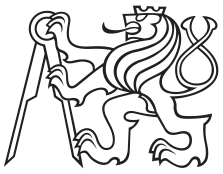
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Creating classifiers robust to adversarial attacks

Bc. Petr Všetečka

Supervisor: Mgr. Viliam Lisý, Ph.D.
Field of study: Open Informatics
Subfield: Artificial Intelligence
April 2018

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Všetečka** Jméno: **Petr** Osobní číslo: **393219**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Umělá inteligence**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Vytváření robustních klasifikátorů vůči manipulovaným datům

Název diplomové práce anglicky:

Creating classifiers robust to adversarial attacks

Pokyny pro vypracování:

Recent research has shown that existing machine learning models can be easily tricked to have high error rate using slight modification of inputs, called adversarial samples. The student will:
1) review automated methods for creating adversarial samples for machine learning and their publically available implementations;
2) review existing methods for creating classifiers robust to such manipulation;
3) implement one of the existing methods;
4) design a novel method for creating robust classifiers based on the algorithms used to compute game theoretic solution concepts;
5) empirically compare the effectiveness of the existing and the new methods on traffic signs recognition and one more data set

Seznam doporučené literatury:

Huang, L., Joseph, A.D., Nelson, B., Rubinstein, B.I. and Tygar, J.D., 2011, October. Adversarial machine learning. In Proceedings of the 4th ACM workshop on Security and artificial intelligence (pp. 43-58). ACM.
Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z.B. and Swami, A., 2016. Practical black-box attacks against deep learning systems using adversarial examples. arXiv preprint arXiv:1602.02697.
Barreno, M., Nelson, B., Sears, R., Joseph, A.D. and Tygar, J.D., 2006, March. Can machine learning be secure?. In Proceedings of the 2006 ACM Symposium on Information, computer and communications security (pp. 16-25). ACM.
McMahan, H.B., Gordon, G.J. and Blum, A., 2003. Planning in the presence of cost functions controlled by an adversary. In Proceedings of the 20th International Conference on Machine Learning (ICML-03) (pp. 536-543).
Fudenberg, D. and Levine, D., 1998. Learning in games. European economic review, 42(3), pp.631-639."

Jméno a pracoviště vedoucí(ho) diplomové práce:

Mgr. Viliam Lisý, katedra počítačů FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **09.02.2018** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **30.09.2019**

Mgr. Viliam Lisý podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____ _____

Datum převzetí zadání Podpis studenta

Acknowledgements

I would like to thank my family and my girlfriend for their continuous support, endless faith and remarkable patience.

I would also like to thank my supervisor Mgr. Viliam Lisý, Ph.D. for his guidance and advices which were necessary stepping stones through the development of this thesis.

Declaration

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne .....

.....

Abstract

The aim of this thesis is to study closely the phenomenon of adversarial samples. It goes through the history of the adversarial attacks, presents a categorization of these attacks and examines several attack algorithms in closer detail. It also presents defense strategies both in theory and in practice. Next, a practical implementation of existing method for creating robust classifiers as well as a novel method using publicly available resources is described in detail, from choosing a dataset to measuring the performance of the system. Lastly, both methods are compared side to side.

Keywords: machine learning, robust classifiers, adversarial samples, convolutional neural networks, traffic signs classifier, Foolbox

Supervisor: Mgr. Viliam Lisý, Ph.D.

Abstrakt

Tato práce se zabývá bližším studiem fenoménu klasifikace manipulovaných dat. Prochází historii těchto útoků, třídí je do skupin a podrobněji rozepisuje fungování vybraných algoritmů. Spolu s útoky popisuje i způsoby obrany, jak teoretické, tak i praktické. Dále předkládá čtenáři detailní popis implementace systému pro vytváření robustních klasifikátorů za použití volně dostupných zdrojů včetně vlastní nové metody, od výběru datasetu až po měření výkonu. Na závěr porovnává obě metody pro zrobustňování klasifikátorů, čistě z veřejných zdrojů a vlastní.

Klíčová slova: strojové učení, robustní klasifikátory, manipulovaná data, konvoluční neuronové sítě, rozpoznávání dopravních značek, Foolbox

Překlad názvu: Vytváření klasifikátorů robustních vůči manipulovaným datům

Contents

Figures

Tables

Chapter 1

Introduction

Neural networks are considerably old concept, first model was created by Frank Rosenblatt in 1957. It received a lot of attention but then the limits of neural networks were discovered to be more restrictive than what researchers hoped for, and the research almost stopped. Rosenblatt's perceptron was a single neuron. As such, it wasn't hard to work with but it was able to classify linearly separable data only. To classify more complicated data, more complicated networks are necessary which requires more power to simulate.

Recently, the neural networks are getting a lot of attention again. One of the reasons is that we are finally getting computation power needed to simulate and train bigger networks. It can still take days but it can be done. Because of that, neural networks are getting more widespread and penetrate into various fields of human activity. Arguably the most famous and talked-about in recent days is the application in autonomous driving vehicles. More information about the history and development can be found in Chapter 2.

But this sudden boom in use means they attract more attention in general. Most importantly for this work, attacks against these networks are more common. Hand in hand with attacks, security needs to be developed and evaluated for these systems. Chapter 3 presents a more in-depth insight in the problematic of both attack and defense. It is often not an easy thing to come up with an appropriate defense strategy, only more so when a new or confusing vector of attack appears, and this is the case for both.

Although the neural networks work really well in solving certain areas of problems, they are very different from human understanding of those problems. This makes it hard for humans to imagine or understand what do the neural networks actually learn and what criteria they base their decisions on, which is critical information for making the final solution more robust. Chapter 2.1 describes how humans found some of these properties and what they are.

Let me present an example with traffic signs. Also, let me leave out the whole process of identifying what is a traffic sign and what is not and just concentrate on classifying them. When a human sees a traffic sign, he compares it with

his knowledge base and finds a class that seems the most probable. Until this point, it is the same with neural networks but the difference between their respective knowledge bases is vast. Humans have prior knowledge about colours and geometric shapes and a great number or various other pictographs which helps with the classification. The neural network does not know about any of this, it has to derive all its knowledge from a number of examples of traffic signs. Therefore it looks for some unique identifiers that only the examples in one class have and uses these found patterns to classify new inputs.

This introduces a big danger for neural networks, because disturbing such a pattern is a very easy feat. Moreover, it often doesn't change the human perception of given sample. After all, a traffic sign with washed out colours or a sticker on the edge is still the same traffic sign for us. But for the neural network, this can easily change the pattern the system learned and make the system classify this new input as a different class. In other words, these are inputs that seem perfectly normal and clear for human perception but they are misclassified by the neural networks. Humans found various ways to create these alterations and named them the *adversarial samples*. More examples are introduced in Chapter 2.2.

There are already many ways to create the adversarial samples. Although it is a new field with only few years since its discovery, a big number of different methods to create the adversarial samples was already created. Many of these come from scientific research and a classification system was put in place. Chapter 3 delves into the depths of creation of adversarial samples. There also exist toolboxes that assist with the attack. One of them, the Foolbox is examined and put to work in this thesis.

Chapter 4 introduces the datasets used in practical examples and experiments. A number of various preprocessing techniques is also discussed there. Chapter 5 on the other hand describes the structure of the classifier chosen to be the basic building block of the system developed in this thesis. The theory of training a neural network classifier is mentioned together with introduction of several techniques used to improve the training process. The results of training the classification model with both datasets are presented too.

The process of making a neural network classifier more robust is described and demonstrated in Chapter 6. First, an existing method is described and evaluated. Then, an idea for improvement is introduced and evaluated as well. Lastly, both methods are compared using different metrics and a conclusion is drawn.

Chapter 2

Related work

There has always been a need for defense against an attacker in automated decision making, either in games [FL98], robot motion planning [MGB03] or in machine learning [BNS+06].

The [BNS+06] specifically thoroughly analyzed and described the potential vectors of attacks on machine learning in general. They focused on examining whether an attacker can manipulate the classification system (and how) to permit a specific attack, or if it is possible for the attacker to significantly degrade the performance of the system. Altogether, they found three axes defining a space of possible attacks:

- `Influence` — *Causative* attacks that can alter the process of training the learner, and *exploratory* attacks that use techniques not interfering with the learning, for example probing the learner. This axis more or less represents the separation into *black-box* and *white-box* later in chapter 3.
- `Specificity` — Deals with the width of area the attacker wants to target. *Targeted* attacks that strive to exploit one particular weakness or service while *indiscriminate* are those attacks where the attacker has a wider scope.
- `Security violation` — This axis defines what the attacker tries to achieve with his attack. He can perform either an *integrity* attack where he seeks to sneak through some badly categorized samples, or an *availability* attack where he tries to lower the performance enough to take down the whole system.

A concise summary of these properties and their mutual relations is shown in Table 2.1. The [BNS+06] also discussed the theory for possible defenses. Similarly as in attack properties, the authors propose three main possible defense vectors:

- `Regularization` which is a technique from statistics used to restrict or bias the choice of the system when it deals with lack of data or noisy data. It is also a good way to keep a high level of generalization of the

		Integrity	*Availability*
Causative:	*Targeted*	Permit a specific intrusion	Create sufficient errors to make system unusable for one person or service
	Indiscriminate	Permit at least one intrusion	Create sufficient errors to make learner unusable
Exploratory:	*Targeted*	Find a permitted intrusion from a small set of possibilities	Find a set of points misclassified by the learner
	Indiscriminate	Find a permitted intrusion	

Table 2.1: The attack model by [BNS+06]. The summary of attack properties and their mutual relations is captured here separated into three axes, together with examples. *Causative* attacks that can alter the process of training the learner, and *exploratory* attacks that use techniques not interfering with the learning. *Targeted* attacks that strive to exploit one particular weakness or service while *indiscriminate* are those attacks where the attacker has a wider scope. *Integrity* attacks where the attacker seeks to sneak through some badly categorized samples, or an *availability* attacks where he tries to lower the performance enough to take down the whole system.

classifying system.

- `Randomization` is a technique easily understood from its name - it adds certain randomness to the decision boundary of the system. This makes it a potential candidate against *targeted* attacks as those are the attacks that try to move the decision boundary past the targeted point.

- `Information hiding` is also easily understood from its name. It represents the effort to hide as much information from the attacker as possible. An alternative name from security engineering circles that gained more renown while basically keeping the same meaning is *security through obscurity*.

A summary of these techniques together with areas in which they are applicable is shown in Table 2.2. There is a trade-off in application of each of these techniques, however. The regularization lowers the effect of legitimate data on the learning process in order to keep high level of generalization. The randomization increases the error rate of the classification system on top of increasing the attacker's work. And finally, the information hiding is usually really hard to put in practice or reinstate once discovered, while also going against the spirit of open-source and the still more popular opinion that only open-source can offer real security and privacy.

Still, most of those defense mechanisms are nowadays usually already implemented in the classification model itself, either directly as a part of the system or in form of some preprocessing (more on this topic in Chapter 4). These attack and defense models are applicable for all machine learning systems but there is one big vulnerability shared by most existing machine learning systems, neural networks in particular, that stands out in present times.

		Integrity	*Availability*
Causative:	*Targeted*	• Regularization • Randomization	• Regularization • Randomization
	Indiscriminate	• Regularization	• Regularization
Exploratory:	*Targeted*	• Information hiding • Randomization	• Information hiding
	Indiscriminate	• Information hiding	

Table 2.2: Theory of defense against the attacks in Table 2.1 [BNS+06], separated in the same categories as the attacks against which they are applicable. The *regularization* is a technique used to restrict or bias the choice of the system when it deals with lack of data or noisy data. *Randomization* adds a certain randomness to the decision boundary of the classification system. Finally, the *information hiding* represents the effort to hide as much information from the attacker as possible.

2.1 Counter-intuitive properties of neural networks

The first to notice this vulnerability were researchers in [SZS+13]. They were interested in learning more about how the neural networks achieve such high performance on visual and speech recognition problems. In their studies they discovered two interesting counter-intuitive properties of neural networks.

- First is the absence of distinction between high level units. There were many other works before that analyzed the semantic meaning of various units [GDDM13, GLS+09, ZF13]. They tried to find the sets of inputs that maximize the activation of given unit. This approach, however, stands on an implicit assumption that the units in the last layer for feature extraction can be efficiently used to extract semantic information. That is an expected intuitive result for human beings. On the contrary, it was proven that a randomly chosen high level unit (or linear combination of thereof) is equally adept to classify the data (i.e. with comparable accuracy) as careful selection of units to select round white flowers for example. This leads to conclusion that neural networks hold the majority of semantic information in the entire space of activations rather than the individual units.

- Second property is the high level of discontinuity of input-output mappings. When dealing with high performance deep neural network, it is common to assume solid generalization, that is to expect that such a system will be able to reliably classify previously unseen samples in correct categories. This system should be robust to small perturbations to its input, perturbations small enough so they are not able to change

the category of the image. A white dog is still a white dog even if it has a bit of grey fur on its ear. Yet, even absolutely imperceptibly small perturbation can change the category of image if crafted for this purpose (see Figure 3.1 for example). These images with imperceptible perturbation are called *adversarial samples*. On top of that, even the best performing neural networks are very susceptible to the adversarial samples. In [SZS+13], they carefully crafted some adversarial samples that were able to trick the network with the network being almost 100% sure about its classification while human could not tell the difference between the crafted adversarial sample and the original picture.

Normally, one would expect these vulnerabilities to be random artifacts originating from supervised learning. But this property was also proven to be shared between different neural networks. It was shown that attackers with access to an independently collected labeled training set from the same population distribution as the one used to train the targeted classifying system could train a model with a different architecture and use it as a substitute for the targeted system. Which means the attackers can find adversarial samples for their substitute and take advantage of the adversarial samples being shared as well. The transferability of changes between different models was studied more in depth in various works and also serves as a base for various black-box attacks [LCLS16]. It is this property and its transferability, that makes the adversarial samples stand out.

2.2 Variety of the adversarial samples application

Since the discovery of adversarial samples, several works have shown that it is surprisingly easy to craft these images in various ways. Adversarial samples pose security concerns because they could be used to perform an attack on machine learning systems, even if the adversary has no access to the underlying model. There are several great examples of what can actually be done. In [KGB16], it was shown that there is no need to feed the adversarial samples directly into the network (which is usually not the case for systems operating in the physical world, for example those which are using signals from cameras and other sensors as input).

Others took a different approach and shown that altering of physical objects in real world and subsequent misclassification of their images is also a real possibility. The researchers in [EEF+17] designed a new algorithm called *Robust Physical Perturbations* which can generate an adversarial sample that is reproducible in real world, on actual physical objects (see Figure 2.1). By simply adding a black and/or white stickers to the object, they are able to achieve 100% success rate in laboratory conditions and over 80% in real world tests, in this case capturing an on-board video from moving car going by modified Stop sign.

Figure 2.1: The image on the left shows a real world graffiti example on a Stop sign. Most humans would not think it is anything unusual, or even suspicious. The image on the right shows a perturbation found by the *Robust Physical Perturbations* algorithm [EEF+17], a brand new Stop sign with black and white stickers added to it to imitate the graffiti. As the authors say: "We design our perturbations to mimic graffiti and thus 'hide in the human psyche'."

Another good example would be demonstration of circumventing the facial recognition systems by simply making a custom pair of glasses [SBBR16]. The researchers here focused on tricking the biometric face recognition systems with several key points in mind. They wanted to make their attacks inconspicuous so no one present to the attack (or watching a recording of the process) would not notice anything out of ordinary, physically realizable and able to deceive actual face recognition systems employed in real world. They also set out two goals, either force the system to recognize the impersonator as a specific authorized user or simply avoid detection as the impersonator himself. They were able to dodge the identification of the impersonator and also successfully impersonate someone similar to the impersonator in all cases. They also achieved about 85% success rate in impersonating basically an arbitrary person without any regard to sex or racial features such as skin color (see Figure 2.2).

(a) (b) (c) (d)

Figure 2.2: Examples of dodging [Figure (a)] with 100% success and impersonations [Figures (b)-(d)] with average about 85% success rate [EEF+17]. The impersonators are in the top row wearing custom glasses, the bottom row shows targets. In order they are Milla Jovovich, the famous actress and model, the participant in the experiment shown in Figure (d) with glasses, and Carson Daly, an American broadcaster.

Chapter 3

Closer examination of adversarial samples

3.1 Classification

There are tens of methods to create adversarial samples already developed. As a consequence, there are various ways to classify these methods. First big division is whether or not the attacker has access to the inner workings of the targeted network. In other words, whether he can obtain knowledge of how the neural network was built, what layers are present, what type of neurons these layers consist of, how they are connected, etc.

This separates possible methods in two groups:

1. `black-box` where the attacker doesn't know anything about the targeted network (all he can do is feed the network some input and observe the outputs), and
2. `white-box` where the attacker has full access to everything.

White-box attacks are much less common in real world, usually the structure of the network is not known or is at least obfuscated to increase protection. Also, as was mentioned in Chapter 2 about defenses, information hiding is a valid defense mechanism despite its drawbacks, even for increasing the security of the systems, and only more so for numerous proprietary private paid systems.

3.1.1 Gradient-based attacks

Another possible approach is to classify the attack methods by main features of the attack algorithm. First of these would be the so called gradient-based attacks. This is a group of algorithms that use gradient of the model on input data to shift the weights just enough to mis-classify the adversarial sample. In other words, these algorithms look at a picture of an airplane, they test which direction in picture space makes the probability of the cat

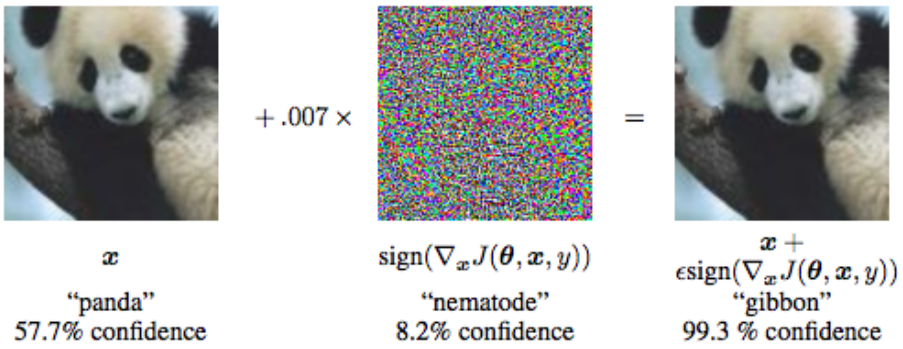


Figure 3.1: A demonstration of fast adversarial example generation applied to GoogLeNet [GSS14] on a sample from ImageNet. On the left is the original input image together with with its label and model's confidence in its prediction. In the middle, the map of gradient signs is shown. Each pixel's RGB value is set to maximum or minimum according to the sign of gradient for given pixel channel. On the right, the created adversarial sample is shown. It is created by taking the original image and adding the created gradient sign map multiplied by very small ϵ. In this case, the $\epsilon = 0.007$. As we can see in the network's label and confidence values, a for humans virtually indistinguishable perturbation radically changes the model's prediction.

class increase, and then they give a little push (i.e. they perturb the input) in that direction. The new, modified image is mis-recognized as a cat. Probably the most famous and used method nowadays is the *Fast gradient sign method* [GSS14].

- `Fast gradient sign method` is an iterative algorithm with high success rate and low time and computational complexity. This method adds the per pixel sign of the gradient to the image, gradually increasing the magnitude until the image is misclassified. To obtain the gradient, it is needed first to compute the derivatives of the loss function according to the pixels of the image. As mentioned in [GSS14], only the sign of the derivative is used. So the output is basically a matrix of the same size as original image, that contains only + or − signs. These signs express the gradient and all that is left to do is to multiply these by some small ϵ to make the next iteration step.

  An example is shown on Figure 3.1. There is an sample image from ImageNet dataset classified by the GoogLeNet [GSS14]. On the left side, the original input image is displayed together with with its label and the GoogLeNet's confidence in its prediction. The input is labeled as x. In the middle, the map of gradient signs is shown. It is created by taking the original picture and finding the direction in picture space that rises the probability of *gibbon* class in the model's prediction. The gradient is a representation of this direction. Each pixel's RGB value in this gradient map is set to maximum or minimum according to the sign of gradient for given pixel channel. On the right side, the created

adversarial sample is displayed. It is created by taking the original image and adding the created gradient sign map multiplied by very small ϵ. In this case, the $\epsilon = 0.007$. It is shown in the network's label and confidence values that a perturbation for humans virtually indistinguishable from the original image, changes radically the model's prediction.

- Other methods in this category are *L-BFGS-B* [TV15] or *DeepFool* [MFF15].

A simple defense is to mask the gradient. However, there are algorithms that can bypass that by various means (substitute model, gradient approximation, etc.). Still, hiding the gradient is solid and efficient protection against these attacks.

3.1.2 Score-based attacks

Other type of attacks could be called score based. These algorithms basically use numerical methods to estimate the gradient and work from here. There are two main algorithms to introduce in this group, *local greedy search* and *single pixel* attack.

- `Single pixel attack` chooses one pixel and sets its value to minimum or maximum value. It is based on the observation that most images actually have some *critical pixels* which have significantly higher influence on model's prediction than the average [NP16]. This algorithm usually runs a predetermined number of iterations and chooses the pixels randomly in hope that it will eventually hit one of the critical pixels.
- `Local greedy search` is based on the idea and shortcoming of the single pixel attack algorithm. A single pixel is not always enough to create an adversarial sample. It might be better to search for a small set of important pixels and perturb them in smaller steps instead. This iterative algorithm first chooses a random pixel in the image. Then it perturbs the pixel and in next iteration it creates a neighbourhood in shape of square with edge size 5 (experimentally found in [NP16]) centered around each pixel perturbed in previous iteration. In this neighbourhood it finds pixel locations using an objective function that minimizes the probability that an adversarial sample with this specific perturbation belongs to its original class. It terminates if it succeeds to push the true label below the kth place in the confidence score vector at any iteration. Otherwise, it proceeds to the next iteration until it hits the predetermined number of iterations.
- There is also a variation of *L-BFGS-B* [TV15] that belongs in this group of attacks since it does not need the gradient, it calculates an approximation and works with that.

The defense strategy against these attacks is to make it harder to compute the gradient estimate. That can be done by adding a stochastic elements

like *dropout* regularization to the model. Another option is to masks both gradients themselves and also their numerical estimate by creating a sharp-edged plateau around the samples [TKP+17].

3.1.3 Transfer-based attacks

These attacks leverage the knowledge about training data used to learn the classification model rather than knowledge on the model itself. They rely on training a substitute model using the same dataset that was used to train the targeted classification system [PMG+17]. Then they make use of the *transferability* property of neural networks as discussed in Chapter 2.1.

A solid defense for this type of attack, based on robust training on a dataset augmented by adversarial examples from an collection of substitute models, was introduced in [PMG+17].

3.1.4 Decision-based attacks

Yet another group of attacks is decision based. Their biggest advantage is that they do not need any extra information (for example knowledge about gradients or probabilities) for their work. These algorithms work usually with blurring the image or reducing contrast or adding some kind of noise until the adversarial sample is constructed. Recently however, a novel algorithm was introduced [BRB17] that achieves much better performance than the rest in this group at the cost of speed.

- `Boundary attack` is a powerful algorithm that works by first adding a big perturbation to the picture to create any adversarial sample. Then, it performs a random walk along the decision boundary between the adversarial sample and the original image. The walk has to follow only two conditions – it needs to stay on the adversarial side of the boundary and every iteration has to lower the distance between the adversarial sample and the original image. When the algorithm cannot make next step while keeping those conditions true or when it reaches maximum number of iteration, the search terminates. A graphic demonstration of the process is shown on Figure 3.2.

Currently, there is no simple or intuitive defense for the group of decision-based algorithms as: *a*) they do not use any information that could be hidden or left out, and *b*) the Boundary attack is a very new addition (yet there is already a method of disturbing the boundary by creating many local minimums, in which the Boundary attack stops farther from the original sample) and the other attack algorithms were mostly overlooked due to their low performance.

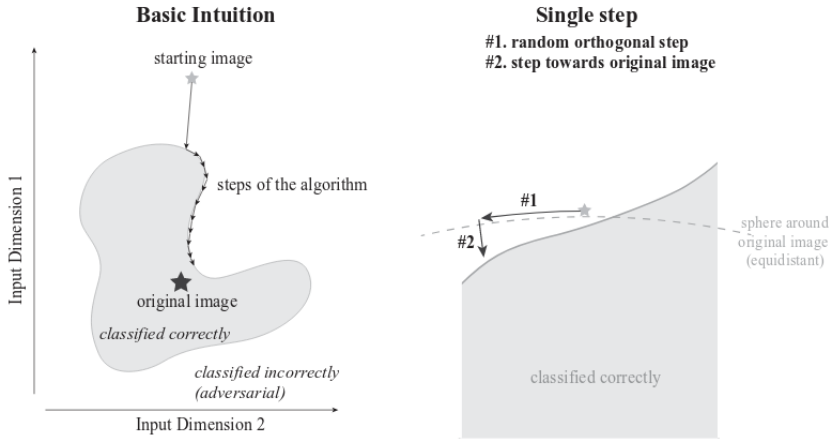


Figure 3.2: The Boundary attack [BRB17] explained. On the left side, the actual process hidden behind the *walk around the boundary* to lower the distance between is depicted. On the right side, a single iteration of drawing a new random direction is demonstrated. First, it projects on a sphere with r equal to the distance between original image and current adversarial sample (#1), then it makes a small step towards the original image without crossing the boundary (#2).

3.2 Foolbox

As was discussed above, there are many different ways of creating adversarial samples. There are also multiple tools to help with that. Out of all these, I decided to use the Foolbox [RBB17].

Foolbox is an open-source python package designed to work with multiple neural network frameworks. It currently contains interfaces for TensorFlow, PyTorch, Theano, Lasagne, Keras and MXNet models.

The whole toolbox is highly configurable, supporting many attack criteria and distance measurements. It provides reference implementations of most published adversarial attack methods including those mentioned above alongside some new ones. Moreover, all the algorithms implemented in Foolbox perform internal hyperparameter tuning to find the minimum adversarial perturbation.

This tool works as a standard python library which provides a big versatility, including implementation of original algorithms or queueing various steps into scripts.

A complete overview of all available attack algorithms currently implemented in Foolbox together with their performance on model described in Chapter 5 trained on the GTSRB dataset described in Chapter 4.1 is shown in Table 3.1.

Category	Algorithm name	Classification times [ms]	Attack times [ms]			L2 distances			Attack fail %
			min	avg	max	min	avg	max	
Gradient-based	Fast Gradient Sign Method	2	10	99.49	2939	0.05	1.49	22.01	0.2
	DeepFool Attack	2	61	138.176	1105	0	0.65	3.56	0
	Saliency Map Attack	2	28	1026.798	12641	0.1	1.58	7.21	0
	Gradient Attack	2	10	120.296	3808	0.05	0.77	9.21	1.8
	L-BFGS-B Attack	2	750	4644.892	18063	0.03	0.6	4.42	0
	SLSQP Attack	2	9431	11719.52	18957	15.74	22.88	28.99	4
Score-based	Local Search	2	38	3613.182	6949	0.48	10.72	19.58	0.4
	Single Pixel Attack	2	5	4367.618	6380	0.66	1.12	1.73	73.4
Decision-based	Gaussian Blur Attack	2	53	745.904	6532	0.05	5.08	20.31	5.6
	Contrast Reduction Attack	2	124	2025.3	3362	0.42	11.25	22.36	4.4
	Additive Uniform Noise Attack	2	168	1657.616	3426	0.1	8.99	22.6	2.6
	Additive Gaussian Noise Attack	2	168	1788.876	3196	0.09	8.48	20.89	5
	Blended Uniform Noise Attack	2	42	1578.924	2918	0.42	11.75	27.95	4.6
	Salt And Pepper Noise Attack	2	1053	2077.612	2643	0.12	6.19	25.45	0
	Boundary Attack	2	1053	2077.612	2643	0.12	6.19	25.45	0

Table 3.1: Foolbox performance comparison. This table shows complete summary of Foolbox algorithms attacking the model described in Chapter 5 trained on the GTSRB dataset described in Chapter 4.1. The attack algorithms are separated in groups according to the classification described in Chapter 3.1 in order they are listed in Foolbox documentation. Next is shown the time for classification a single image to give the reader a perspective on performance. Following are statistics on the attack algorithms themselves, first the times the algorithm needs to successfully create an adversarial sample, then the L2 distances between original and adversarial samples, and finally the percentage of failures for the attack algorithm. Each test was executed on the same five hundred images randomly selected from the GTSRB dataset with the single exception of the *SLSQP* algorithm and the *Boundary* algorithm, which due to their low speed was run only 50 times.

Chapter 4

Datasets

This work utilizes two datasets:

1. `German Traffic Sign Recognition Benchmark` dataset [SSSI11] as the main dataset to present various steps in development and test multiple configurations and results. This is a traffic signs dataset that allows demonstration of many various techniques in a way that is easy to understand for humans.

2. `Modified National Institute of Standards and Technology` dataset [LBBH98] to demonstrate generalization of this work at important points. It is a simple yet famous and very often used dataset of handwritten digits.

4.1 German Traffic Sign Recognition Benchmark dataset

There is quite a lot of different high quality traffic signs datasets because of the boom in popularity of autonomous driving. There are well organized datasets from Sweden [LF11], Belgium [MTBG13] or USA [ST14], to name a few.

For this work I decided to use arguably the most famous and the most used dataset of its type, the German Traffic Sign Recognition Benchmark (GTSRB) dataset [SSSI11] created in 2011 for a classification challenge held at the International Joint Conference on Neural Networks (IJCNN). This dataset comprise of 39 209 training images and 12 630 test images separated into 43 classes, properly labeled and taken from real traffic video around Germany in various weather conditions. The dataset is quite unbalanced (see Figure 4.1) but the distribution in dataset should mostly correspond to the real distribution of traffic signs around Germany, given how it was constructed. All the images are resized to '32 x 32' pixels. Each image

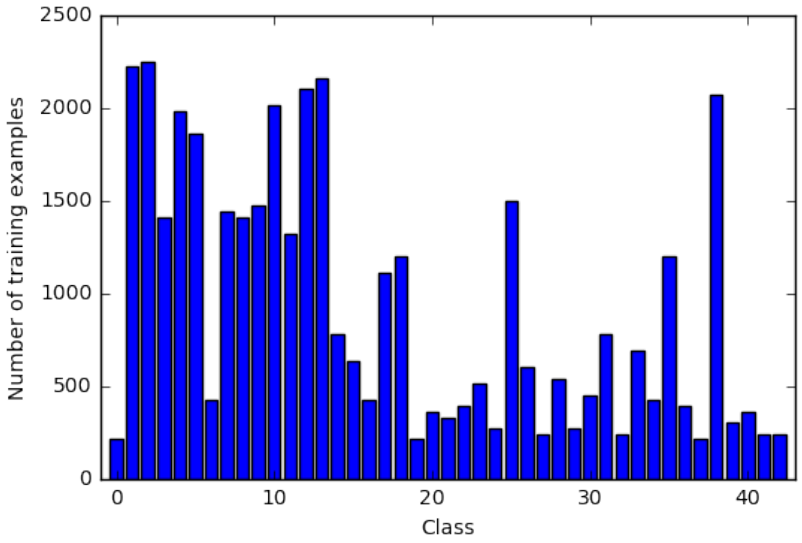


Figure 4.1: The distribution of German Traffic Sign Recognition Benchmark dataset [SSSI11]. The dataset is quite unbalanced but the distribution in dataset basically corresponds to the real distribution of traffic signs around Germany.

therefore is '32 x 32 x 3' matrix of pixel intensities as integers in range $[0, 255]$ representing the R, G and B colour channels respectively.

4.1.1 Preprocessing

In order to use the dataset with state-of-the-art performance, preprocessing is an advantageous step as the images varies greatly both in quality and weather conditions (see Figure 4.2). Typically, preprocessing has several goals to achieve. It modifies the data so it fits target model as every dataset is usually in different format and often more than one dataset is used to demonstrate results. It also improves performance of the classification model by enhancing the important features so they have bigger impact on the training process. It usually also improves speed of the training as it does some part of the work just once ahead of the training process instead of repeating it for every instance and iteration again and again during the actual training.

- The first step of preprocessing for the GTSRB dataset would be to remove colour information from the pictures. This can be best done by taking only the *luma* component of original image (the Y channel from $YCbCr$ representation). This preprocessing step was proposed by Sermanet and LeCun [SL11]. Their experiments show that classification models using greyscale images as inputs achieve at least similar performance as coloured version of the same model, and even that using colour information is often worse than using no colour at all.

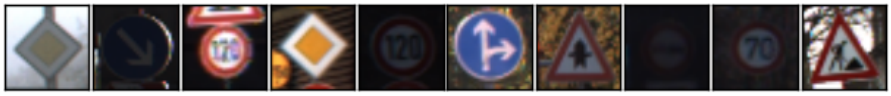


Figure 4.2: Example of original (unchanged) images from German Traffic Sign Recognition Benchmark dataset.

Figure 4.3: Example of the same images as in Figure 4.2 after application of all preprocessing steps — leaving out the colour information, transformation to floats in range $[0, 1]$ for higher precision and equalization of the histogram (the transformation to floats is not important here just yet). Most notably, the advantage of histogram equalization should be obvious on first sight.

- Second step is to scale the range from $[0, 255]$ integer value in the original image to $[0, 1]$ float values. This step is important for two reasons. First, it allows much higher precision of pixel intensity which is a requirement for creation of adversarial samples with very small perturbations. Secondly, it also serves the purpose of fitting the dataset to the model as will be shown in Chapter 5.
- The last step is to correct the brightness and contrast so all the pictures are of comparable quality. This means loosing some information but allows the model to focus on what is important for classification (the Yield sign is still Yield sign whether it is dark, rainy or sunny). The best way to do that is to equalize the histogram, i.e. make the darkest spot absolute black, brightest spot absolute white and proportionally scale the rest of the picture.

When all the steps above are implemented, the final result would look just like Figure 4.3.

However, these changes alter the data before it meets the neural network itself. That effectively puts these alterations out of the model we want to attack using adversarial samples. It makes sense in normal applications because it saves a lot of computation time during both training and testing phase of the neural network. But here it makes the attack algorithms create adversarial samples on data that is not actual input which leads to loss of some data and also is not intuitive for humans (Figure 4.4).

To avoid the full impact of the performance hit associated with leaving out preprocessing completely while keeping the input data as close to reality as possible, it was decided to omit the preprocessing steps that loose information. That means keeping only the transformation to floats as a preprocessing while keeping colour information and skipping the histogram equalization. The model will be working with fully coloured images which increases mostly space

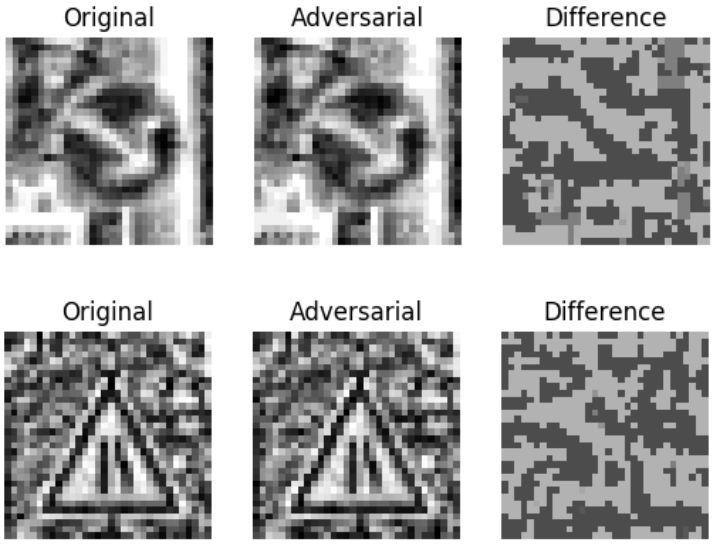


Figure 4.4: Adversarial sample examples after complete preprocessing. On the left are original images, in the middle created adversarial samples and on the right is the difference between the two scaled up for improved visibility.

requirements (triples them) and skips the histogram equalization completely. This allows to work with data that reflects real life applications (Figure 4.5) without any big performance hit.

4.2 Modified National Institute of Standards and Technology dataset

The Modified National Institute of Standards and Technology (MNIST) dataset was created in 1998 [LBBH98]. Despite its age, this dataset is increasingly more popular in recent years, and just in year 2017 it received over 35% of all its citations since creation (according to IEEE and Google Scholar statistics).

The MNIST dataset is a well known collection of handwritten single digits. It has 60 000 samples in the training dataset (split into 55 000 training part and 5 000 validation part) and 10 000 samples in the testing dataset. Each sample was rescaled to '28 x 28' pixels and anti-aliased (see Figure 4.6).

Due to the nature of this dataset, there is no preprocessing necessary, the dataset is distributed as completely ready to be used in training.

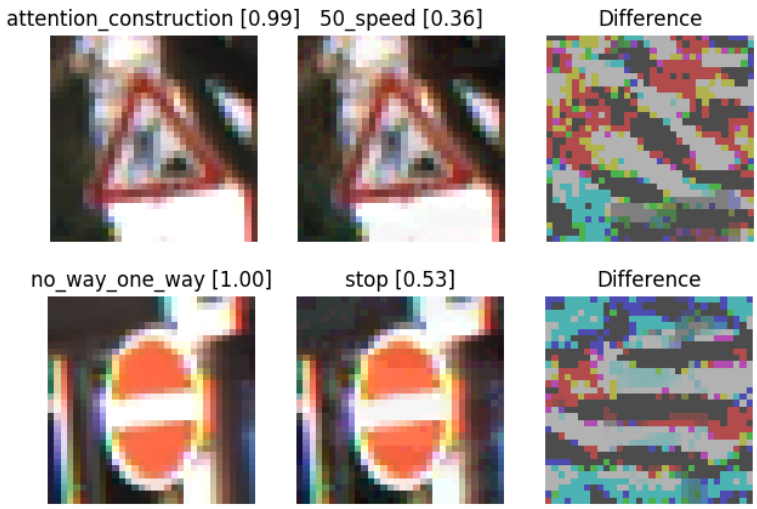


Figure 4.5: Adversarial sample examples on input data with colour information and without histogram equalization. On the left are original images, in the middle created adversarial samples and on the right is the difference between the two scaled up for improved visibility. The images are titled with a label of the prediction of the model trained up to 99% accuracy together with the model's confidence in its prediction (in brackets).

Figure 4.6: Example of 10 samples from the MNIST dataset. On the top row, there is 10 unedited samples from the dataset, on the bottom row are correct labels.

Chapter 5

Model construction

The idea for the model comes from the work of Alex Staravoitau [Sta17] and goes all the way back to the work of Sermanet and LeCun [SL11]. It is a fairly simple deep neural network classifier that has 3 convolutional layers for feature extraction and 1 fully connected layer as the classifier.

The model uses multi-scale features; that is, the output of each of the convolutional layers feeds not only in the subsequent layer but also directly to the last classifying layer. To ensure equal resolution of the final convolutions, all the input skipping the rest of convolution layers is *max-pooled* (discretized/ downscaled on per-sample basis). Figure 5.1 shows full model architecture with all max-pool conversions and their magnitude step by step.

5.1 Regularization

For training process, a number of regularization techniques is used to keep a high level of generalization. The background for regularization as a defense technique was already mentioned in Chapter 2. This is a good goal to achieve in every classifier but it is even more important here where we try to create adversarial samples as close as possible to original inputs.

1. `Dropout` This technique prevents over-fitting by preventing complex adaptations on training data. At each training stage, individual neurons are either kept in the net with probability p, or dropped out of the net with probability $1 - p$. This technique is usually employed on fully connected layers only as those are the most prone to over-fitting. In [Sta17], the author gets a slight improvement in performance when using some dropout on all layers, therefore it is kept on all layers here as well (exact number for each layer is shown in Table 5.1).

2. `L2 regularization` This technique simply adds an additional error, proportional to the squared magnitude of the weight vector, to the error at each node. The L2 regularization has the intuitive interpretation of

	Layer type	# of neurons	p	Dropout
Layer 1	5x5 Convolutional	32	0.9	10% of neurons
Layer 2	5x5 Convolutional	64	0.8	20% of neurons
Layer 3	5x5 Convolutional	128	0.7	30% of neurons
Layer 4	Fully connected	1024	0.5	50% of neurons

Table 5.1: Detailed per-layer description of the dropout regularization used in model described in Chapter 5.1. Lists all layers in the neural network, what type each layer is, how many neurons it consists of, what is the probability p of keeping each neuron in the layer at current training stage, and also what is the dropout of each layer.

heavily penalizing peaky weight vectors and preferring diffuse weight vectors. Due to multiplicative interactions between weights and inputs this has the useful property of encouraging the network to use all of its inputs a little rather than some of its inputs a lot. The higher the proportionality constant, the higher is the penalty for large weight vectors. Here, a value $\lambda = 0.0001$ is used.

3. `Early stopping` As the name suggests, this technique stops the training process before it runs all the way to the end. The limit (also called *patience*) chosen for this model is 100 epochs. It rolls back to the last best-performing model once it detects over-fitting to the training data. The metric used here is *cross entropy loss* as opposed to more commonly used *accuracy*. The reason for that is when the model is confident about its prediction, it should generalize better.

5.2 Implementation

This model is built using TensorFlow and is written in Python.

TensorFlow is an interface for expressing machine learning algorithms, and an implementation for executing such algorithms. It supports a wide range of devices, can run on both CPU and GPU backend, and is often supported by other tools and programs [AAB+16]. The version of TensorFlow used was the most current one, 1.7.0 with GPU support.

Python was used in version 3.5.2 and the Foolbox in version 1.1.0.

5.3 Training

Training was done on original dataset without any modification other than mapping the data samples to model inputs. The learning rate parameter was set to 0.0001. For each epoch, the dataset is randomly split into the training part (75% of the dataset which equals to 29 406 samples) and the validation

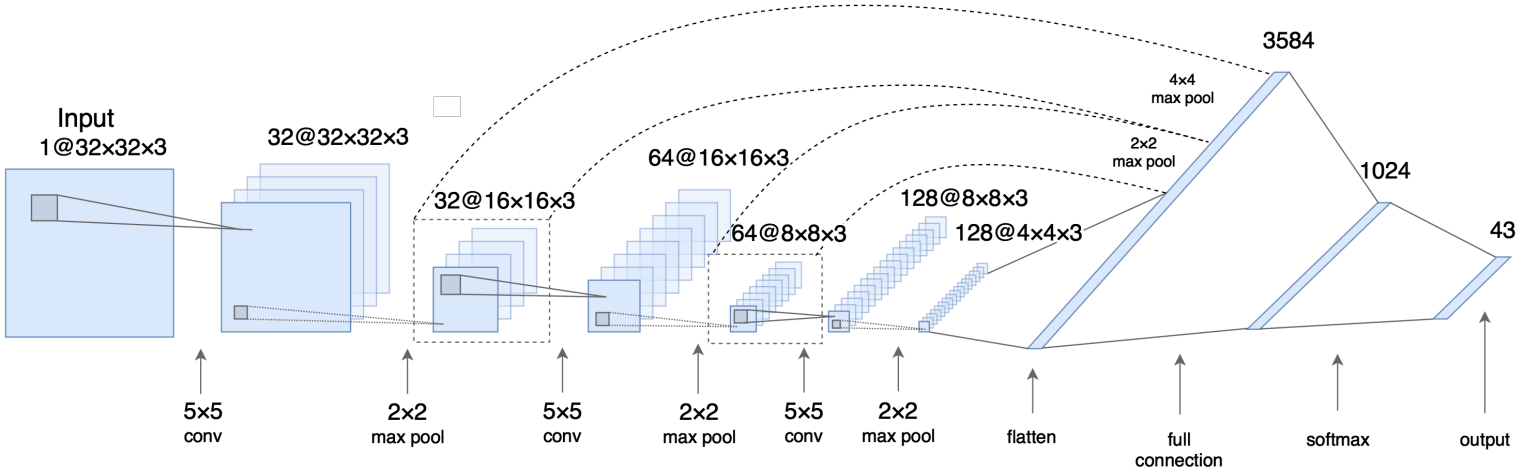


Figure 5.1: Depiction of complete model architecture step by step. It shows acutely where each convolutional layer is as well as from where the bigger-scale feature is extracted and to where it is max-pooled. After all feature extraction comes the flattening down to one dimension vector and feeding this vector into the fully connected softmax classifier. It is based on [Sta17] and expanded to support all colour information.

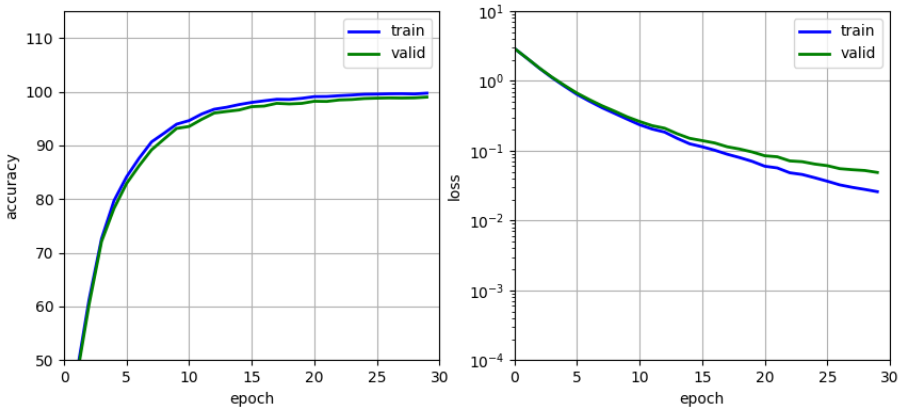


Figure 5.2: Complete history of training the model up to 99% accuracy. On the left, development of accuracy is shown. On the right side is the loss of the model for each epoch.

part (25% of the dataset which equals to 9 803 samples). These parts are fed to the model in randomly selected batches of 256 samples to take advantage of the massive multi-processing capabilities of GPUs while avoiding putting too much data into VRAM at once (the intention here is to avoid overflowing the VRAM) to keep the training speed as high as possible.

In each epoch, first the network is trained on the training part of dataset for this epoch, then evaluated using the validation part. Typically, the validation is performed only every so often (there are various metrics to decide on which epochs the model should evaluate itself) as it is common to train the model for hundreds or thousands of epochs to achieve the highest score possible. Since the goal of this work is not to achieve the highest classification score, a 99% accuracy was decided to be the goal for this part.

The model described in this chapter achieves 99% accuracy on training dataset in 21 epochs and in 30 epochs on validation dataset (for development of accuracy of the model see Figure 5.2 or Table 5.2). The model in this state was saved and used as a base for the next step.

5.3.1 MNIST dataset

Until now, only the GTSRB dataset was discussed. However, as was mentioned in Chapter 4, at several points of interest, the second dataset MNIST will be included. The same model as described above can be also used to classify the MNIST dataset.

Since the dataset was created to be ready to use without any preprocessing, the changes are not extensive. First, the input size needs to be downscaled to a square with edge length of 28 pixels. Same goes for output, the model needs to choose only one out of 10 classes instead of original 43. Lastly, the

	Epoch	1	2	3	4	5	30
940M	Epoch time [s]	46	43	44	43	43	43
	Train accuracy [%]	27.1	47.3	61.2	72.6	79.7	99.7
	Valid. accuracy [%]	26.8	46.5	60.3	72.1	78.3	99
1050Ti	Epoch time [s]	9	8	8	8	8	8
	Train accuracy [%]	28.4	46.3	61.5	74	79.5	99.6
	Valid. accuracy [%]	28.2	45.2	60	72.4	78.2	98.9

Table 5.2: Overview of training process of the model on GTSRB dataset described in Chapter 4.1. A comparison of speed between training on Nvidia GTX 940M and Nvidia GTX 1050Ti is shown here as well as the randomness present in the process of training a neural network. The difference in performance is not so significant here but it will be much more important when the attacks come into the equation.

model can be simplified to one channel as this dataset is greyscale by default.

Such a model achieves the 99% accuracy threshold in just a 10 epochs. See Figure 5.3 for step-by-step development.

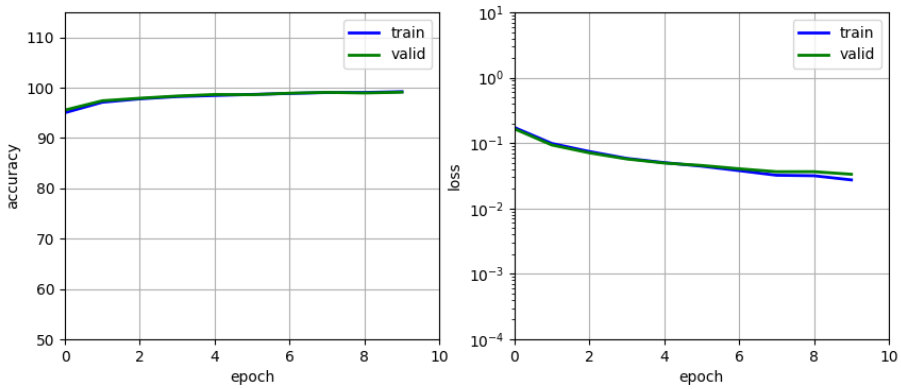


Figure 5.3: Complete history of training the model up to 99% accuracy on the MNIST dataset. On the left, development of accuracy is shown. On the right side is the loss of the model for each epoch.

Chapter 6

Making the classifier more robust

6.1 Practical black-box attacks and defenses

As discussed in Chapters 2 and 3, there are many various ways to attack a neural network classifier. However, since the first discovery of adversarial samples only dates as far as 2013 [SZS+13], there are not many techniques categorized as *black-box* attacks. First practical example came in 2017 [PMG+17]. This work demonstrates viability of adversarial attacks against completely unknown models with two fairly simple algorithms (the *Goodfellow et al. algorithm* [GSS14] and the *Papernot et al. algorithm* [PMJ+15]). Their method of attack in general is to:

1. perform a training on substitute deep neural network
   a. collect a small set of inputs and outputs
   b. choose an architecture of the substitute (based on type of data to classify)
   c. train the substitute on the set obtained by repeatedly querying the original model
2. use this substitute model to craft an adversarial sample
3. verify that this adversarial sample works on original model as well

Because of its popularity and many citations and practical uses, this technique is used in this work as a reference to compare with. The algorithm chosen to work with is the *Goodfellow et al. algorithm*, also known as the *Fast Gradient Sign Method* (described in detail in Chapter 3.1.1), under which name it is present in Foolbox. The *Papernot et al. algorithm* is also present in Foolbox, under the name of *Saliency Map Attack*. However, due to its over 10 times lower speed (average time needed to create an adversarial sample on the model described in Chapter 5 trained on the GTSRB dataset is over 1 second while the FGSM has the average of 100 milliseconds), it was substituted with the *DeepFool Attack* algorithm which has performance

comparable to the FGSM (i.e., average attack time of 140 milliseconds). The second algorithm works as a representation for other adversarial algorithms to demonstrate the transferability of robustness (or lack of thereof) against the adversarial samples between different attack algorithms. For more details refer to Table 3.1.

It would be impossible to recreate all the pieces from scratch in scope of this thesis. Fortunately, the Foolbox takes care of the attacking procedure (see Chapter 3.2 for details on Foolbox). As a result, after correct setting of the environment and connecting all the pieces from previous chapters together, the necessary code can be written in 3 lines.

```
model = foolbox.models.TensorFlowModel(
                            tf_x_batch, logits, range)
attack = foolbox.attacks.FGSM(model)
adversarial = attack(image, label)
```

The last step after that is to use the Foolbox in accordance with [PMG+17] as a tool to make the classifier more robust. That means to create an adversarial sample to each sample used to train the network and use the adversarial sample to train the network as well, with the same level of significance as the original sample. Since there is a big number of random selections in the process, the solution is to intercept the batch when it is going to be used for training, generate corresponding batch of adversarial samples, and send these two batches to the network as one batch for training (see Chapter 5.3 for closer explanation on the training process). This can be done without taking any effect on the actual process due to the batching nature of the training process (i.e., the order of pictures in one batch is irrelevant to the training process).

This procedure effectively doubles the amount of samples the network is trained on, however the time needed to perform this kind of training rises many times over due to adversarial sample creation being about at least 50 times more demanding than simple classification (see Table 3.1). On top of that, the work and time needed to create an adversarial sample increases proportionally with the model's resistance to the adversarial samples, which even further prolongs the training process. Tables 6.1 and 6.2 display the training process to make the classifier more robust as described in [PMG+17]. Table 6.5 shows the comparison of performance between both GPUs. It is also the last experiment executed on the GTX 940M as the time required to get results is already several days.

Classification accuracy and adversarial sample accuracy shows how well the model classify samples from given dataset, original and adversarial respectively. The attack accuracy shows how many times the attack successfully finds an adversarial sample. A consequence to that is that the sum of percentage from correct classification and attack accuracy can go over 100% since some samples can end up in both groups.

Epoch	0	1	2	3	4	5	6	7	8	9	10
Training time [minutes]		44	58	67	79	95	106	115	123	131	138
Validation time [minutes]		27	31	35	40	45	47	50	52	53	55
Classification accuracy [%]	99	96.73	97.51	97.61	98.1	98.17	98.37	98.53	98.62	98.48	98.62
Adver sample accuracy [%]		2.96	3.58	5.43	7.3	9.36	10.43	12.03	12.84	13.59	14.14
FGSM accuracy [%]		97.1	96.6	94.7	92.8	90.7	89.7	88.1	87.4	86.6	86
FGSM average time [ms]		66	77	84	91	94	93	91	92	93	98
FGSM L2 distance		2.27	2.74	3.02	3.23	3.37	3.3	3.28	3.31	3.37	3.53
DeepFool accuracy [%]		100	100	100	100	100	100	100	100	100	100
DeepFool average time [ms]		67	69	70	70	72	72	73	73	73	75
DeepFool L2 distance		0.89	0.97	1.01	1.02	1.07	1.07	1.09	1.1	1.12	1.14

Table 6.1: Summary of first ten epochs of adversarial training using *FGSM algorithm* as described in [PMG+17]. The initial model, displayed in column under epoch 0, is described in Chapter 5 and trained on the GTSRB dataset up to 99% accuracy. The table shows time needed to train and then evaluate the model together with the model's accuracy on both the original dataset input and on-spot created adversarial samples. Next are the statistics for FGSM algorithm (which it is being trained on) and DeepFool algorithm to represent the transferability of robustness against adversarial samples. A comparison with Table 6.2 shows that both algorithms perform in similar fashion.

Epoch	0	1	2	3	4	5	6	7	8	9	10
Training time [minutes]		45	57	69	80	96	107	116	124	132	138
Validation time [minutes]		28	31	34	40	44	46	49	52	56	55
Classification accuracy [%]	99	96.68	97.23	97.05	97.56	97.75	97.91	97.95	98.27	98.48	98.32
Adver sample accuracy [%]		8.79	9.32	9.92	12.49	14.34	15.05	16.34	17.93	19.63	19.5
FGSM accuracy [%]		97.1	96.6	95.7	93.6	91.2	90.6	89	87.7	86.3	85.7
FGSM average time [ms]		69	79	88	93	93	94	92	99	98	93
FGSM L2 distance		2.32	2.75	3.08	3.36	3.25	3.31	3.28	3.5	3.47	3.33
DeepFool accuracy [%]		100	100	100	100	100	100	100	100	100	100
DeepFool average time [ms]		68	71	72	74	74	75	75	76	79	76
DeepFool L2 distance		0.91	0.96	1	1.03	1.06	1.08	1.09	1.1	1.12	1.14

Table 6.2: Summary of first ten epochs of adversarial training using *DeepFool algorithm* as described in [PMG+17]. The initial model, displayed in column under epoch 0, is described in Chapter 5 and trained on the GTSRB dataset up to 99% accuracy. The table shows time needed to train and then evaluate the model together with the model's accuracy on both the original dataset input and on-spot created adversarial samples. Next are the statistics for FGSM algorithm to represent the transferability of robustness against adversarial samples and DeepFool algorithm (which it is being trained on). A comparison with Table 6.1 shows that both algorithms perform in similar fashion.

Epoch	1	2	3	4	5	6
Training time [minutes]	107	131	184	227	257	281
Validation time [minutes]	61	76	95	108	116	123
Classification accuracy [%]	98.65	98.83	98.84	99.02	98.89	98.88
Adver sample accuracy [%]	2.15	4.25	8.41	10.75	12.32	13.52
FGSM accuracy [%]	97.8	95.7	91.6	90.8	89.41	89.1
FGSM average time [ms]	158	196	193	200	206	213
FGSM L2 distance	2.57	3.14	3.07	3.2	3.29	3.41
DeepFool accuracy [%]	100	100	100	100	100	100
DeepFool average time [ms]	148	146	147	151	153	153
DeepFool L2 distance	0.98	1.03	1.06	1.11	1.12	1.14

Table 6.3: Summary of first six epochs of adversarial training using FGSM algorithm as described in [PMG+17], this time trained on Nvidia GTX 940M. The table shows time needed to train and then evaluate the model together with the model's accuracy on both the original dataset input and on-spot created adversarial samples. Next are the statistics for FGSM algorithm (which it is being trained on) and DeepFool algorithm to represent the transferability of robustness against adversarial samples. A comparison with Table 6.1 shows again the level of influence caused by the randomization of training a neural network classifier. The differences in accuracy and L2 distance values are basically negligible in this context and should be evened out if a much higher number (at least hundreds) of epochs was trained. The difference in performance in terms of speed, however, is much higher and is the reason why the rest of the experiments is executed on the more powerful Nvidia GTX 1050Ti.

6.2 Caching the adversarial samples

As can be seen in the previous section, the model can be trained to be more robust against one particular attack algorithm. Unfortunately, it comes at a big price. The time needed to train the model increases by over a 1 000 times. The normal training takes about 8 seconds per epoch as is shown in Table 5.2, for a total of approximately 81 seconds for first 10 training epochs. However, the training of first 10 epochs using the method from Chapter 6.1 takes grand total of 1 391 minutes for training using FGSM algorithm and 1 399 for training using the DeepFool attack.

The main reason for this colossal difference is the ineffectiveness of using hardware. As was shown in Table 3.1, the justifiable difference for untrained (on adversarial samples) classifier is about 50 times. The utilization of GPU during original training as described in Chapter 5.3 was constantly 100%. However, the utilization went down to 40% during the adversarial training from previous section. The structure of the algorithm is to blame, as it requires to create an adversarial sample to each original sample at the time of training that sample. The highly efficient continuous batch training is interrupted to create the corresponding adversarial samples every time. As was described in Chapter 6.1, the implementation of [PMG+17] algorithm

	Training dataset	Testing dataset
Original dataset size	39 209	12 630
Time needed to create the cache	39 minutes	11 minutes
Created adversarial samples	38 426	11 271

Table 6.4: The statistics related to creation of cache of adversarial samples. The first row shows how many samples are in each dataset. Next is the time needed to go through the whole dataset in batch. Lastly, the number of successfully created adversarial samples is shown. The success rate of the attack algorithm was 100% which means the rest are samples that were already misclassified.

used in this thesis works with whole batches instead of single samples, but it is obviously still not good enough.

6.2.1 Simple caching method

Since it is not possible to effectively create the adversarial samples in the middle of training, a solution is to create a cache of the adversarial samples (potentially multiple layers cache, i.e., have multiple adversarial samples cached for each original sample), fill it before executing the training process, and use this cache during the training instead of creating a new adversarial sample on the spot. The result of this process for one layer of cache is shown in Table 6.4. The time needed to do this operation clearly does not suffer from the same drop in speed as the previous method. The most probable reason for that is the fact that there is only one model present while in method described in Chapter 6.1, there is a need to shuffle two models (the training model and the Foolbox model) and perform operations on both of them in turns, over a hundred times each epoch.

The results of training using the cached adversarial samples are shown in Table 6.5. Also, a new metric is present, showing how many adversarial samples in the cache are valid for each epoch. The estimation that the reason for very low speed of adversarial training is related to the creation of adversarial samples was proven true. The same 10 epochs that took 1 391 minutes (23 hours and 11 minutes) are completed in 32 minutes now. Another positive result of this experiment is the proof that there is seemingly no need to have more than one adversarial sample to one original input thanks to the fact that approximately 95% of the cached adversarial samples transfer between epochs (they are still valid adversarial samples).

Unfortunately, another problem came up instead. The accuracy for classification of adversarial samples stays basically zero, although after 5 epochs some small fraction of the adversarial samples starts to be classified correctly. Due to the vast improvement in terms of speed, it is possible to try training more epochs and observe the behaviour of the model over more iterations. Table 6.6 shows the same method using DeepFool attack as its

Epoch	0	1	2	3	4	5	6	7	8	9	10
Training time [minutes]		2	2	2	2	2	2	2	2	3	3
Validation time [minutes]		1	1	1	1	1	1	1	1	1	1
Classification accuracy [%]	99	82.12	91.81	94.58	95.1	95.73	96.31	96.57	96.56	97.02	96.97
Valid adversarial samples		28 024	27 942	28 211	28 453	28 549	28 629	28 693	28 781	28 791	28 826
Adver sample accuracy [%]		0	0	0	0	0	0.07	0.03	0.1	0.19	0.13
FGSM accuracy [%]		100	100	100	100	100	99.95	99.97	99.92	99.82	99.88
FGSM average time [ms]		49	51	58	67	84	75	80	99	106	84
FGSM L2 distance		1.68	1.62	1.96	2.4	2.93	2.59	2.83	3.33	3.71	2.84
DeepFool accuracy [%]		95.3	95.81	96.75	97.26	98.14	97.97	98.45	98.46	98.27	98.21
DeepFool average time [ms]		66	70	69	78	78	85	80	81	80	80
DeepFool L2 distance		0.67	0.6	0.69	0.71	0.8	0.76	0.8	0.81	0.85	0.74

Table 6.5: Summary of first ten epochs of simple caching method using the *Fast Gradient Sign Method* as its base algorithm. The initial model, displayed in column under epoch 0, is described in Chapter 5 and trained on the GTSRB dataset up to 99% accuracy. The table shows time needed to train and then evaluate the model together with the model's accuracy on both the original dataset input and the created adversarial samples. Next are the statistics for FGSM algorithm (which it is being trained on) and DeepFool algorithm to represent the transferability of robustness against adversarial samples. The table acts as a direct comparison with Table 6.1 with added metric showing how many adversarial samples were used from cache instead of being generated in the middle of training process.

base algorithm over the span of 100 epochs.

As can be seen in Table 6.6, training of the model for more epochs using the simple caching does not improve the classifier's accuracy on adversarial samples. It does make the work of the attacker more difficult as it increases both the time needed to create an adversarial sample as well as the distance between the adversarial sample and original image.

There are two possible explanations for this behaviour. Either the adversarial samples "age" in some way and they are not effective on training the classifier when used repeatedly, or they need to be crafted in the middle of the training process and not beforehand. Both these reasons are not very intuitive but the same can be said about the very existence of adversarial samples. Luckily, there is a test to try the potential "aging" of the adversarial samples even though it means a loss of most of the speed originally gained by caching. Table 6.7 shows the behaviour of the model when the whole cache is invalidated and recreated each epoch right before the training process begins.

As can be seen in the the table, there is no difference that could be said to be out of scope of random error. Obviously, there is not problem with "aging" of the adversarial samples. It seems there is a need for the adversarial sample to be created in the middle of the training process in order to make good use of the adversarial sample to make the classifier more robust.

6.3 Comparison of the practical black-box method and the simple caching method

6.3.1 Results on GTSRB dataset

As the previous tables have shown, the practical black-box method has steady performance in the most important aspect of making the classifier more robust, the number of adversarial samples correctly classified. The transferability between different algorithms was proven to be negligible as previous research also confirmed [PMG+17]. However, this method suffers from big losses in terms of speed.

The simple caching method displays over a thousand times better speed and also slightly better L2 distances between adversarial samples and original data (due to higher number of epochs), however it fails completely when it comes to accuracy.

6.3.2 Results on MNIST dataset

Next, both methods were evaluated on the MNIST dataset. Table 6.8 shows the original black-box method [PMG+17] using the *Fast Gradient Sign Method*

Epoch	0	1	2	3	5	10	20	40	60	80	100
Training time [minutes]		3	3	2	2	2	2	3	4	7	9
Validation time [minutes]		1	1	1	1	1	1	1	1	1	1
Classification accuracy [%]	99	87.94	93.79	94.66	95.44	96.65	98.52	99.15	99.31	99.36	99.25
Valid adversarial samples		28 054	28 047	28 299	28 568	28 888	28 988	28 758	28 073	27 109	26 151
Adver sample accuracy [%]		0	0	0	0	0	0.01	0.01	0.05	0.03	0.04
FGSM accuracy [%]		100	100	100	100	100	100	100	99.96	99.98	99.97
FGSM average time [ms]		51	52	49	59	66	63	75	113	144	155
FGSM L2 distance		1.71	1.65	1.63	2.04	2.42	2.24	2.74	4.23	5.37	5.74
DeepFool accuracy [%]		95.92	96.81	97.67	98.37	98.97	99.25	99.32	98.92	98.75	98.87
DeepFool average time [ms]		69	71	65	73	94	79	94	105	115	126
DeepFool L2 distance		0.72	0.64	0.6	0.72	0.91	0.7	0.76	0.79	0.99	0.91

Table 6.6: Summary of 100 epochs of simple caching method using *DeepFool algorithm* as its base algorithm. The initial model, displayed in column under epoch 0, is described in Chapter 5 and trained on the GTSRB dataset up to 99% accuracy. The table shows time needed to train and then evaluate the model together with the model's accuracy on both the original dataset input and the created adversarial samples. Next are the statistics for FGSM algorithm to represent the transferability of robustness against adversarial samples and DeepFool algorithm (which it is being trained on). Just like Table 6.5, it has added metric showing how many adversarial samples were used from cache instead of being generated in the middle of training process.

Epoch	0	1	2	3	4	5	6	7	8	9	10
Training time [minutes]		36	42	45	49	51	53	55	57	58	60
Validation time [minutes]		1	1	1	1	1	1	1	1	1	1
Classification accuracy [%]	99	96.5	97.79	98.13	98.32	98.44	98.55	98.68	98.74	98.88	98.87
Valid adversarial samples		0	0	0	0	0	0	0	0	0	0
Adver sample accuracy [%]		0.04	0.12	0.12	0.17	0.11	0.13	0.06	0.08	0.11	0.05
FGSM accuracy [%]		99.97	99.89	99.89	99.83	99.92	99.87	99.94	99.92	99.89	99.95
FGSM average time [ms]		76	119	111	100	134	144	131	159	135	162
FGSM L2 distance		2.53	4.52	4.18	3.75	4.86	5.46	5.14	6.02	5.2	6.08
DeepFool accuracy [%]		96.09	98.07	98.71	98.85	99.03	99.36	99.15	99.2	99.22	99.32
DeepFool average time [ms]		75	97	105	103	116	130	111	139	130	144
DeepFool L2 distance		1.06	1.76	1.88	2.12	2.1	2.91	2.12	2.45	2.45	2.86

Table 6.7: Summary of ten epochs of simple caching method with cache invalidation using *DeepFool algorithm* as its base algorithm. The initial model, displayed in column under epoch 0, is described in Chapter 5 and trained on the GTSRB dataset up to 99% accuracy. The table shows time needed to train and then evaluate the model together with the model's accuracy on both the original dataset input and the created adversarial samples. Next are the statistics for FGSM algorithm to represent the transferability of robustness against adversarial samples and DeepFool algorithm (which it is being trained on). The "Valid adversarial samples" field shows that the cache is being invalidated every epoch.

for generating the adversarial samples. As expected, the model was trained to be more robust at higher rate, due to its simplicity. However, the time needed to train an epoch grew by a lot, 489 minutes vs 44 minutes on GTSRB dataset (Table 6.1) on first epoch. Also, the much higher defense rate shows clearly that the attack algorithms in general are successful at attacking a classifier robust to one different attack algorithm.

The performance of Simple caching method on the MNIST dataset is shown in Table 6.9. The creation of the cache was again much longer, taking 3 hours and 32 minutes for 55 thousand samples. In comparison, the cache for GTSRB dataset was created in 39 minutes for approximately 39 thousand images (see Table 6.4). As the table shows, the simple caching method works on MNIST dataset a little different than on the GTSRB dataset.

It is clearly shown that the Simple caching method does produce the expected result of making the classifier more robust. It also performs an epoch faster than the original method. On the other hand, the robustness increases a little slower when compared to the original method. Still, even in three epochs the training speed surpasses that of the original method so I would say the method works as intended.

The much longer times on MNIST dataset (when compared with the GTSRB dataset) are partly due to almost twice as big training dataset (less than 30 thousands on the GTSRB dataset versus 55 thousands on the MNIST dataset), but the most important reason is probably related to the increased difficulty of finding an adversarial sample. Significantly higher times for generation of an adversarial sample can be observed for all the training performed on the MNIST dataset. The L2 distances between original images and the adversarial samples also increased significantly and the robustness of the model also increases much faster in comparison with the GTSRB dataset.

What is a little surprising is the increasing number of valid adversarial samples between epochs. More adversarial samples, that are still valid, means that less adversarial samples need to be generated every next epoch. It basically means that with each iteration the network looses a piece of its ability to adapt to the adversarial samples.

6.3.3 Summary

The Simple caching method performs faster on all tests, which is satisfactory as the slowness of the original method was the impulse for designing the caching method. The Simple caching method also trains the robust classifier somewhat slower than the original method. That is not unexpected, as intuitively a new, freshly constructed adversarial samples should have bigger impact on the model than adversarial samples that the classifier has already seen two or three times. However, the caching also shows a distinct feature that is highly unfavourable.

Epoch	0	1	2	3
Training time [hh:mm]		08:09	12:05	13:34
Validation time [minutes]		77	90	97
Classification accuracy [%]	99	99.06	99.18	99.36
Adver sample accuracy [%]		31.86	51.32	59.26
FGSM accuracy [%]		68.14	48.68	40.74
FGSM average time [ms]		414	394	403
FGSM L2 distance		6.83	6.49	6.64
DeepFool accuracy [%]		100	100	100
DeepFool average time [ms]		220	219	221
DeepFool L2 distance		2.77	2.73	2.75

Table 6.8: Summary of three epochs of adversarial training using *Fast Gradient Sign Method* as described in [PMG+17]. The initial model, displayed in column under epoch 0, is described in Chapter 5 and trained on the MNIST dataset up to 99% accuracy. The table shows time needed to train and then evaluate the model together with the model's accuracy on both the original dataset input and the created adversarial samples. Next are the statistics for FGSM algorithm (which it is being trained on) and DeepFool algorithm to represent the transferability of robustness against adversarial samples. This table is a direct comparison with Table 6.1 on different dataset, most notably the higher defense rate and computational time.

Epoch	0	1	2	3
Training time [hh:mm]		05:30	07:02	08:41
Validation time [minutes]		60	64	68
Classification accuracy [%]	99	98.56	98.82	99.05
Valid adversarial samples	54 927	4 648	5 746	6 910
Adver sample accuracy [%]		20.16	34.66	47.34
FGSM accuracy [%]		79.84	65.34	52.66
FGSM average time [ms]		374	383	390
FGSM L2 distance		6.61	7.71	8.78
DeepFool accuracy [%]		100	100	100
DeepFool average time [ms]		245	251	256
DeepFool L2 distance		2.85	2.88	2.91

Table 6.9: Summary of three epochs of simple caching method using *DeepFool algorithm* as its base algorithm. The initial model, displayed in column under epoch 0, is described in Chapter 5 and trained on the MNIST dataset up to 99% accuracy. The table shows time needed to train and then evaluate the model together with the model's accuracy on both the original dataset input and the created adversarial samples. Next are the statistics for FGSM algorithm to represent the transferability of robustness against adversarial samples and DeepFool algorithm (which it is being trained on). The "Valid adversarial samples" field shows that the cache is being invalidated every epoch.

It was observed on the GTSRB dataset that the model was completely unable to train on the cached adversarial samples despite them being misclassified in the verification process. Several tests were performed to analyze this unusual behaviour.

- Skipping the verification process and generation of new adversarial samples did not help as shown in Table 6.7.
- Partial invalidation of the cached adversarial samples did not change anything either.
- Giving all adversarial samples an uniform label did completely destroy the model as it should (the observed accuracy after first epoch was 3.28%).

The reason for this behaviour seems to be something in the very nature of caching. The results on the MNIST dataset support this theory as well – it is shown that over 90% of the cached adversarial samples had to be recreated in the middle of training (i.e., using the original method) in the first epoch and this corresponds to the gain in robustness as well as changes in speed. Unfortunately, it is unclear precisely why the caching of adversarial samples results in the model being unable to train on those samples.

Chapter 7

Conclusion

The system for making the convolutional neural network classifier more robust was successfully developed, using only open-source resources. Two appropriate datasets were selected and prepared to be worked on. A suitable model of convolutional neural network was found and adjusted for the needs of this thesis, then trained on both datasets to meet the criterion to continue on the next step. A toolbox was found to help with attacking the model and then also included inside the model to help with making the model more robust.

Two different algorithms for creating the adversarial samples, the *Fast Gradient Sign Method* and the *DeepFool attack*, were chosen to attack the system. Two different metrics, the percentage of correctly classified adversarial samples and the L2 distance between original data and adversarial sample created from that data, were introduced to evaluate the robustness of the system. Precise measurement of time for both attack algorithms as well as for whole epochs was also added in the system. An already existing method of making the classifier more robust was successfully implemented.

An issue with speed was found during the development in the already existing method of making the classifier more robust. A new method was designed to solve this problem. The solution was to add a cache to store the adversarial samples, fill the cache before initiating the training process and use this cache in the training process instead.

The problem with speed was immediately solved by using the newly created cache. The L2 distance between original and adversarial samples also showed a nice improvement but another problem appeared instead. The accuracy metric of the robustness (i.e. the percentage of correctly classified adversarial samples) went down to zero, effectively showing that the caching is not a suitable mechanism for making the classifier more robust against the adversarial samples. This was not the expected outcome but both the logic behind the decision to use the cache as well as the design of the caching method seemed to be built on solid ground.

The situation was examined and a couple of experiments were run to identify the problem or discover an error in the proposed method. All performed tests suggest that the proposed method works as designed and that the problem lies somewhere in the caching of adversarial samples. Unfortunately, the reason for this behaviour is not currently known and further research on this topic is encouraged.

To sum up, all objectives of the thesis assignment were fulfilled:

1. a review of methods for creating the adversarial samples was done with further analysis in Chapters 2 and 3.
2. a review of existing methods for making the classifier more robust was also discussed in Chapter 3 together with other possible defenses
3. an existing method was described and implemented in Chapter 6 and a novel method was designed there as well
4. both methods were compared on two classifier models constructed to be as close as possible while working on two different datasets using two different attack algorithms and three metrics: the robustness of the model in percents, the L2 distance between original data and created adversarial samples, and finally the time needed to create an adversarial as well as the time needed to make the model more robust.

Appendix A

Bibliography

[AAB+16] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*, ArXiv e-prints (2016).

[BNS+06] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. D. Tygar, *Can machine learning be secure?*, Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security (New York, NY, USA), ASIACCS '06, ACM, 2006, pp. 16–25.

[BRB17] W. Brendel, J. Rauber, and M. Bethge, *Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models*, ArXiv e-prints (2017).

[EEF+17] Ivan Evtimov, Kevin Eykholt, Earlence Fernandes, Tadayoshi Kohno, Bo Li, Atul Prakash, Amir Rahmati, and Dawn Song, *Robust physical-world attacks on machine learning models*, CoRR **abs/1707.08945** (2017).

[FL98] Drew Fudenberg and David Levine, *Learning in games*, European Economic Review **42** (1998), no. 3-5, 631–639.

[GDDM13] R. Girshick, J. Donahue, T. Darrell, and J. Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation*, ArXiv e-prints (2013).

[GLS+09] Ian J. Goodfellow, Quoc V. Le, Andrew M. Saxe, Honglak Lee, and Andrew Y. Ng, *Measuring invariances in deep networks*, Proceedings of the 22Nd International Conference on

Neural Information Processing Systems (USA), NIPS'09, Curran Associates Inc., 2009, pp. 646–654.

[GSS14] I. J. Goodfellow, J. Shlens, and C. Szegedy, *Explaining and Harnessing Adversarial Examples*, ArXiv e-prints (2014).

[KGB16] A. Kurakin, I. Goodfellow, and S. Bengio, *Adversarial examples in the physical world*, ArXiv e-prints (2016).

[LBBH98] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE **86** (1998), no. 11, 2278–2324.

[LCLS16] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song, *Delving into transferable adversarial examples and black-box attacks*, CoRR **abs/1611.02770** (2016).

[LF11] Fredrik Larsson and Michael Felsberg, *Using fourier descriptors and spatial models for traffic sign recognition*, Image Analysis (Berlin, Heidelberg) (Anders Heyden and Fredrik Kahl, eds.), Springer Berlin Heidelberg, 2011, pp. 238–249.

[MFF15] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, *DeepFool: a simple and accurate method to fool deep neural networks*, ArXiv e-prints (2015).

[MGB03] H Brendan McMahan, Geoffrey J Gordon, and Avrim Blum, *Planning in the presence of cost functions controlled by an adversary*, In Proceedings of the Twentieth International Conference on Machine Learning, 2003.

[MTBG13] M. Mathias, R. Timofte, R. Benenson, and L. Van Gool, *Traffic sign recognition - How far are we from the solution?*, The 2013 International Joint Conference on Neural Networks (IJCNN), Aug 2013, pp. 1–8.

[NP16] N. Narodytska and S. Prasad Kasiviswanathan, *Simple Black-Box Adversarial Perturbations for Deep Networks*, ArXiv e-prints (2016).

[PMG+17] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami, *Practical black-box attacks against machine learning*, Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security (New York, NY, USA), ASIA CCS '17, ACM, 2017, pp. 506–519.

[PMJ+15] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami, *The limitations of deep learning in adversarial settings*, CoRR **abs/1511.07528** (2015).

[RBB17] J. Rauber, W. Brendel, and M. Bethge, *Foolbox: A Python toolbox to benchmark the robustness of machine learning models*, ArXiv e-prints (2017).

[SBBR16] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter, *Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition*, Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (New York, NY, USA), CCS '16, ACM, 2016, pp. 1528–1540.

[SL11] P. Sermanet and Y. LeCun, *Traffic sign recognition with multi-scale convolutional networks*, The 2011 International Joint Conference on Neural Networks, July 2011, pp. 2809–2813.

[SSSI11] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel, *The German Traffic Sign Recognition Benchmark: A multi-class classification competition*, IEEE International Joint Conference on Neural Networks, 2011, pp. 1453–1460.

[ST14] Sayanan Sivaraman and Mohan M. Trivedi, *Active learning for on-road vehicle detection: a comparative study*, Machine Vision and Applications **25** (2014), no. 3, 599–611.

[Sta17] Alex Staravoitau, *Traffic signs classification with a convolutional network*, 2017.

[SZS+13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus, *Intriguing properties of neural networks*, CoRR **abs/1312.6199** (2013).

[TKP+17] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, *Ensemble Adversarial Training: Attacks and Defenses*, ArXiv e-prints (2017).

[TV15] P. Tabacof and E. Valle, *Exploring the Space of Adversarial Images*, ArXiv e-prints (2015).

[ZF13] M. D Zeiler and R. Fergus, *Visualizing and Understanding Convolutional Networks*, ArXiv e-prints (2013).

Appendix B

CD Content

In table B.1 are listed names of all root directories on CD together with a short description

Directory name	Description
ctuthesis-master	thesis sources in latex format and all the figures organized in sub-folders
sources	Python source codes organized in sub-folders "gtsrb" and "mnist" according to datasets
training-sessions	complete outputs of all training sessions during development organized in sub-folders "gtsrb" and "mnist" according to datasets

Table B.1: CD Content

Appendix C

Training outputs

This appendix contains complete output of one selected training session to display all the information that did not make it in the tables in Chapter 6.

C.1 Training session from Chapter 6.1 using Fast Gradient Sign Method as the base algorithm

```
Number of training examples = 39209
Number of testing examples = 12630
Image data shape = (32, 32, 3)
Number of classes = 43
================================================
============= RESUMING TRAINING =============
================================================
=================== DATA ====================
            Training set: 29406 examples
          Validation set: 9803 examples (now only 1000)
             Testing set: 12630 examples
              Batch size: 256
    Adversarial training: True
=================== MODEL ===================
--------------- ARCHITECTURE ----------------
             Type       Size     Dropout (keep p)
 Layer 1     5x5 Conv   32       0.9
 Layer 2     5x5 Conv   64       0.8
 Layer 3     5x5 Conv   128      0.7
 Layer 4     FC         1024     0.5
---------------- PARAMETERS -----------------
     Learning rate decay: Disabled (rate = 0.0001)
       L2 Regularization: Enabled (lambda = 0.0001)
          Early stopping: Enabled (patience = 100)
 Keep training old model: Enabled
```

```
================= TRAINING ==================
 Timestamp: 2018/04/14 14:59:40
 Training 29406 samples (+ 29406 adversarials)
 Estimated time: 101 minutes
      Timestamp: 2018/04/14 14:59:41
      Training done
      Timestamp: 2018/04/14 15:44:18
 Validating 1000 samples
 Estimated time: 2 minutes
      Timestamp: 2018/04/14 15:44:18
-------------- EPOCH    0/10 --------------
Validation loss: 0.17655453, accuracy: 96.73%
      Best loss: inf at epoch 0
    Adversarial samples statistics:
Validation loss: 1.06570339, accuracy: 2.97%

         Foolbox attack statistics:
Validation norm: 2.26526799,  success: 97.11%
     FGSM times: 5,  66.24,  1240 [ms]
  DeepFool norm: 0.89076862,  success: 100.00%
 DeepFool times: 25,  66.98,  613 [ms]
   Elapsed time: 01:11:52
      Timestamp: 2018/04/14 16:11:29



 Training 29406 samples (+ 29406 adversarials)
 Estimated time: 101 minutes
      Timestamp: 2018/04/14 16:11:30
      Training done
      Timestamp: 2018/04/14 17:08:35
 Validating 1000 samples
 Estimated time: 2 minutes
      Timestamp: 2018/04/14 17:08:35
-------------- EPOCH    1/10 --------------
Validation loss: 0.16902132, accuracy: 97.51%
      Best loss: 0.17655453 at epoch 0
    Adversarial samples statistics:
Validation loss: 1.07711542, accuracy: 3.58%

         Foolbox attack statistics:
Validation norm: 2.73840619,  success: 96.57%
     FGSM times: 5,  77.46,  1212 [ms]
  DeepFool norm: 0.96766962,  success: 100.00%
 DeepFool times: 25,  69.63,  878 [ms]
   Elapsed time: 02:39:32
```

```
      Timestamp: 2018/04/14 17:39:09



 Training 29406 samples (+ 29406 adversarials)
 Estimated time: 101 minutes
      Timestamp: 2018/04/14 17:39:10
      Training done
      Timestamp: 2018/04/14 18:46:40
 Validating 1000 samples
 Estimated time: 2 minutes
      Timestamp: 2018/04/14 18:46:40
-------------- EPOCH    2/10 --------------
Validation loss: 0.15774907, accuracy: 97.61%
      Best loss: 0.16902132 at epoch 1
    Adversarial samples statistics:
Validation loss: 1.05356121, accuracy: 5.43%

         Foolbox attack statistics:
Validation norm: 3.01517622,  success: 94.75%
     FGSM times: 5,  84.28,  1286 [ms]
  DeepFool norm: 1.01472624,  success: 100.00%
 DeepFool times: 24,  70.70,  707 [ms]
   Elapsed time: 04:22:26
      Timestamp: 2018/04/14 19:22:03



 Training 29406 samples (+ 29406 adversarials)
 Estimated time: 101 minutes
      Timestamp: 2018/04/14 19:22:04
      Training done
      Timestamp: 2018/04/14 20:41:12
 Validating 1000 samples
 Estimated time: 2 minutes
      Timestamp: 2018/04/14 20:41:12
-------------- EPOCH    3/10 --------------
Validation loss: 0.13423176, accuracy: 98.10%
      Best loss: 0.15774907 at epoch 2
    Adversarial samples statistics:
Validation loss: 1.00745368, accuracy: 7.30%

         Foolbox attack statistics:
Validation norm: 3.22618910,  success: 92.86%
     FGSM times: 5,  91.40,  1290 [ms]
  DeepFool norm: 1.02189458,  success: 100.00%
```

```
 DeepFool times: 25,  70.40,  1271 [ms]
   Elapsed time: 06:21:50
      Timestamp: 2018/04/14 21:21:27



 Training 29406 samples (+ 29406 adversarials)
 Estimated time: 101 minutes
      Timestamp: 2018/04/14 21:21:27
      Training done
      Timestamp: 2018/04/14 22:56:29
 Validating 1000 samples
 Estimated time: 2 minutes
      Timestamp: 2018/04/14 22:56:29
-------------- EPOCH    4/10 --------------
Validation loss: 0.13280101, accuracy: 98.17%
      Best loss: 0.13423176 at epoch 3
    Adversarial samples statistics:
Validation loss: 1.01087189, accuracy: 9.36%

         Foolbox attack statistics:
Validation norm: 3.36661524,  success: 90.74%
     FGSM times: 5,  94.46,  1271 [ms]
  DeepFool norm: 1.07245468,  success: 100.00%
 DeepFool times: 25,  72.58,  917 [ms]
   Elapsed time: 08:42:01
      Timestamp: 2018/04/14 23:41:38



 Training 29406 samples (+ 29406 adversarials)
 Estimated time: 101 minutes
      Timestamp: 2018/04/14 23:41:39
      Training done
      Timestamp: 2018/04/15 01:28:15
 Validating 1000 samples
 Estimated time: 2 minutes
      Timestamp: 2018/04/15 01:28:15
-------------- EPOCH    5/10 --------------
Validation loss: 0.12239983, accuracy: 98.37%
      Best loss: 0.13280101 at epoch 4
    Adversarial samples statistics:
Validation loss: 0.97993243, accuracy: 10.43%

         Foolbox attack statistics:
Validation norm: 3.29705516,  success: 89.75%
```

```
     FGSM times: 5,  92.82,  1254 [ms]
  DeepFool norm: 1.06966712,  success: 100.00%
 DeepFool times: 24,  72.50,  982 [ms]
   Elapsed time: 11:15:28
      Timestamp: 2018/04/15 02:15:05



 Training 29406 samples (+ 29406 adversarials)
 Estimated time: 101 minutes
      Timestamp: 2018/04/15 02:15:06
      Training done
      Timestamp: 2018/04/15 04:10:28
 Validating 1000 samples
 Estimated time: 2 minutes
      Timestamp: 2018/04/15 04:10:28
-------------- EPOCH    6/10 --------------
Validation loss: 0.11328512, accuracy: 98.53%
      Best loss: 0.12239983 at epoch 5
    Adversarial samples statistics:
Validation loss: 0.96255738, accuracy: 12.03%

         Foolbox attack statistics:
Validation norm: 3.27853484,  success: 88.13%
     FGSM times: 5,  90.89,  1232 [ms]
  DeepFool norm: 1.08965698,  success: 100.00%
 DeepFool times: 25,  72.56,  1006 [ms]
   Elapsed time: 14:00:39
      Timestamp: 2018/04/15 05:00:16



 Training 29406 samples (+ 29406 adversarials)
 Estimated time: 101 minutes
      Timestamp: 2018/04/15 05:00:17
      Training done
      Timestamp: 2018/04/15 07:03:08
 Validating 1000 samples
 Estimated time: 2 minutes
      Timestamp: 2018/04/15 07:03:08
-------------- EPOCH    7/10 --------------
Validation loss: 0.10221992, accuracy: 98.62%
      Best loss: 0.11328512 at epoch 6
    Adversarial samples statistics:
Validation loss: 0.93385792, accuracy: 12.84%
```

```
        Foolbox attack statistics:
Validation norm: 3.31337871,  success: 87.41%
     FGSM times: 5,  91.66,  1281 [ms]
  DeepFool norm: 1.10182841,  success: 100.00%
 DeepFool times: 25,  73.33,  1180 [ms]
   Elapsed time: 16:54:57
      Timestamp: 2018/04/15 07:54:33



 Training 29406 samples (+ 29406 adversarials)
 Estimated time: 101 minutes
      Timestamp: 2018/04/15 07:54:36
      Training done
      Timestamp: 2018/04/15 10:05:03
 Validating 1000 samples
 Estimated time: 2 minutes
      Timestamp: 2018/04/15 10:05:03
-------------- EPOCH    8/10 --------------
Validation loss: 0.09846409, accuracy: 98.48%
      Best loss: 0.10221992 at epoch 7
    Adversarial samples statistics:
Validation loss: 0.91585940, accuracy: 13.59%

        Foolbox attack statistics:
Validation norm: 3.37189140,  success: 86.58%
     FGSM times: 5,  93.38,  1283 [ms]
  DeepFool norm: 1.11833447,  success: 100.00%
 DeepFool times: 25,  73.06,  1815 [ms]
   Elapsed time: 19:58:39
      Timestamp: 2018/04/15 10:58:16



 Training 29406 samples (+ 29406 adversarials)
 Estimated time: 101 minutes
      Timestamp: 2018/04/15 10:58:17
      Training done
      Timestamp: 2018/04/15 13:16:09
 Validating 1000 samples
 Estimated time: 2 minutes
      Timestamp: 2018/04/15 13:16:09
-------------- EPOCH    9/10 --------------
Validation loss: 0.10074635, accuracy: 98.62%
      Best loss: 0.09846409 at epoch 8
    Adversarial samples statistics:
```

```
Validation loss: 0.92633694, accuracy: 14.14%

         Foolbox attack statistics:
Validation norm: 3.53540606,  success: 85.99%
     FGSM times: 5,  98.48,  1257 [ms]
  DeepFool norm: 1.13684214,  success: 100.00%
 DeepFool times: 25,  74.68,  900 [ms]
   Elapsed time: 23:11:53
      Timestamp: 2018/04/15 14:11:30
```