried to evaluate the adversarial robustness of a neural network. Instead of evaluating the network on a test set, they proposed to only use a set of non-adversarial examples, and for each example, compute the distance to the closest adversarial example. This involves repeatedly invoking an SMT solver to find the closest adversarial example for each test input. The rationale behind this approach is that increased adversarial robustness will increase the average distance to the closest adversarial example.

Like the metric proposed by Carlini et al, the metrics in this thesis depend on SMT solving and not on a test set with generated adversarial examples. However, unlike their metric, computing the new metrics requires only one question (or a fixed number of questions) to the SMT solver for each test input.[3] Additionally, they are specifically tailored to determining the quality of adversarial attack detectors rather than the adversarial robustness of a neural network.

In summary, the motivation behind the new metrics is that they estimate the quality of attack detectors independent of which attack technique is used. This also allows us to compare the values for the new metrics to the confusion matrix values. If the confusion matrix values are consistently better than the SMT metrics, then that may indicate that the detector is indeed good at detecting adversarial attacks generated with a specific attack technique, but not very good at detecting attacks in general.

## 4.3 Research question

For this thesis, implementations were made and experiments were done to answer the following central research question:

> Does calculating the two proposed metrics (described in Section 4.1) result in information about the quality of detectors that contradicts their confusion matrices?

If this turns out to be the case (e.g. the confusion matrix suggests that a detector is very good, but the SMT metrics suggest the opposite), then we can conclude that the new metrics provide valuable information that cannot be deduced from only the confusion matrix. If, on the other hand, the proposed metrics do not lead to different conclusions than the confusion matrix, then the new metrics are of less added value.

[2] This has indeed happened before; see e.g. [5].

[3] To calculate the average distance to the closest adversarial example, one has to iteratively invoke the SMT solver and perform binary search. In contrast, the new metrics search for a specific input bounded around each original input, which does not require iteratively invoking the SMT solver.

# Chapter 5

# Method

To answer the research question, we calculate the proposed metrics and the confusion matrices for several setups of classification networks and detectors. This chapter describes the full experimental setup that was used, followed by some important comments about the data set that was used for these experiments. The chapter closes with a description of how our proposed metrics were encoded and calculated using the Marabou tool.

## 5.1 Experimental setup

A schematic representation of the verification setup for calculating the SMT metrics can be seen in Figure 5.1. Some properties of neural networks can already be verified using existing tools (see Chapter 3). However, existing approaches do not encode attack detectors.

The adversarial attack detectors we used detect attacks on a neural network that recognizes handwritten digits. We trained this network to classify the images of handwritten digits from the MNIST data set [16]. In other words, the network was trained to recognize which digit from 0 to 9 is shown in an image.[1] To keep the setup simple, we chose not to use a deep convolutional neural network (commonly used in image recognition tasks) but a small fully connected feed-forward network instead. A classification network with one hidden layer, 32 hidden nodes and the ReLU activation function was used (shown in Figure 5.2). It has one input for each of the pixels in the input images, and ten output nodes. Each output node represents one of the classes. In the output layer, no activation function is used. This network was trained for 20 epochs. When classifying an image, the class of the output node with the highest value is the network's predicted class.[2] Before training, the

[1] To train all neural networks in our experiments, the Adam optimizer with a learning rate of 0.001 was used, using the cross-entropy loss function. When training SVMs, the squared hinge loss was used.

[2] If there are multiple output nodes with the same highest value, an arbitrary decision is made between them.

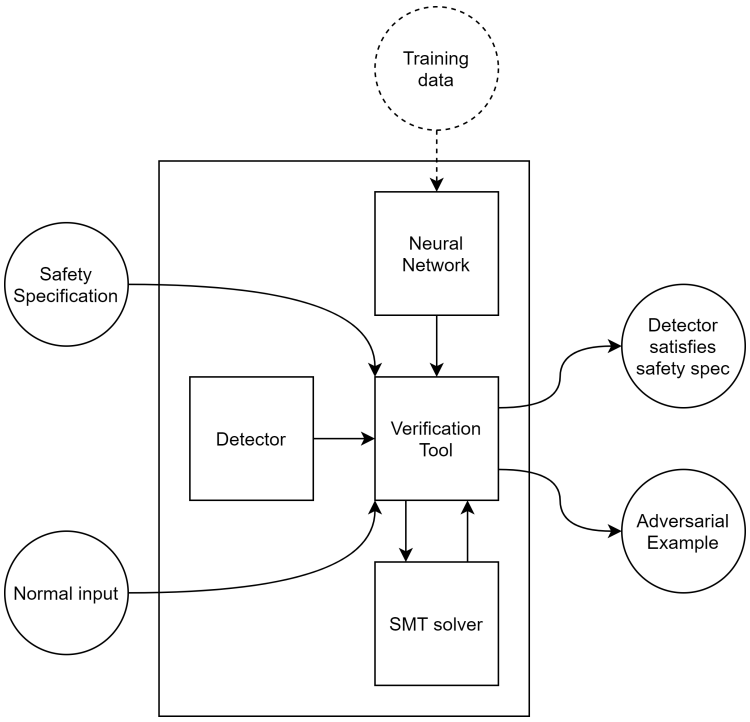



Figure 5.1: Verification setup for an attack detector. A verification tool creates a formal encoding of the neural network and the attack detector, in an SMT input language. The user gives a safety specification for a normal input of the neural network, such as: "Does there exist a perturbation of this input within a distance of 0.002 that will both fool the neural network AND pass the detector?" The verification tool passes the encodings and the question to an SMT solver. This solver solves the problem; the verification tool translates its output back to the original problem and outputs the solution.

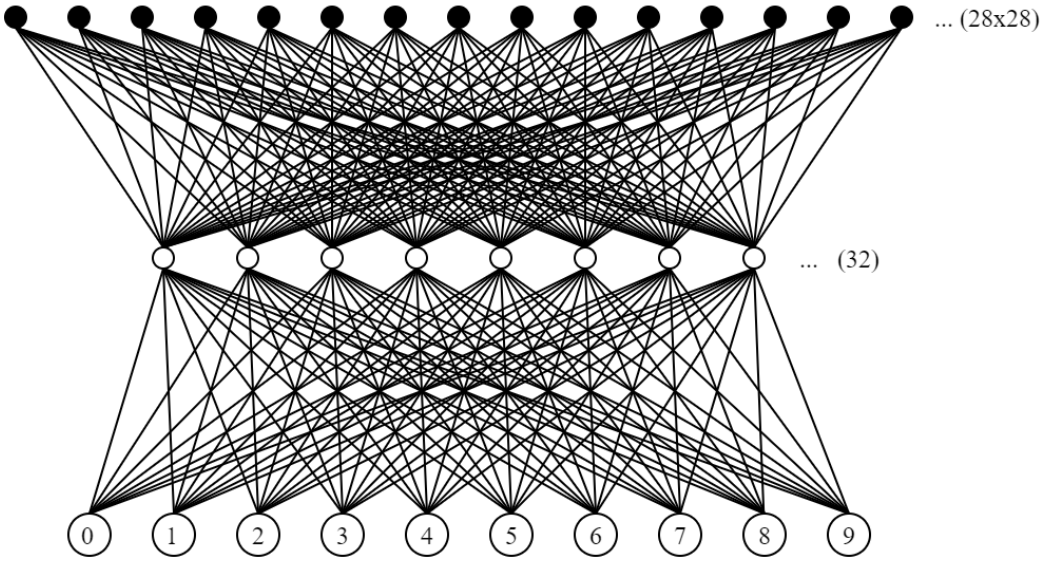



Figure 5.2: The MNIST classifier. In this picture, computation happens from top (input) to bottom (output).

biases are set to 0 and the other weights are initialized randomly using the Glorot uniform initializer [9]. This network architecture gives us worse than state-of-the-art techniques on the MNIST data set [3], but our setup is a good trade-off between accuracy and network and training complexity.

We trained several adversarial attack detectors to detect whether an input to the classification network is adversarial. The following detectors were trained:

- A linear SVM. This detector may be seen as a simplification of the SafetyNet approach [17]. In contrast to the SafetyNet paper, this detector does not use thresholds and instead operates directly on the values of the ReLU nodes in the classification network. In addition, it uses a linear basis function instead of an RBF kernel.

- A fully connected feed-forward neural network with one hidden layer, 10 nodes per hidden layer, the ReLU activation function on the hidden nodes and no activation function on the output nodes. We refer to this as the "1x10" neural network detector.

- A fully connected feed-forward neural network with two hidden layers, 32 nodes per hidden layer, the ReLU activation function on the hidden nodes and no activation function on the output nodes. We refer to this as the "2x32" neural network detector.

The neural network detectors are similar to the ones used by Metzen et al. [18], although Metzen et al. also propose a special training procedure for extra adversarial

robustness which was not used in this case. The neural network detectors were all trained for 10 epochs.[3] They are similar to the classification network shown in Figure 5.2, except the number of hidden nodes are different, and the second detector network has two hidden layers instead of one. Additionally, the output layer of these detectors only contain two nodes: one that indicates "adversarial" and one that indicates "not adversarial". Again, in a computation, the output node with the highest value determines the network's classification decision.

These detectors were chosen because they are fairly small and simple while still achieving reasonable accuracies, as shown in Chapter 6. Their small size and simplicity also makes it faster and easier to calculate our metrics.

Furthermore, we created three instances of each of these detectors and connected each instance to a specific layer of the MNIST classification network. The first instance classified the input images directly; hence it decided whether an input image is adversarial without taking the MNIST classification network into account. The second instance takes the output of the hidden layer of the MNIST classification network as its input. The third instance decides adversariality based on the output values (logits) of the MNIST classification network.

In total, this leaves us with $3 \cdot 3 = 9$ different detector setups. The detectors were trained on a training data set with normal and adversarial examples. To obtain this training data set, we started with the normal MNIST training set and generated attacks on the images using the Fast Gradient Sign Method (FGSM) [10] with parameter $\epsilon = 0.1$. We kept only the subset of original examples for which the original classification was correct, but for which FGSM successfully found an adversarial example. This set of normal examples was combined with the found adversarial examples to form the training set. Hence, the training set is perfectly class-balanced, as there is one adversarial example for each normal example.

For each of these setups, we determined the values of its confusion matrix, as well as its values for our two proposed metrics. To determine the values of the confusion matrix, we generated a test set with both normal and adversarial examples from the MNIST test set, analogously to how the training set was generated. To calculate the new metrics, 200 images were randomly selected from the MNIST test set. The same 200 images were used to calculate the metrics for all detector setups.

We used the following libraries to implement our experiments in the Python programming language:

- TensorFlow [1], for creating, training and testing the neural networks for both the MNIST classifier and the detectors.
- Scikit-learn [21], for creating, training and testing the linear SVM detector.
- CleverHans [20], for generating adversarial examples with the FGSM method.

[3] All detectors were trained for the same amount of epochs to make experimenting convenient. The goal of this thesis was not to train the best possible detectors, but to compare the quality of detectors using different metrics.

(a) The original image.

(b) The same image, with added random noise of at most 0.2.

Figure 5.3: Adding random noise to MNIST images.

- Marabou [14], for calculating our proposed metrics using SMT solving.

## 5.2 Noisy MNIST

In our experiments, the MNIST data set [16] was used: a set of images of handwritten digits. Each sample is a square of 28 by 28 grayscale pixels. We represent each pixel value as a float in the range $[0, 1]$ where 0 is black and 1 is white. The MNIST data set is divided in a set of 60 000 training images and 10 000 test images. We did not use a validation set. It is often used to evaluate machine learning techniques, but when evaluating techniques related to adversarial attacks on it, we need to take a precaution.

A potential problem with this data set was pointed out by Carlini & Wagner [5]: any pixel that is not part of the digit in a sample, has the exact value 0. Digit recognition networks can overfit on this property, meaning that if only a tiny amount was added to these black pixels, the network may be confused and produce a misclassification. Adversarial attacks can easily make use of this property.

Since this issue is specific to the MNIST data set, evaluating detectors on the original MNIST data set might result in results that do not generalize. For this reason, we modified the images in the MNIST data set in the following way: a random noise value from the range $[0, 0.2]$ (with uniform probability) is added to each pixel. An example of this can be seen in Figure 5.3. This modification prevents the detector from detecting adversarial examples solely by observing that the non-digit pixels are no longer pitch black. This modified MNIST data set was used in all experiments: for training the classifier as well as the detectors.

## 5.3 Marabou

We now turn to a description of how our new metrics may be calculated using SMT.

Most automated SMT solvers use the *Simplex* algorithm [7] at their core. The way the Simplex algorithm works will not be discussed here; see e.g. [19, 6] for a detailed description. The Simplex algorithm requires that all of the constraints in the input query are linear (in)equalities. This becomes a problem when we want to verify properties of neural networks, because neural networks typically do not represent linear calculations. This is caused by their use of non-linear activation functions such as the ReLU function. An image of the ReLU function is shown in Figure 5.4. It is clearly not linear.

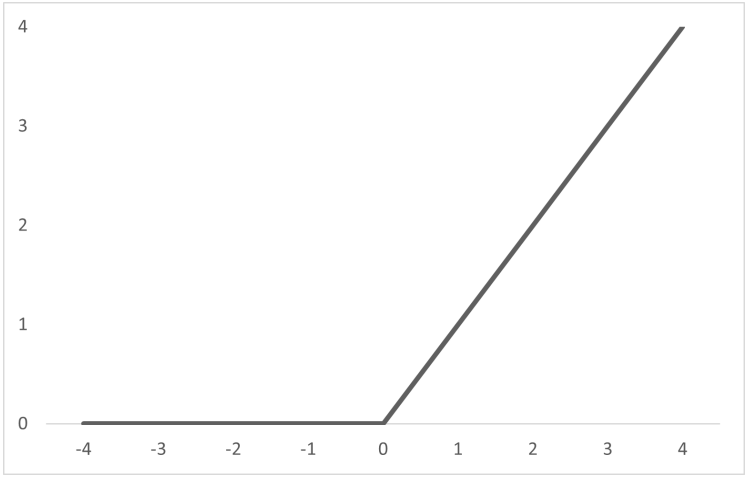



Figure 5.4: The ReLU activation function.

The open source Marabou tool [14] is based on the Reluplex [13] algorithm. Reluplex extends the Simplex algorithm to make it possible to handle ReLU constraints. It essentially does this by using the fact that although the ReLU function is not linear, it is *piecewise linear*. That is, it is both linear on the interval $(-\infty, 0]$, defined by the function $\phi(x) = 0$; and on the interval $[0, +\infty)$, defined by the function $\phi(x) = x$. In principle, this observation makes it possible to do a case split on each ReLU node, resulting in a system of only linear equations. This leads to an unfeasible number of cases, however.[4]

Instead of case splitting on every node, Reluplex explicitly keeps track of upper and lower bounds for all variables while performing Simplex. When a lower or upper bound of 0 is discovered on a ReLU node, a case split on that node is no longer necessary. When an upper or lower bound of 0 cannot be derived easily, the case split can still be done. In practice, it turns out that this optimization makes the verification process significantly faster than when case splitting on every ReLU node [13].

[4]Worst case, $2^n$ where $n$ is the number of nodes.

## 5.4 SMT encoding of the queries

To calculate our proposed quality metrics for detectors using Marabou, we repeatedly need it to find false positives or false negatives within a certain bound of an existing original input. We will now see how these queries can be encoded in SMT. Figure 5.5 shows a concrete example of how to construct an SMT query following the method described in this section.

The question posed by these queries is essentially the following: does there exist an input which satisfies some constraints? Namely: 1) it is close enough to my original input; and 2) it is a false positive (or a false negative) for a specific classifier and detector. Such an input is essentially a collection of real numbers (e.g. pixel values). If we represent each input dimension (pixel) with one variable, then an input is an assignment to all input variables. Hence, the SMT solver needs to either find an assignment to the input variables that satisfies these constraints, or conclude that such an assignment does not exist (within reasonable time).

In summary, the SMT encoding will have the following ingredients:

1. A variable for each dimension of the desired answer to the query.
2. A constraint that specifies that the answer to the query must be close enough to the original input.
3. A constraint that specifies that the answer to the query must be a false positive (or false negative) for the classifier and detector under consideration.

The first ingredient is straightforward: we simply create one variable $x_i$ for each input dimension (pixel).

For the second ingredient, we need to limit the possible values of these variables to a range around the original input. Since we chose the bound to be a Chebyshev distance of at most 0.002, we can straightforwardly encode this using a constraint on each individual variable. For each variable, we specify that the difference between its assignment and the value of the corresponding input dimension in the original input cannot be greater than 0.002. Formally, for the variable $x_i$, suppose that $x'_i$ is the value of the corresponding dimension in the original input. Then, we require that $x'_i - 0.002 \leq x_i \leq x'_i + 0.002$.

The third ingredient specifies either that the answer is a false positive or a false negative. Whether an input is a false positive or a false negative depends on the outcome when it is used as an input for the classifier and the detector. Namely, it is only a false positive if the detector classifies it as adversarial, but the main classifier classifies it correctly; and it is only a false negative if the detector classifies it as non-adversarial, but the main classifier classifies it incorrectly. Therefore, we need to encode the computations of the main classifier and the detector in SMT and add constraints on their outcomes.

The main classifier is simply a neural network. Therefore, to encode the computation of the classifier in SMT, we need to encode a neural network. The Marabou

The Reluplex encoding of a neural network works as follows [13]. For each hidden layer $0 < j < n$, we introduce two variables for each node $0 \leq i < D_j$, namely $a_i^j$ and $r_i^j$. The first variable, $a_i^j$, is constrained to the activation value of the node. The second variable, $r_i^j$, is constrained to the ReLU'ed activation value of the node. Concretely, $a_i^j$ constrained to:

$$a_i^j = \mathbf{W}_j[i] \begin{pmatrix} r_0^{j-1} \\ \vdots \\ r_{D_{j-1}-1}^{j-1} \end{pmatrix} + \mathbf{b}_j$$

where $\mathbf{W}_j$ is the weight matrix of layer $j$; and $r_i^j$ is constrained to:

$$r_i^j = \mathsf{ReLU}(a_i^j)$$

For each node $0 \leq i < D_0$ in the input layer, we only have one variable $r_i^0$, which will ultimately take a (pixel) value of the network's input. For each node $0 \leq i < D_n$ in the output layer, we only have one variable $a_i^n$, constrained to the activation value of that node (as for the hidden layers). Hence, the variables $a_i^0$ and $r_i^n$ do not exist. By construction, the introduced constraints ensure that the variables can only take the values that represent the neural network's computation. In particular, if the input variables $r_i^0$ are assigned the values of a particular input for the neural network, then the constraints will ensure that the output variables $a_i^n$ take the values that would result from computing the network's output values on this input in the usual way. This is how the Reluplex algorithm encodes a neural network in its variant of SMT.

Box 5.1: How Reluplex encodes neural networks in SMT.

tool already has built-in support for importing neural networks, as proposed in the Reluplex paper [13]. Box 5.1 describes how neural networks are encoded in SMT.

To specify that the result must be a false positive or a false negative, we encode both the main classification network and the detector network. In case the detector is a linear SVM, we can add the necessary constraints without requiring extra variables, as we shall see shortly. We first consider the case when the detector is a neural network. For increased readability, we will assume in the following that the main classification network is a MNIST classifier, but the approach is not limited to the MNIST case.

We want Marabou to find an image similar to an original input that is classified either correctly or wrongly by the MNIST classification network. We already introduced variables $x_i$ to hold the pixel values. To input these to the classification network, we simply add one constraint for each pixel. Namely, $x_i = r_i^0$ for all $0 \leq i < D_0$ where $r_i^0$ are the input variables of the Marabou encoding of the

classification network.[5]

The MNIST classification network has ten output nodes, and hence ten corresponding output variables $a_i^n$ in the encoding. The output node with the highest value represents the network's final output value. Therefore, if we want to specify that the input is classified correctly (in the false positive case), we add nine constraints $a_j^n < a_i^n$ for all $j \neq i$ where $i$ is the correct class.

Conversely, if we want to specify that the input is classified incorrectly (in the false negative case), there are nine possible cases: namely, in each case, the input is classified as one of the nine other classes. Therefore, we duplicate the entire encoding nine times. In each of these copies, we specify with nine constraints that it is classified as one of the other classes, as discussed above. If Marabou can find an example that satisfies all constraints for any of the nine copies, then we have found an input that is classified incorrectly. If all nine queries are unsatisfiable, then we conclude that such an example does not exist.

We have now specified that any example found by Marabou must be either classified correctly or incorrectly by the MNIST classifier. Next, we must specify that the detector must either flag the example as adversarial or non-adversarial (depending on whether it must be a false positive or a false negative). We distinguish two cases: one where the detector is a neural network, and one where the detector is a linear SVM.

In case the detector is a neural network, we can encode the requirement similarly to how we encoded the constraint on the output of the MNIST classifier. First, we encode the network using the method described in Box 5.1. (From now on, we will write ${}^c a$ and ${}^c r$ if we mean the variables from the MNIST classifier, and ${}^d a$ and ${}^d r$ if we mean the variables from the detector network.) Next, we specify which variables are used as input to the detector network. Depending on the experiment, the detector can be connected to the MNIST classifier's input, the classifier's hidden layer values or the classifier's output layer values. To specify this, we add one constraint for each input variable. If the detector is connected to the classifier's input, we add the constraints ${}^d r_i^0 = {}^c r_i^0$ for all $0 \leq i < D_0$. If it is connected to the classifier's first hidden layer, we add the constraints ${}^d r_i^0 = {}^c r_i^1$. If it is connected to the classifier's output layer, we add the constraints ${}^d r_i^0 = {}^c a_i^n$.[6] A detector network has two outputs: one representing "adversarial" and one representing "non-adversarial". Again, the output node with the highest value determines the class. Therefore, we can constrain the detector to label the input as adversarial or non-adversarial by adding the constraint ${}^d a_0^n < {}^d a_1^n$ or ${}^d a_1^n < {}^d a_0^n$, depending on whether we are looking for false positives or false negatives.[7] This way, any example found

[5] Equivalently, we could remove the variables $x_i$ and turn all constraints on $x_i$ into constraints on the corresponding $r_i^0$. This is what we did in our test implementation.

[6] Again, we can equivalently remove the detector's input layer variables and turn all constraints on them into constraints on the corresponding variables of the MNIST classifier, which we did in the implementation.

[7] We deliberately avoid the case when ${}^d a_1^n = {}^d a_0^n$, as that means the classification is uncertain and the detector may choose to either classify the input as adversarial or non-adversarial. If ${}^d a_1^n$

| | False Positive | False Negative |
|---|---|---|
| Input Layer | $f\left(\begin{pmatrix} {}^{c}r_0^0 \\ \vdots \\ {}^{c}r_{D_0-1}^0 \end{pmatrix}\right) > 0$ | $f\left(\begin{pmatrix} {}^{c}r_0^0 \\ \vdots \\ {}^{c}r_{D_0-1}^0 \end{pmatrix}\right) \leq 0$ |
| First Hidden Layer | $f\left(\begin{pmatrix} {}^{c}r_0^1 \\ \vdots \\ {}^{c}r_{D_1-1}^1 \end{pmatrix}\right) > 0$ | $f\left(\begin{pmatrix} {}^{c}r_0^1 \\ \vdots \\ {}^{c}r_{D_1-1}^1 \end{pmatrix}\right) \leq 0$ |
| Output Layer | $f\left(\begin{pmatrix} {}^{c}r_0^n \\ \vdots \\ {}^{c}r_{D_n-1}^n \end{pmatrix}\right) > 0$ | $f\left(\begin{pmatrix} {}^{c}r_0^n \\ \vdots \\ {}^{c}r_{D_n-1}^n \end{pmatrix}\right) \leq 0$ |

Table 5.1: Encoding the linear SVM constraint depending on which layer is input to the detector, and whether to look for a false positive or a false negative.

by Marabou will be either flagged as adversarial or non-adversarial (according to our wish) by detector neural networks.

If the detector is not a neural network, but a linear SVM, we do not need to encode a detector network using the procedure in Box 5.1. Instead, one constraint suffices. As described in Chapter 2, the SVM is simply a function $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + \mathbf{b}$, where $\mathbf{w}$ and $\mathbf{b}$ are fixed constants. An input $\mathbf{x}$ is labeled as adversarial if $f(\mathbf{x}) > 0$, and $f(\mathbf{x}) \leq 0$ otherwise. We can use this exact equation as a linear inequality constraint, taking into account which layer the detector uses as input. Table 5.1 lists the correct constraints in each situation.

and ${}^{d}a_0^n$ are not equal, the detector's decision is deterministic. The detector's choice matters even if they are almost equal, since a potential attacker will be looking for a false positive or false negative regardless of how certain the detector is.

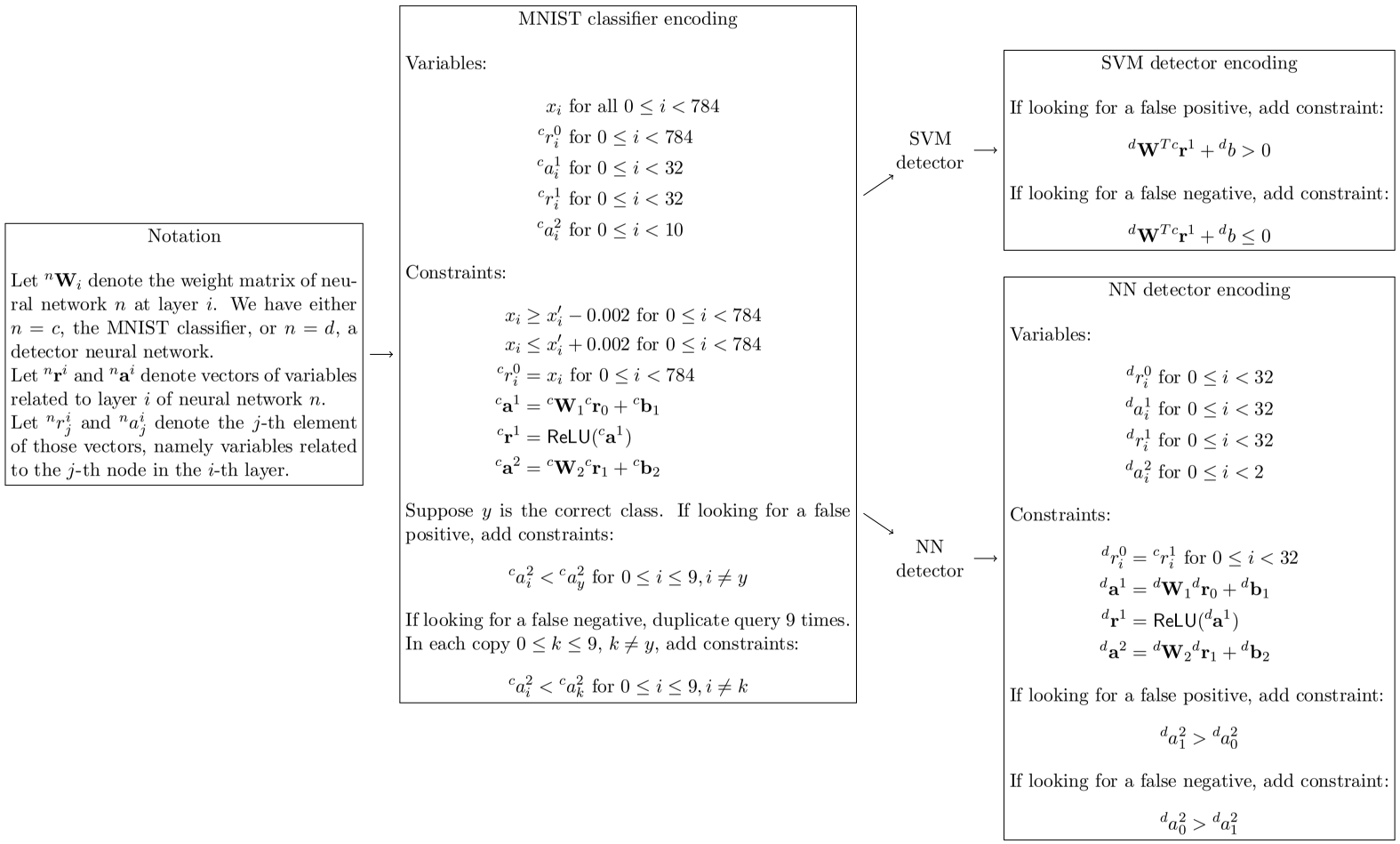



Figure 5.5: Creating an SMT encoding for finding false positives or false negatives in a setting with a neural network that classifies MNIST images into one of ten classes. This diagram assumes that the MNIST classifier has three layers: the input layer with $28 \cdot 28 = 784$ nodes, the hidden layer with 32 nodes and the output layer with 10 nodes. The detector is connected to the classifier's hidden layer. The neural network detector (if chosen) has three layers: the input layer, the hidden layer with 32 nodes and the output layer with 2 nodes.

In summary, this procedure encodes the classification network and the detector as variables and constraints in Marabou and restricts the result to be a false positive or false negative around the original input. This way, we obtain queries to calculate our proposed metrics with Marabou. The total amount of required SMT variables is no more than twice the number of nodes in the classification and detector neural networks. Hence, the number of variables scales linearly with the size of the neural networks.

# Chapter 6

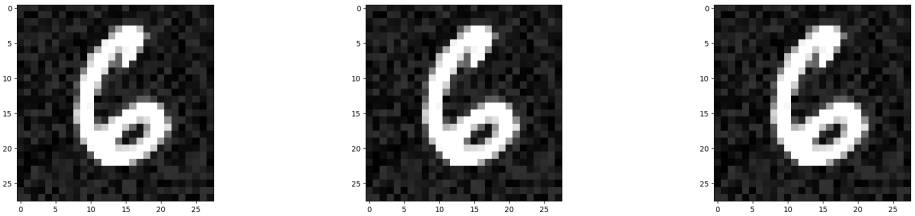
# Results

## 6.1 General results

After training the MNIST classification network, it achieved an accuracy of 0.957 on the test set.

After running the FGSM attack on the noisy MNIST data set, the number of resulting examples that were truly adversarial was 58 408 for the training set and 9540 for the test set. Hence, the size of the entire training and test sets for the detectors, which combined the adversarial examples with the originals, was respectively $2 \cdot 58\,408 = 116\,816$ and $2 \cdot 9540 = 19\,080$.

Figure 6.1 shows a false positive and a false negative generated on one of the inputs by Marabou.



(a) The original image. Classified as "6".

(b) A false negative. Classified as "7", but the detector does not flag it as adversarial.

(c) A false positive. Classified as "6", but the detector flags it as adversarial.

Figure 6.1: False positives and false negatives generated by Marabou on an original image of a handwritten "6". For these examples, the neural network detector with 2 hidden layers and 32 nodes per hidden layer was used, trained on the hidden node values of the MNIST classifier. The distance between the found examples and the original is so small (exactly 0.002 in both cases) that it is hard to notice the difference.

## 6.2 Confusion matrix and SMT scores do not always agree

The confusion matrices and SMT metrics of all setups are listed in Appendix A in Boxes A.1–A.9. Thanks to the small "closeness" bound of 0.002, none of the Marabou queries took longer than one minute to complete. The results are visually summarized with graphs in Figures 6.2 and 6.3.

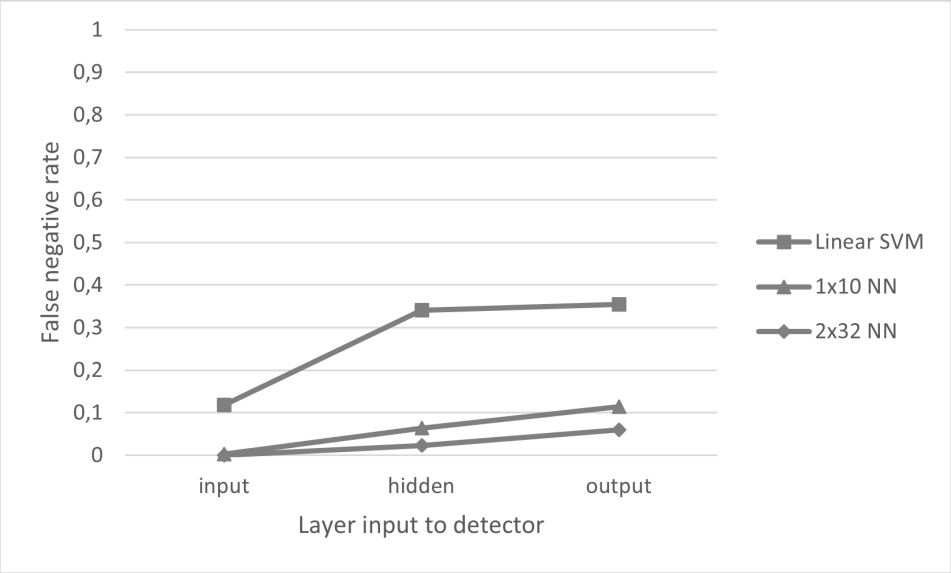
It can be observed from these results that even though the scores in the confusion matrix sometimes look quite good, the new metrics sometimes paint a different picture. For example, the neural network detectors which get the original input image as their image have very few false negatives in their confusion matrices (33 and 2 out of 19 080 examples). However, in both cases, *all* of the 200 original test inputs used for the SMT metrics have a false negative within a distance of 0.002.

## 6.3 False Negatives metrics negatively correlate

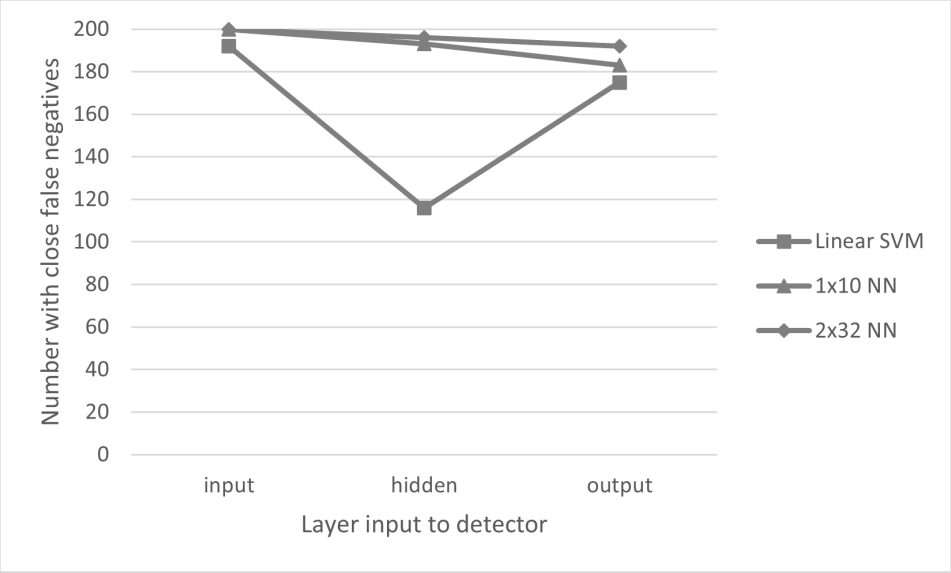
We are interested whether the discrepancy between the confusion matrix and the SMT metrics holds more broadly than for these two examples alone. To do this, we draw a scatter plot to see if the number of false negatives in the confusion matrix correlates to the number of inputs with close false negatives (found with SMT) across different detector setups. The result is shown in Figure 6.4. Pearson's sample correlation coefficient for these sets is $-0.804$. If the confusion matrix is reliable, then one would expect that when the number of false negatives increases, so will the number of inputs with close false negatives. Remarkably, in this case, there is a rather strong negative correlation instead. So for the false negatives case, the confusion matrix scores do not correspond to the scores found with SMT.

## 6.4 False Positives metrics weakly positively correlate

The same approach is taken to compare the number of false positives in the confusion matrix and the number of inputs with close false positives (found with SMT). The result is shown in Figure 6.5. Pearson's sample correlation coefficient for these sets is 0.673. Again, for a reliable confusion matrix, one would expect the number of false positives and the number of inputs with close false positives to have a positive correlation. This indeed appears to be the case, although it is not a strong correlation. So for the false positives case, the confusion matrix scores do correspond somewhat to the scores found with SMT.

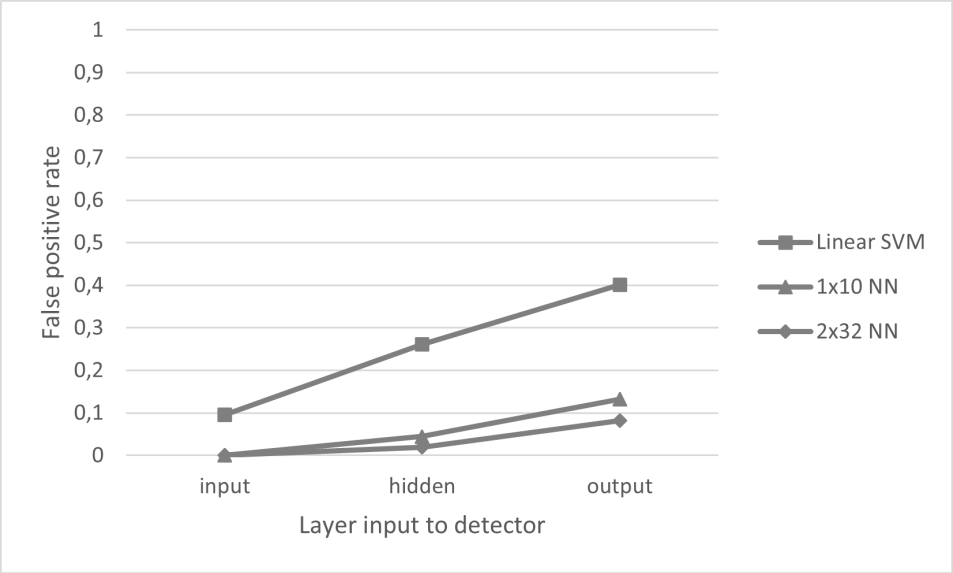



(a) False negative rates for detectors with different inputs.

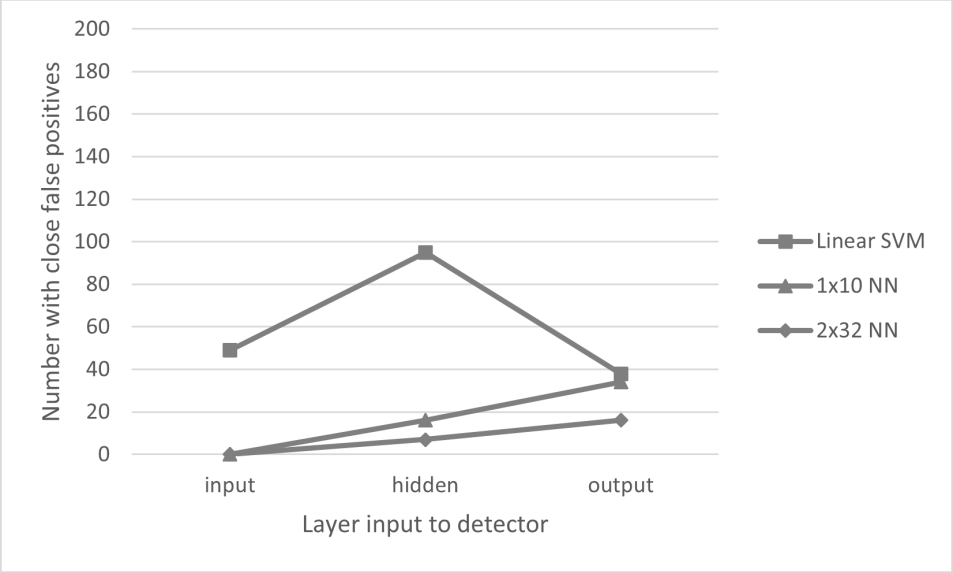



(b) The number out of 200 test inputs with close false negatives for detectors with different inputs.

Figure 6.2: False negatives metrics.




(a) False positive rates for detectors with different inputs.




(b) The number out of 200 test inputs with close false positives for detectors with different inputs.

Figure 6.3: False positives metrics.

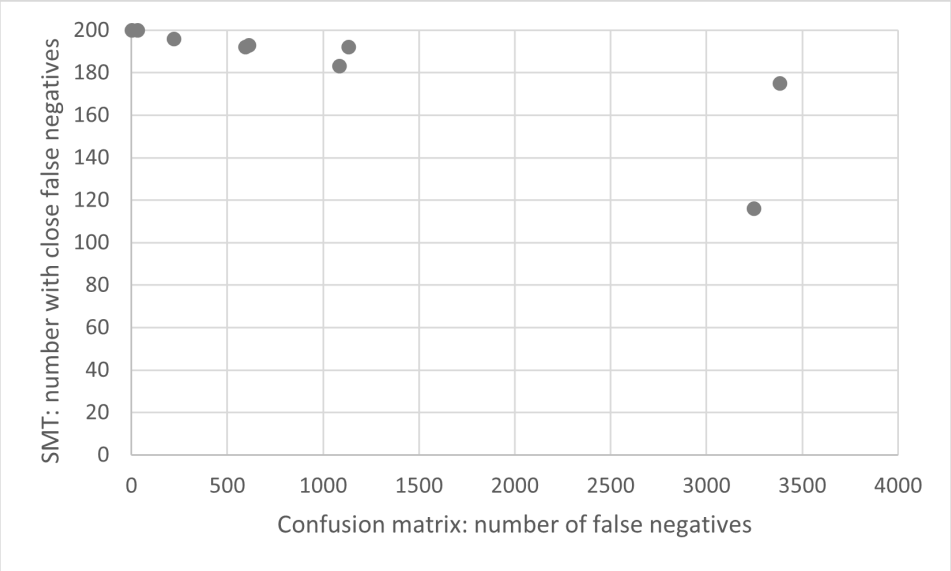



Figure 6.4: Scatter plot investigating a potential correlation between the number of false negatives in the confusion matrix and the number of test inputs with a close false negative. Each dot represents a detector setup.

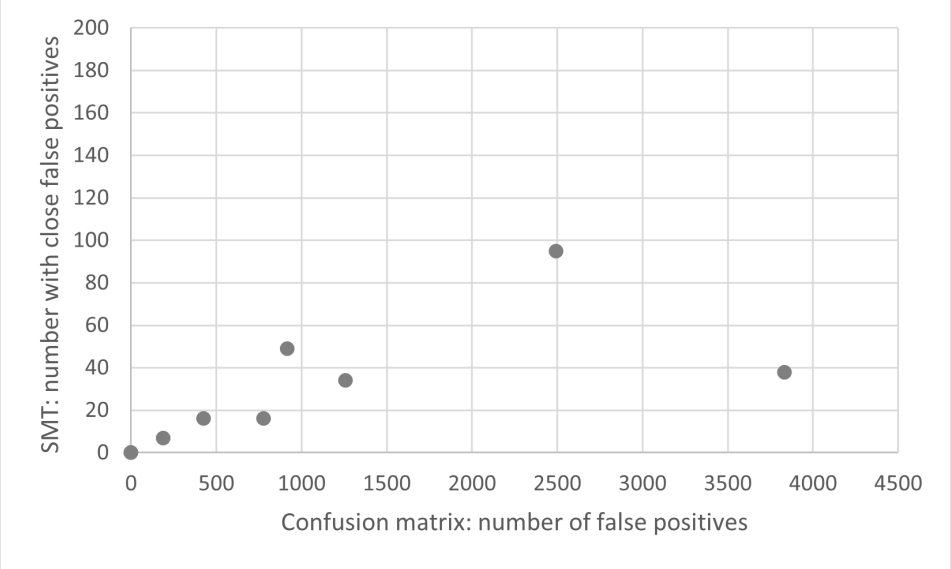



Figure 6.5: Scatter plot investigating a potential correlation between the number of false positives in the confusion matrix and the number of test inputs with a close false positive. Each dot represents a detector setup.

# Chapter 7

# Discussion of the Results

The results from the previous chapter lead to a number of interesting conclusions. In this chapter, we first answer the research question based on the results. Next, we analyze the results to find some additional insights about the tested detectors. We close with a discussion and suggestions for future work.

## 7.1 Added value of the new metrics

Recall that our research question was:

> Does calculating the two proposed metrics (described in Section 4.1) result in information about the quality of detectors that contradicts their confusion matrices?

Our results indicate that this is sometimes indeed the case. For example, in the case of the neural network detectors trained on the original input images, the confusion matrix scores suggest the detector is very good, but the SMT metrics indicate the opposite. Namely, in all of the 200 test cases, an attacker only has to perturb an original input to a distance of no more than 0.002 to obtain a working adversarial example.

In the general case, this added value is also confirmed. This is shown by Figure 6.4 and the calculated correlation coefficient of $-0.804$. If detector A has fewer false negatives than detector B (according to the confusion matrix), then you would expect fewer of the 200 inputs to be close to a false negative for A than to a false negative for B. This negative correlation shows the opposite. Figure 6.2 shows the same: the false negative rates of the detectors go up as the detectors are connected to later layers of the classification network (implying degrading performance), whereas the numbers of inputs with close false negatives generally go down (implying improving performance).

This apparent paradox can be explained by the test set used to calculate the confusion matrix. Recall the main difference between the confusion matrix and the SMT metrics: the confusion matrix is calculated based on a set of adversarial inputs

calculated with the FGSM attack, while the SMT metrics look for adversarial inputs regardless of how they can be generated. Apparently, when there are few false negatives in the confusion matrix, the detector is good at recognizing adversarial inputs generated with FGSM, but worse at recognizing the adversarial examples the SMT solver throws at it.

Since the correlation is negative, it would appear that the detectors are even overfitting on the examples generated with FGSM, and sacrificing some performance on other adversarial examples in the process. As their performance on FGSM adversarial examples improves (according to the confusion matrix), their performance on the SMT adversarial examples becomes worse. In any case, the false negatives metric calculated with SMT adds some very interesting information that cannot be deduced from the confusion matrix alone.

We also need to consider the false positive case. That is, examples that are not adversarial, but still labeled as adversarial by the detector. In this case, there is a positive correlation between the SMT metric and the false positives metric from the confusion matrix. This suggests that even if the detector overfits on FGSM adversarial examples, the detector still does not flag significantly more non-adversarial examples as being adversarial. The difference between the metrics is therefore less extreme than in the false negatives case, Nevertheless, it can still be interesting to see that even though a detector scores fairly good on the false positives metric in the confusion matrix, there are still false positives close to many inputs.

## 7.2 Additional insights

In addition to the value of the SMT metrics, there is a number of other interesting points that can be made when looking at the results from the previous chapter.

First of all, we can compare the two different neural network detectors and the SVM detector. When looking at the test accuracy or the false negative or false positive rates computed from the confusion matrix, the 2x32 neural network performs the best, followed by the 1x10 neural network, followed by the linear SVM. This is confirmed by the SMT metric that measures the number of inputs with close false positives. However, the number of inputs with close false negatives shown in Figure 6.2b shows the reverse order. This may be because the larger neural networks have more parameters and are therefore more prone to overfitting on the adversarial examples present in the training set, as we have discussed before. Hence, to train a good detector, it is important to use a training set with diverse adversarial examples, and a detector that does not have too many parameters, to prevent overfitting. The number of inputs with close false negatives proves to be a good way to check whether a detector is overfitting.

Furthermore, there is an interesting difference in the performance of detectors trained on different inputs. Each detector was trained three times: one copy was trained on the input layer values (i.e. the original images), one on the hidden layer values of the classification network, and one on the output layer of the classification

network. According to the accuracy, false positive rate and false negative rate, each of the detectors performs the best on the input layer, followed by the hidden layer, followed by the output layer. The number of inputs with a close false positive show the same trend, except for an outlier of the linear SVM trained on the output layer. The number of inputs with a close false negative again shows the opposite trend, for which a potential cause has already been discussed.

The fact that the detectors performed better on earlier layers is interesting, since both Metzen et al. [18] and Lu et al. [17] suggested attaching the detectors at deeper layers. This may have to do with the simplicity of our classification network, which only had one hidden layer. In contrast, the mentioned papers used more complicated classification networks such as residual and convolutional networks.

Finally, an interesting point from the results is that, for all tested detector setups, there are more inputs with close false negatives than inputs with close false positives. This means that the inputs are more frequently close to an input that is adversarial but not detected, than to an input that is non-adversarial yet labeled as adversarial by the detector. This suggests that the detectors are more likely to miss an adversarial example than to wrongly flag a non-adversarial input as adversarial. The confusion matrices do not confirm this, however: they do not always report more false negatives than false positives. More research is necessary to investigate this further and to find the root cause of this observation.

## 7.3 Limitations and future work

The research in this thesis has a number of limitations. In this section, several of them are discussed. At the same time, these limitations represent possible directions for future research.

First of all, the experiments in this thesis were only done with classification networks trained on the MNIST data set. As already discussed in Section 5.2, this data set has some potential problems. Some of these problems were reduced by adding noise, as discussed; however, there may be more issues that cause the results of the experiments to be specific to the MNIST data set. To further validate our results, it would be interesting to run similar tests on classification networks trained on other data sets, such as CIFAR-10 [15].

Another limitation of the experiments in this thesis is that the detectors were only trained on sets with adversarial examples generated with the FGSM attack. Arguably, putting adversarial examples generated with multiple (and stronger) attacks could improve the detectors' quality. It would be interesting to see if the metrics calculated with SMT improve when the detectors are trained on such improved training sets.

In addition, this thesis only considered linear SVMs and some fairly simple neural networks as detectors. The SafetyNet detector [17] discussed in Chapter 3 used an RBF-SVM instead of a linear SVM, and Metzen et al. [18] used special training methods for the detectors. In a realistic application, the detector would be tuned

to give optimal adversarial example detection rates. Therefore, applying the experiments in this thesis to better detector setups could make the results more relevant in realistic scenarios.

Furthermore, a choice was made in this thesis to only consider the Chebyshev distance metric and a "closeness" bound of 0.002. It might be interesting to experiment with different definitions of distance, and to attempt to encode those in an SMT solver. It is also worth noting that even if an attacker only has to perturb an image by a distance of no more than 0.002 to find an adversarial example, that does not immediately mean that the classification network is easy to attack. Finding the correct image may still be very hard for an attacker depending on, for example, their knowledge of the network and available computing power.

It would also be interesting to run experiments on classification networks with much larger inputs, such as ImageNet [8]. To classify these images accurately, larger classification networks than the one used in this thesis are typically used, which only contained one hidden layer. The fact that this thesis only experimented with relatively low-dimensional inputs and small classification networks may limit the validity of the results for realistic situations. To run the experiments from this thesis on larger inputs and networks, some work would have to be done to scale the Marabou experiments to larger queries and prevent them from timing out. This could be done by, for example, only considering a subset of the most important input dimensions; or by scaling up the used computing power. There has already been some research that used cloud computing power to run Marabou queries [27]; it might be interesting to see if this is applicable to the queries used in this thesis.

Finally, there may be some opportunities to experiment with some more practical applications. A case study would be interesting to find out if the SMT metrics proposed in this thesis are useful in real-life situations where adversarial attack detectors are used.

# Bibliography


[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (USA, 2016), OSDI'16, USENIX Association, p. 265–283.

[2] Athalye, A., Engstrom, L., Ilyas, A., and Kwok, K. Synthesizing robust adversarial examples. In *Proceedings of the 35th International Conference on Machine Learning* (10–15 Jul 2018), J. Dy and A. Krause, Eds., vol. 80 of *Proceedings of Machine Learning Research*, PMLR, pp. 284–293.

[3] Benenson, R. Classification datasets results, 2021. `http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html`.

[4] Carlini, N., Katz, G., Barrett, C., and Dill, D. L. Provably minimally-distorted adversarial examples, Feb 2018. arXiv:1709.10207.

[5] Carlini, N., and Wagner, D. Adversarial examples are not easily detected. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security* (Nov 2017), ACM.

[6] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms*, 3rd ed. The MIT Press, 2009.

[7] Dantzig, G. B. *Linear Programming and Extensions.* Princeton University Press, 1991.

[8] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition* (2009), IEEE, pp. 248–255.

[9] Glorot, X., and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International*

*Conference on Artificial Intelligence and Statistics* (May 2010), Y. W. Teh and M. Titterington, Eds., vol. 9 of *Proceedings of Machine Learning Research*, PMLR, pp. 249–256.

[10] Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations* (Mar 2015). arXiv:1412.6572.

[11] Gross, D., Jansen, N., Pérez, G. A., and Raaijmakers, S. Robustness verification for classifier ensembles. In *Automated Technology for Verification and Analysis* (2020), Lecture Notes in Computer Science, Springer International Publishing, p. 271–287.

[12] Huang, X., Kwiatkowska, M., Wang, S., and Wu, M. Safety verification of deep neural networks. In *Computer Aided Verification* (2017), Springer International Publishing, pp. 3–29.

[13] Katz, G., Barrett, C., Dill, D. L., Julian, K., and Kochenderfer, M. J. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Computer Aided Verification* (2017), Springer International Publishing, pp. 97–117.

[14] Katz, G., Huang, D. A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., Dill, D. L., Kochenderfer, M. J., and Barrett, C. The Marabou framework for verification and analysis of deep neural networks. In *Computer Aided Verification*, Lecture Notes in Computer Science. Springer International Publishing, 2019, pp. 443–452.

[15] Krizhevsky, A. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, May 2012.

[16] LeCun, Y., Cortes, C., and Burges, C. MNIST handwritten digit database. *ATT Labs [Online] 2* (2010). `http://yann.lecun.com/exdb/mnist`.

[17] Lu, J., Issaranon, T., and Forsyth, D. SafetyNet: Detecting and rejecting adversarial examples robustly. In *2017 IEEE International Conference on Computer Vision (ICCV)* (Oct 2017), IEEE.

[18] Metzen, J. H., Genewein, T., Fischer, V., and Bischoff, B. On detecting adversarial perturbations. In *Proceedings of 5th International Conference on Learning Representations (ICLR)* (2017). arXiv:1702.04267.

[19] Moura, L. D., and Bjørner, N. Satisfiability modulo theories. *Communications of the ACM 54*, 9 (Sep 2011), 69–77.

[20] Papernot, N., Faghri, F., Carlini, N., Goodfellow, I., Feinman, R., Kurakin, A., Xie, C., Sharma, Y., Brown, T., Roy, A., Matyasko, A.,

Behzadan, V., Hambardzumyan, K., Zhang, Z., Juang, Y.-L., Li, Z., Sheatsley, R., Garg, A., Uesato, J., Gierke, W., Dong, Y., Berthelot, D., Hendricks, P., Rauber, J., and Long, R. Technical report on the CleverHans v2.1.0 adversarial examples library. arXiv:1610.00768.

[21] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[22] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *Nature 323*, 6088 (Oct 1986), 533–536.

[23] Serban, A., Poll, E., and Visser, J. Adversarial examples on object recognition. *ACM Computing Surveys 53*, 3 (Jul 2020), 1–38.

[24] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks. In *Proceedings of 2nd International Conference on Learning Representations (ICLR)* (Feb 2014). arXiv:1312.6199.

[25] Tjeng, V., Xiao, K., and Tedrake, R. Evaluating robustness of neural networks with mixed integer programming. In *Proceedings of 7th International Conference on Learning Representations (ICLR)* (Feb 2019). arXiv:1711.07356.

[26] Wiyatno, R. R., Xu, A., Dia, O., and de Berker, A. Adversarial examples in modern machine learning: A review, Nov 2019. arXiv:1911.05268.

[27] Wu, H., Ozdemir, A., Zeljic, A., Julian, K., Irfan, A., Gopinath, D., Fouladi, S., Katz, G., Pasareanu, C. S., and Barrett, C. W. Parallelization techniques for verifying neural networks. In *2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel, September 21-24, 2020* (2020), IEEE, pp. 128–137.

# Appendix A

# Results Data

This appendix lists the confusion matrices and SMT metrics of all tested detector setups.

| | **Actually adversarial** | **Actually normal** |
|---|---|---|
| **Classified as adversarial** | 8408 | 917 |
| **Classified as normal** | 1132 | 8623 |

Test accuracy: 0.8926
Number out of 200 test inputs close to a false negative: 192
Number out of 200 test inputs close to a false positive: 49

Box A.1: Test results for the linear SVM detector trained on the classifier's input layer values.

| | **Actually adversarial** | **Actually normal** |
|---|---|---|
| **Classified as adversarial** | 6291 | 2491 |
| **Classified as normal** | 3249 | 7049 |

Test accuracy: 0.6992
Number out of 200 test inputs close to a false negative: 116
Number out of 200 test inputs close to a false positive: 95

Box A.2: Test results for the linear SVM detector trained on the classifier's hidden layer values.

| | **Actually adversarial** | **Actually normal** |
|---|---|---|
| **Classified as adversarial** | 6156 | 3831 |
| **Classified as normal** | 3384 | 5709 |

Test accuracy: 0.6219
Number out of 200 test inputs close to a false negative: 175
Number out of 200 test inputs close to a false positive: 38

Box A.3: Test results for the linear SVM detector trained on the classifier's output layer values.

| | **Actually adversarial** | **Actually normal** |
|---|---|---|
| **Classified as adversarial** | 9507 | 0 |
| **Classified as normal** | 33 | 9540 |

Test accuracy: 0.9983
Number out of 200 test inputs close to a false negative: 200
Number out of 200 test inputs close to a false positive: 0

Box A.4: Test results for the 1x10 neural network detector trained on the classifier's input layer values.

| | **Actually adversarial** | **Actually normal** |
|---|---|---|
| **Classified as adversarial** | 8926 | 424 |
| **Classified as normal** | 614 | 9116 |

Test accuracy: 0.9456
Number out of 200 test inputs close to a false negative: 193
Number out of 200 test inputs close to a false positive: 16

Box A.5: Test results for the 1x10 neural network detector trained on the classifier's hidden layer values.

| | **Actually adversarial** | **Actually normal** |
|---|---|---|
| **Classified as adversarial** | 8454 | 1257 |
| **Classified as normal** | 1086 | 8283 |

Test accuracy: 0.8772
Number out of 200 test inputs close to a false negative: 183
Number out of 200 test inputs close to a false positive: 34

Box A.6: Test results for the 1x10 neural network detector trained on the classifier's output layer values.

| | **Actually adversarial** | **Actually normal** |
|---|---|---|
| **Classified as adversarial** | 9538 | 0 |
| **Classified as normal** | 2 | 9540 |

Test accuracy: 0.9999
Number out of 200 test inputs close to a false negative: 200
Number out of 200 test inputs close to a false positive: 0

Box A.7: Test results for the 2x32 neural network detector trained on the classifier's input layer values.

| | **Actually adversarial** | **Actually normal** |
|---|---|---|
| **Classified as adversarial** | 9318 | 189 |
| **Classified as normal** | 222 | 9351 |

Test accuracy: 0.9785
Number out of 200 test inputs close to a false negative: 196
Number out of 200 test inputs close to a false positive: 7

Box A.8: Test results for the 2x32 neural network detector trained on the classifier's hidden layer values.

| | **Actually adversarial** | **Actually normal** |
|---|---|---|
| **Classified as adversarial** | 8971 | 776 |
| **Classified as normal** | 569 | 8764 |

Test accuracy: 0.9295
Number out of 200 test inputs close to a false negative: 192
Number out of 200 test inputs close to a false positive: 16

Box A.9: Test results for the 2x32 neural network detector trained on the classifier's output layer values.