DEPARTMENT OF ELECTRONIC AND COMPUTER ENGINEERING

DISTRIBUTED COMPUTING SYSTEMS ENGINEERING MSC

Analysis and Optimisation of a Safety Verification Framework for Deep Neural Networks

Year of Submission: 2020

Abstract

Using deep neural networks in safety critical applications lead to the necessity to verify the safety of the neural networks. Image perturbations like scratches in the cameras lens or even bad weather conditions can lead to misclassification. For applications like self driving cars a misclassification can lead to a disastrous outcome. Several approaches were introduced to verify the safety of neural networks. This dissertation aims to implement and optimise a safety verification framework considering adversarial perturbations, since previous implementations provided good results but were not particularly performant. Since the training of deep neural networks already is time consuming, a quick safety verification would be beneficial. The safety verification framework is developed with the programming language python using the deep learning library keras with theano backend. Optimisation through high performance programming libraries for calculations as well as parallelisation of certain methods speeds up the safety check significantly. Experiments on different deep neural networks with different datasets and using a baseline implementation of the framework are used to compare and evaluate the performance of the implemented framework. Furthermore the optimised framework could be incorporated in the learning process to automatically configure the network to be safe against certain perturbations. In addition, the framework can be implemented in another programming language that allows you to better utilize the resources of a system through parallelisation.

Contents

List of Figures

Acronyms

ANN Artificial Neural Network.

DNN Deep Neural Network.

CNN Convolution Neural Network.

ReLu Rectified Linear Units.

SIFT Scale Invariant Feature Transform.

MCTS Monte Carlo Tree Search.

MNIST Modified National Institute of Standards and Technology.

CIFAR Canadian Institute For Advanced Research).

SMT Satisfiability Modulo Theories.

BLAS Basic Linear Algebra Subprograms.

MKL Math Kernel Library.

API Application Programming Interface.

CPU Central Processing Unit.

GPU Graphics Processing Unit.

CUDA Compute Unified Device Architecture.

Introduction

Artificial Neural Networks have been evolving greatly since the 1940's and are now used in many areas, such as autonomous driving. Image classification is used to detect street signs or traffic lights. But despite further developments, ANNs are still mostly unstable due to adversarial perturbations. These adversarial perturbations are minimal changes to the input image which lead to misclassification. For applications in assisted driving or autonomous cars, these misclassification can lead to disastrous outcomes, for example forcing a car of the road for detecting a sign for turning left instead of right. Therefore to ensure safety in decision making a way to verify the classification of the ANN is needed. For this purpose several approaches were proposed using Satisfiability Modulo Theory to implement a verification framework or by using SIFT and MCTS. This thesis project focuses on analysing the recent and plausible approach presented by Xiaowei Huang et al. [HKW+16]. Furthermore the framework will be implemented and optimized with parallelising several parts of the framework since this approach gets more time consuming with more complex neural networks and larger images. The following Chapter 2 sums up the project management approach. Chapter 3 gives general insight in the practical and technical background. Chapter 4 summarizes the the analysis of the approach and gives details about the design and implementation. The following chapter discusses the evaluation and the accomplished optimisation steps. The final chapter summarizes the conclusion and further work.

1.1 Motivation

Using ANNs in highly automated systems like autonomous driving poses safety risks due to the unstable nature of those with respect to adversarial perturbations. Having scratches on a cameras lens or perturbations due to snow or rain or even an active manipulation of input data by an attacker can lead to misclassification of an image. Several reports in recent years stated that systems in self-driving cars misinterpreted traffic signs and eventually lead to accidents. There are several research papers concreting the problems with regard to safety in artificial neural networks or AI in general [AOS+16] [SAMR18]. Robustness, accuracy and confidence of a network are often not sufficient to ensure safety [NYC14]. For minimising the risk of misclassification and to verify ANNs, several approaches were introduced. Most of the approaches share the same problem of being time consuming and not very scalable when working on real world deep neural networks [KBD+17] [KW17] [Ehl17]. The recent approach of Xiaowei Huang et al. [HKW+17] promises good results in finding adversarial perturbations and works in the matter of seconds when working with a few dimensions. But for more complex neural networks and larger images the computation takes much longer. Having an input image with the dimensions of 28x28x1 there are 784 dimensions to be checked in the input layer. Propagating deeper through a CNN, the dimensions are also getting bigger e.g. for the first convolutional layer having 26x26x32 dimensions. Since the training of a neural network is already quite time consuming, a safety verification without great additional overhead would be beneficial. As stated by Huang the performance and scalability of this method can be significantly improved through parallelisation, which is attempted in this project.

1.2 Aims and Objectives

The aim of the project is to analyse and evaluate an existing method for safety verification of deep neural networks and implement an optimised verification framework based on the analysis with possible improvements in scalability and performance.

The essential objective is to design an improved verification framework for deep neural networks based on the analysis. Another objective is the evaluation of the framework and comparison to previous approaches. Based on the research done throughout the project a verification framework will be developed using the programming language python and neural networks running with keras and theano backend [Ker] [The]. By parallelising several steps in the verification algorithm, a performance improvement in terms of running time is expected while maintaining the accuracy and correctness with which adversarial perturbations are detected.

Project Management

This chapter outlines the organisational topics of the thesis project. Section 2.1 describes the experimental methods to be used throughout the project. The following two sections contain the time plan of the project and determine deliverables and describe the overall expected outcomes of the project.

2.1 Experimental and Investigative Methods

The initial survey brought insight and general understanding of the problem. The further procedure is to analyse the prototypical approaches in more detail and to implement a prototype in order to have a basis for testing, evaluating and optimising the newly implemented framework. On this basis possible improvements and further optimisations regarding scalability and performance can be carried out. The resulting algorithm or framework can then be tested with various datasets and evaluation can take place. To distinguish the improvements of the algorithm a detailed comparison to other methods will be done.

2.1.1 Development

The first steps of development are designing the system and setting up the environment accordingly. When everything is set up and the data in terms of pre trained networks and datasets is prepared, the implementation commences. First tests are carried out during the end phase of implementation. During testing, the result are also validated before evaluation and optimisation can start. The final steps are evaluating the results and comparing it to approaches and methods it is based on.

2.1.2 Writing

The writing parts are commenced in parallel to the implementation. Introduction and Fundamentals can be written independently of the implementation. Writing about the implementation and evaluation are commenced after the correspondent part of the implementation is done. The writing is concluded 2-3 weeks prior to submission to review and correct.

2.2 Deliverables and Specific Outcomes

The expected outcome is the research and study of a safety verification framework for deep neural networks, as well as to determine limitations and possible optimisations of this approach and a possibly improved and optimized framework based on the research.

2.3 Time Plan

A time plan has been created to ensure efficient time management depicted in 2.1

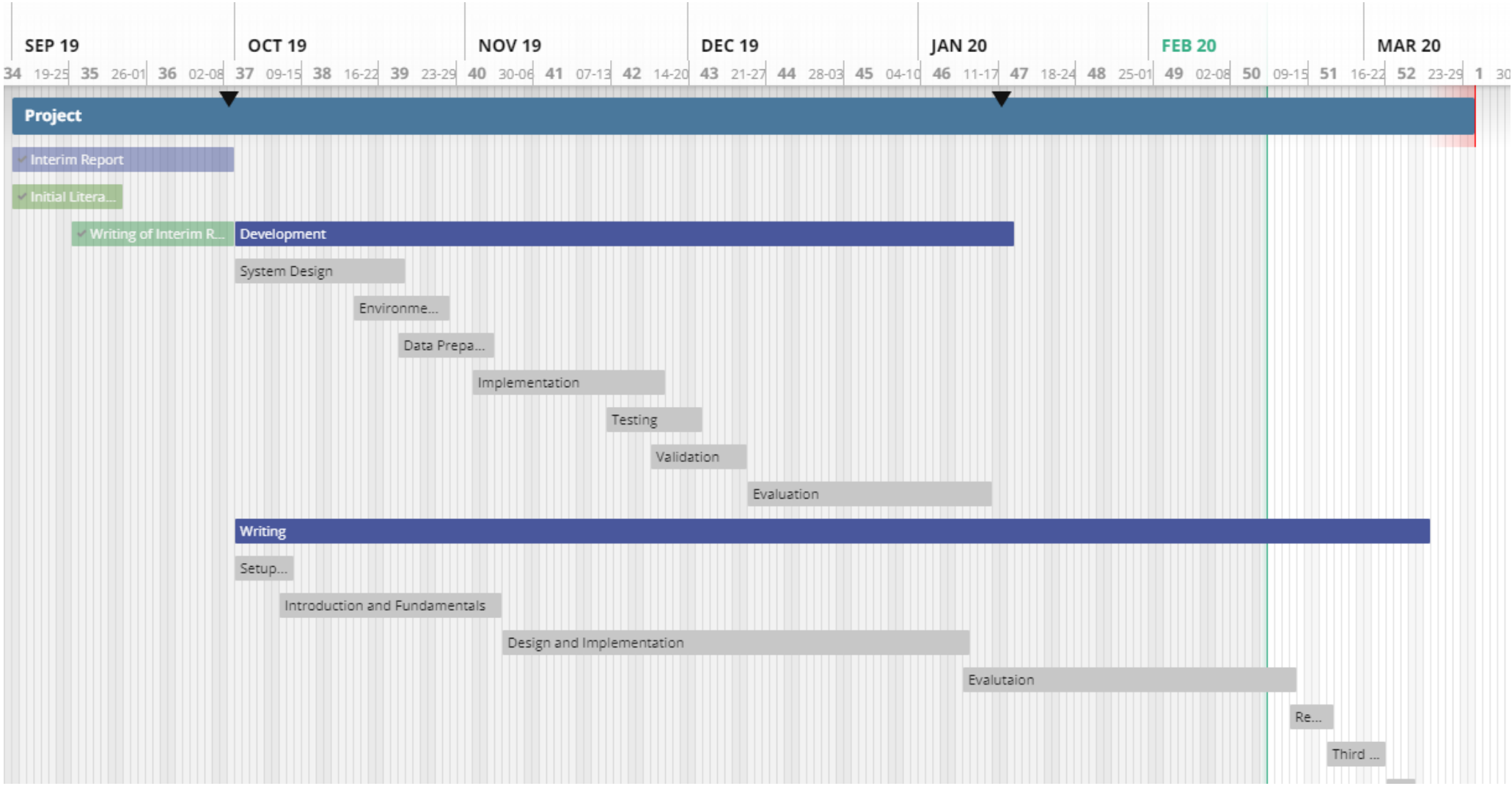

Figure 2.1: Time Plan

Literature Review

In this chapter important background information for the project is provided.

3.1 Artificial Neural Network

Artificial Neural Networks are modelled to loosely mimic the human brain [Kri07]. They consist of multiple artificial neurons which are mostly connected unidirectionally. These neurons have an activation function which is used to process the input. The connections between neurons are so called edges, which pose to have weights for learning purposes. Neurons are structured in layers. Especially in Deep Neural Networks multiple layers are used. The neurons are connected between layers. In contrary of systems using an algorithmic approach, ANNs learn by example [MW+14]. A depiction of the basic structure of an ANN can be seen in following figure 3.1.

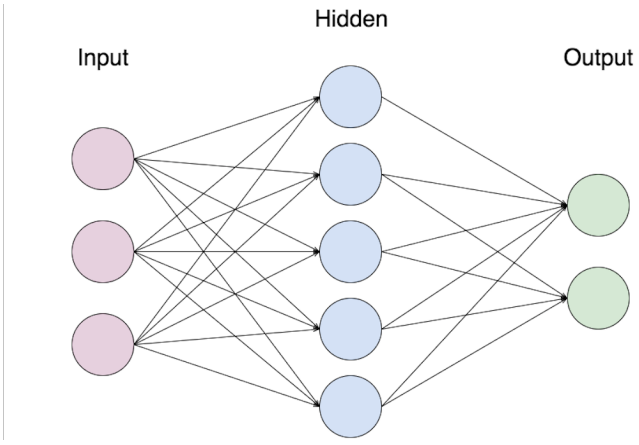

Figure 3.1: Simple Example of an ANN [Dac17]

3.1.1 Neurons

A neuron can have multiple inputs which are summed up in the first step. The sum is then processed with the activation function leading to the output of the neuron [ZHS09]. The basic structure of a neuron is depicted in figure 3.2.

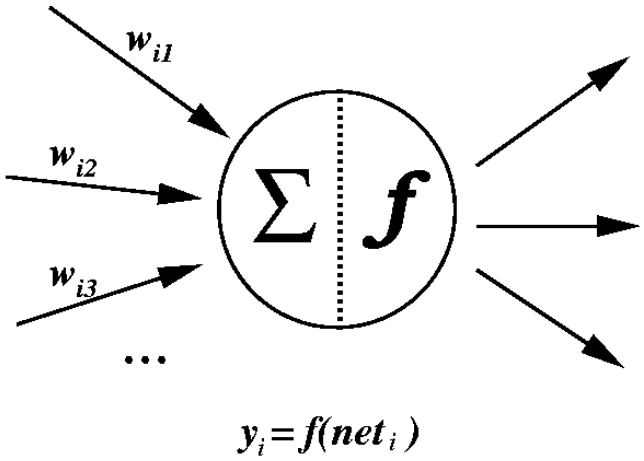

Figure 3.2: Simple Example of a Neuron [Sch14]

Common activation functions are following [AHS+14]:

- *Binary Step:* It is a threshold based function outputting a signal to the next layer when a value is above or below a certain threshold.
- *Linear:* The inputs are multiplied by a factor and therefore the output is linear to to the weighted input.
- *Sigmoid:* It is non linear and prevents jumps in the output values. The output values are bound between 0 and 1.
- *Rectified Linear Unit:* ReLu is very efficient and allows the network to converge quickly. Despite it looking like a linear function it is not and allows for back-propagation.

The activation function is selected depending on the input data and the expected outcome.

3.1.2 Types

There are several types of artificial neural networks, which are mainly distinguished by the way neurons and layers are connected with each other [SNA17]. ANNs are often split in two categories, the supervised and unsupervised. The supervised training of neural networks is done by having labelled input data. The network is trained and compares against the labels to check if its classification matches the label. These networks are mostly used for classification problems and regression problems.

Unsupervised training has unlabelled data. The network has to automatically find structure in the data by extracting features and analysing them. It is often used when labelled data is too expensive or hard to gather. Since there are no labels to compare to, it is hard to measure the accuracy of a network trained unsupervised. The most common and simplest is the feed forward network. Here the process is unidirectional from input over hidden layers to the output. Deep neural networks are basically ANNs with multiple layers. Another type is the Convolutional Neural Network, which is mainly used for image processing and image classification. A more detailed insight in convolutional neural networks is presented in chapter 3.2 since it is the main type of deep neural network used throughout this project.

3.1.3 Layers

Depending on the type of network the structure of the layers in a neural network might vary. There is usually one input and one output layer in an ANN. The number of hidden layers and how they are connected with each other often determines the type of network in interaction with the activation function used. The input layer simply passes the information to the neurons in the first hidden layer. The inputs to the layers are often weighted to enable more adjust-ability. The neurons in a hidden layer process the data and pass it to next layer. The output layer combines the results and forms the output [KB00].

3.2 Convolutional Neural Networks

The Convolutional Neural Network is a special form of the artificial neural network. It has several layers of folding, therefore also a deep neural network, and is very well suited for machine learning and applications with artificial intelligence in the field of image and speech recognition. The CNNs functionality is modeled to a certain extent on biological processes. The structure is similar to the visual cortex of the brain. The Convolutional Neural Network consists of several layers. These layers are convolution layers, subsampling layers and fully connected layers. The training of a convolutional neural network is usually supervised. One of the founders of the Convolutional Neural Network is Yann LeCun [LB98].

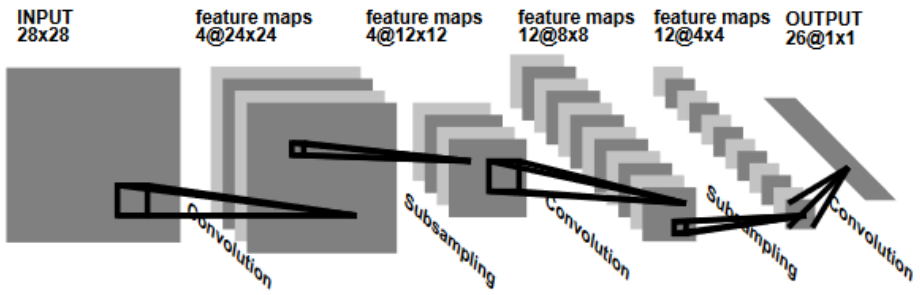

Figure 3.3: CNN for image processing[LB98]

In the convolutional layers is the actual folding takes place. It is able to recognize and extract individual characteristics in the input data. In image processing, these can be features such as lines, edges or certain shapes. The input data are processed in the form of a matrix. The pooling layer or subsampling layer, compresses and reduces the resolution of the recognized features. The layer uses methods such as max pooling or average pooling for this. Pooling discards unnecessary information and reduces the amount of data. This does not reduce the efficiency of machine learning. The calculation speed usually increases due to the reduced data volume. The Convolutional Neural Network concludes with the fully connected layer. It joins the repetitive sequences of the convolutional and pooling layers. All characteristics and elements of the previous

layers are linked to each output characteristic. The fully connected neurons can be arranged in several levels. The number of neurons depends on the classes or objects that the neural network should distinguish. The CNN uses filters to identify and extract features of the input images. The recognition of the structures is location-independent within the image. First, the CNN recognizes simple structures such as lines, points or edges in the first layers. On the following layers, the convolutional neural network learns combinations of these structures such as simple shapes or curves. With deeper layers more complex structures can be identified. The data is continuously sampled and filtered in the layers. In the last step, the results are assigned to the classes or objects to be recognized.

3.3 Data Acquisition

For applying the verification algorithm, a pre trained network is required, since training and testing a network can take a long time. Therefore several pre-trained networks for different datasets will be considered, such as MNIST Handwritten Dataset, an ImageNet Dataset or a Dataset for traffic signs. Having a well trained network diminishes the probability of having errors in the verification process due to a badly trained network.

3.4 SMT

Satisfiability modulo theories are mostly used in computer science and mathematical logic. They are decision problems for formulas with respect to combinations of background theories as classical first-order logic with equality. A solver for such theories is the Z3-Solver, which is also used in this project. The Z3-Solver was developed at Microsoft Research by Bjorner et al. [Z3g]. This solver supports for example following theories:

- Bit-Vectors
- Integer Arithmetic
- Real Arithmetic
- Integer DIfference Logic
- ...

SMT-Solvers are often used to verify programs. In this project the solver is used to solve several theories using linear real arithmetic with existential and universal quantifiers and without universal quantifiers. The strategy is called QFLRA (quantifier-free linear rational arithmetic.

3.5 Adversarial Examples

Adversarial Examples are basically inputs to a neural network which lead to an incorrect output. Several research revealed how it easy it is to fool a neural network, by manipulating the input. Even changing one pixel of an input image as described in [SVS17] can lead to a falsified classification.

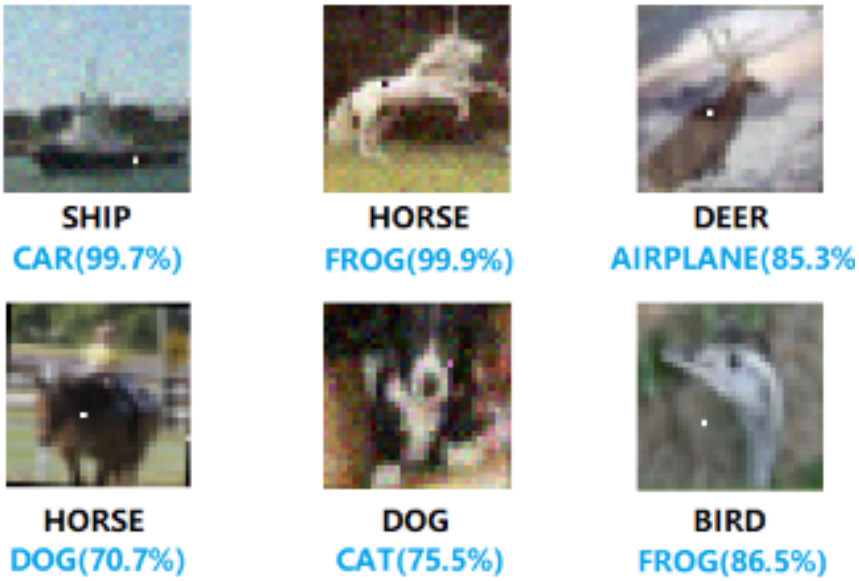

Figure 3.4: Example of One Pixel Attacks[SVS17]

The manipulation on the input image is not necessarily visible to a human observer. As described in [PMG+16] imperceptible manipulations can lead to misclassification while still looking like an unmodified image to the human eye.

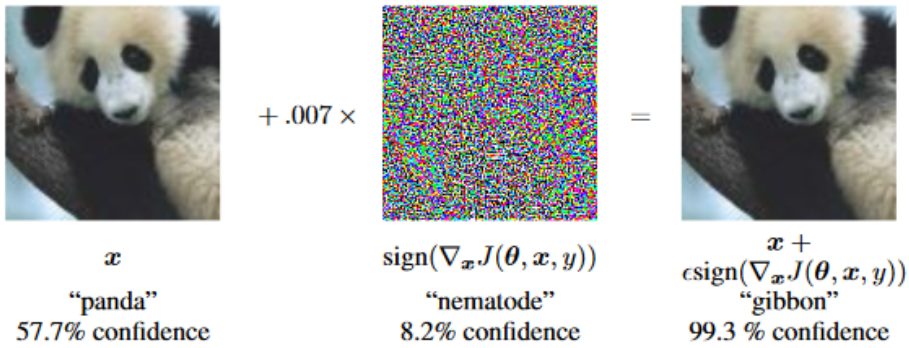

Figure 3.5: Example of adversarial example by imperceptible manipulation[GSS15]

3.6 Summary of the safety verification approach

The team around Xiaowei Huang of the University of Oxford developed an automated verification framework feed forward deep neural networks [HKW+17]. Their basic approach is to search for perturbations in each region of an image by discretization and propagate the analysis layer by layer. In this method the safety verification is reduced to a search for adversarial example and therefore safety can be achieved if no misclassification are found in all layers. Also falsification can be detected which then can be used to tune the network or to be examined by a human. The method defines a region around a given point and a set of manipulations. These manipulations have to be specified for a certain classification problem. For layer by layer analysis a mapping between the layers is needed to determine how a manipulation in one layer affects the next. A region has to be determined such that it satisfies the mapping between consecutive layers. The refine-ability of manipulations has to be defined such that a manipulation in the previous is re-finable in the current layer if there exists a sequence of manipulations in the current layer which implements the manipulation in the previous layer. Having these preconditions and definitions the verification proceeds as follows: With k being the index of the current investigated layer, N being the network, x being the input, n_k being the region and Δ_k the set of manipulations. Verify whether $N, n_k, \Delta_k \models x$, if $N, n_k, \Delta_k \models x$ then report that N is safe at x with respect to $n_k(a_{x,k})$ and Δ_k, and continue to layer $k+1$; if $N, n_k, \Delta_k \not\models x$, then report an adversarial example. This algorithm was implemented using SMT solvers. Experiments were done on several neural network classifiers for MNIST or traffic sign classification and led to found adversarial examples in seconds when working on few dimensions. But the verifying process is exponentially big regarding the number of features and therefore has a high complexity. They state that this approach is open to improvements through parallelisation regarding scalability and performance.

Design and Implementation

The proposed verification framework is mostly based on existing work by Xiaowei Huang et al [HKW+16] since it is the most recent and rather plausible approach there is. The following chapter deals with the analysis of existing approaches regarding safety verification of deep neural networks, highlighting the adopted approach and describing the architecture and implementation. This project does not consider the approach proposed in [WHK18] incorporating MCTS and SIFT for feature detection.

4.1 Adopted Approach

For the adopted approach proposed by Huang et al. several definitions have to be introduced regarding the notion of safety of classification decisions. Especially based on the concept of perturbations and manipulations of an image where a human observer would classify the image as the original. Security is considered for a classification decision and parametrised through a class of manipulations and a neighbouring region. Since the approach is basically a search for adversarial perturbations, finiteness has to be ensured. Therefore so called “ladders”which describe iterated application of successive manipulations, non deterministic branching and state the conditions for an exhaustive search.

4.1.1 Architecture and Design

4.1.1.1 Safety

This method defines a region around a point so that all points in this defined region are classified as the same by a human observer to support the classification decision. The region is described by a diameter d with respect to a pre specified norm measuring the vicinity to the point x. The definition of safety for a classification decision for this approach is based on the definition of robustness by Fawzi et al. [FFF15]. It states that for a network f is not robust at the point x if there is a point y in the region$n = z \in D_{L0} | ||z-x|| <= d$ of the input layer so that $f(x) \neq f(y)$. The point y at a minimal distance to point x represents an adversarial example. The difference to Fawzis definition of robustness to the definition of safety in this approach is that it is considered layer by layer and therefore define a region n_k as a subspace of D_k at each Layer L_k for $k \in [0...n]$. This is based on the research in [LBH15] [Mal16] that in deep neural networks the image is more and more transformed in to a representation where the single classes are distinguishable by a linear classifier the deeper the layer goes. Therefore safety for a network N at a point x is described by the robustness of the classification decision at point x against perturbations within the defined region $n_k(a_{x,k})$. The calssification decision is still based on output of the activation function in output layer L_n while the perturbation takes place in any layer L_k.

4.1.1.2 Manipulations

A manipulation in this framework is an operator which generates image perturbations such as scratches on the camera lens or bad weather conditions. Depending on the use case and the classification problem, the manipulation can be defined by the user. It is important to choose reasonable types of manipulations and distance metrics. For this reason the operator $\delta_k : D_{L_k} \rightarrow D_{L_k}$ is used over the activation functions in the vector space of layer k, as well as Manhattan(L^1) and Euclidean (L^2) norms for distance measurement between the image and the perturbation through δk based on the norms described by Fawzi, Goodfellow, Bastani, et al. [BIL+16] [FFF15] [GSS14]. If a manipulation is applied to an activation it will result in another activation with changed values of some dimensions. For this reason the manipulation is presented as a hyper-rectangle. It is defined for two activations $a_{x,k}$ and $a_{y,k}$ of a layer L_k by following:

$$rec(a_{x,k}, a_{y,k}) = X_{p \in Pk}[min(a_{x,k}(p), (a_{y,k}(p)), max(a_{x,k}(p), (a_{y,k}(p))] \tag{4.1}$$

A problem is that a region n_k may contain countless of activations and therefore this approach needs discretisation to ensure a finite exploration of the region. A valid manipulation is defined as follows. For an activation $a_{x,k}$, a set of manipulations $V(a_{x,k}) \subseteq \Delta_k$ is valid if the point $a_{x,k}$ is inside of $rec(V(a_{x,k}), a_{x,k})$. A depiction of valid manipulation in two-dimensional space is presented in figure 4.1. The activation $a_{x,k}$ is in the middle of the dashed boxes, the arrows represent manipulations and the dashed boxes are the hyper-rectangles for the respective manipulations. A notion of minimal manipulation is introduced to ensure discrete spaces. With this approach it is sufficient to look for a misclassification only at the end points. A minimal manipulation is defined as follows. A minimal manipulation δ_k on an activation $a_{x,k}$ exists if there is no other manipulations ${\delta^1}_k$ and ${\delta^2}_k$ and an activation $a_{y,k}$ with ${\delta^1}_k(a_{x,k}) \leq \delta_k(a_{x,k}), a_{y,k}) = {\delta^1}_k(a_{x,k}, \delta_k(a_{x,k}) = {\delta^2}_k(a_{x,k}$ and $a_{y,n} \neq a_{x,n}$ and $a_{y,n} \neq \delta_{k,n}(a_{x,n})$. Therefore a minimal manipulation has no finer or smaller manipulation that leads to another classification.

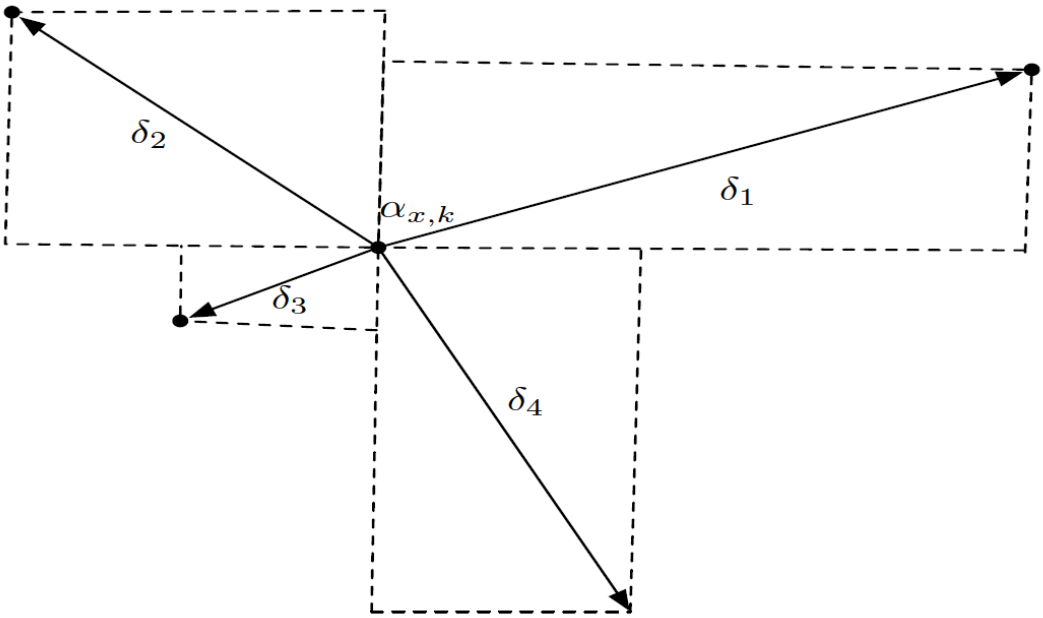

Figure 4.1: Example of a set of manipulations [HKW+16]

4.1.1.3 Bounded Variation

Since the manipulation takes place in layer L_k and the check for the classification decision in the output layer, a method to ensure an exhaustive and finite coverage of the region needs to be introduced. For this purpose the concept of bounded variation is adapted, where a continuity assumption on the mapping from space D_{L_k} to D_{L_n} is done [APF00]. With an activation $a_{x,k}$ and its respective region $n_k(a_{x,k})$ a ladder is defined, which contains a set of activations ld including $a_{x,k}$ and finitely many other activations in the associated region. The activations in a ladder are ordered in a increasing manner, to ensure that every activation in the ladder appears only once and has a successor.The last activations successor is outside the region. Following figure 4.2 illustrates ladders. As seen in the figure above, the ladder starts from $a_{x0,k}$ and each consecutive activation

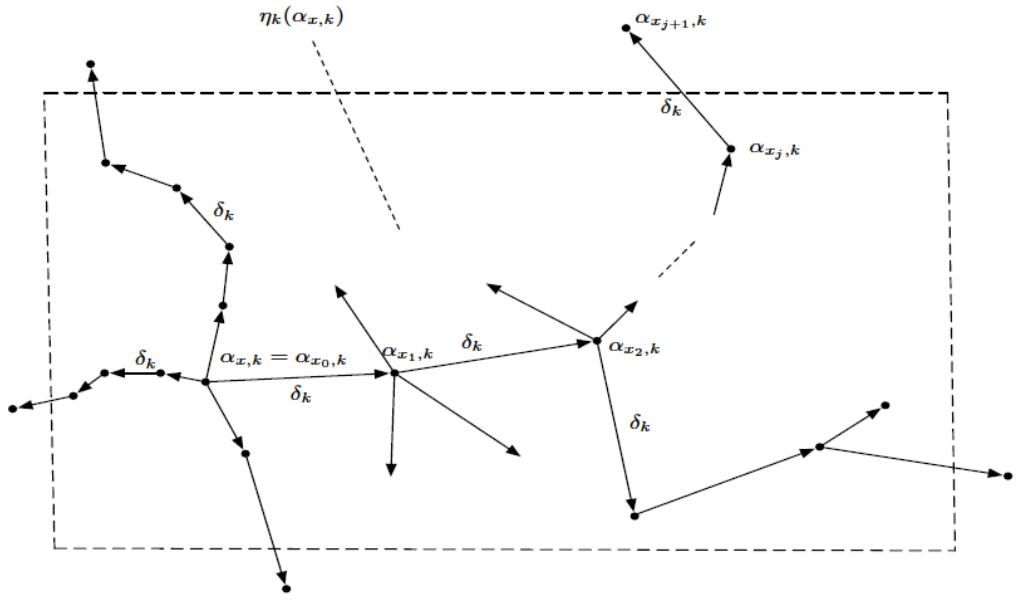

Figure 4.2: Example of ladders in a region [HKW+16]

is a result of a manipulation δ_k on a previous activation. The ladder ends with the final activation $a_{x_j,k}$ having its successor outside of the region. A ladder is complete if each node is an activation and each branch represents a manipulation.

4.1.1.4 Layer Analysis

To proceed the analysis layer by layer, refinement of the manipulations as well as mapping in both directions are needed. The mapping to the next layer is defined as $\Phi_k : D_{L_k-1} \rightarrow D_{L_k}$. The mapping to the previous layer is defined as the inverse of Φ_k and looks as follows: $\Psi_k : D_{L_k} \rightarrow D_{L_k-1}$. To propagate the safety of a region at a point x to deeper layers several assumptions have to be made. First assumption is that functions n_k that map activations to regions exist and have following restrictions on the functions mentioned before:

$$\begin{aligned} n_k(a_{x,k}) &\subseteq D_{L_k}, fork = 0,...,n \\ a_{x,k} &\in n_k(a_{x,k}), fork = 0,...,n \\ n_{k-1}(a_{i,k-1}) &\subseteq \Psi_k(n_k(a_{x,k})), forallk = 1,...,n \end{aligned} \tag{4.2}$$

The first two state that each function has a region around the activation and the third states that mapping the region n_k from the current layer to the previous layer via Ψ has to cover the region n_{k-1}. A visual representation can be seen in figure 4.3.

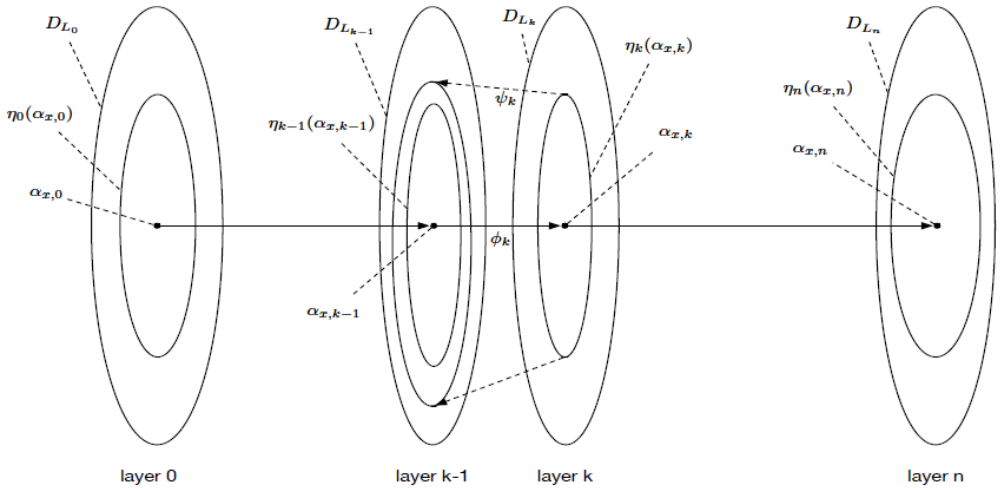

Figure 4.3: Example of layer by layer analysis [HKW+17]

With bigger size and complexity of a deep neural network it might become unsolvable with a given set of manipulations. For this reason a refinement relation between manipulations for consecutive layers is defined. Basically a manipulation in the previous layer is refinable in the current layer if there si a sequence of manipulations in the current layer that implements the manipulation in the previous layer.

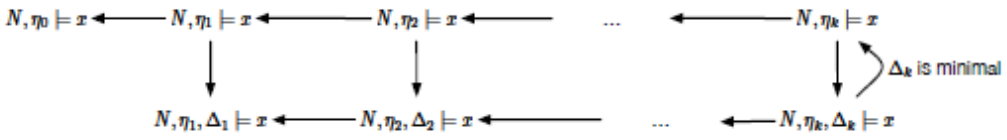

Figure 4.4: Refinement method [HKW+17]

4.1.1.5 Feature Decomposition and Discovery

Despite ensuring finiteness with methods mentioned before, a high dimensionality of a region can make the computation impractical. To lower the dimensionality of a region, features are introduced. Regions are partitioned into sets of features enabling practicality with regions having thousands of dimensions. A feature is defined for each point in the space as the most explicit salient feature. As described in [CIS+08], natural images have high-dimensional manifolds which themselves embed tangled manifolds to represent their features. These usually have lower dimensions than their parent. Classification algorithms can be used to separate them into sets of tangled manifolds and therefore leads to the assumption that these features are independent. So these can be regarded independently and pose an opportunity to reduce the problem size. An activation can include multiple features, but there is a general relation between these. So a feature can be defined on each activation as a single region. This assumption enables the possibility of feature exploration in parallel. So a feature $f_k(a_{y,k})$ si defined by a specified number of dimensions $dims_{k,f}$. The region $n_k(a_{x,k})$ is partitioned into a set of features. For such a partition $p(n_k(a_{x,k}))$ a function, containing one point for each feature, is defined as following:

$$acts(x,k) = \{a_{x,k} \in x | x \in p(n_k(a_{x,k}))\} \tag{4.3}$$

The check is carried out by checking if the points in the function have the same class as $a_{x,k}$ checking the 0-variation of all features in the partition.

4.1.1.6 Regions and Manipulations

Selecting the regions to be considered in the framework can be automated. The region $(n_k(a_{x,k})$ is defined by a subset of $dims_k$ from P_k where the activation values are furthest away from the average value of the layer. The information of these activations is more explicit than of other dimensions. Manipulations over this explicit information is more likely to result in a class change. Having the average activation value of layer L_k as $avg_k = (\sum_{p \in P_k} a_{x,k}(p))/n_k$ and $dims_k(n_k(a_{x,k})$ as the first dimensions $p \in P_k$ with greatest values $|a_{x,k}(p) - avg|$ among all dimensions, following can be defined:

$$(n_k(a_{x,k}) = \times_{p \in dims_k(n_k(a_{x,k})} [a_{x,k}(p) - s_p * m_p, a_{x,k}(p) + s_p * m_p] \tag{4.4}$$

where s_p is a small span and m_p is the number of spans for $dims_k$ containing the activation $a_{x,k}$. For an activation in the region $n_k(a_{x,k})$ a manipulation can be defined as:

$$\delta_k^d(a_{x,k})(p) = \begin{cases} a_{y,k}(p) - s_p & \text{if } d(p) = -1 \\ a_{y,k}(p) & \text{if } d(p) = 0 \\ a_{y,k}(p) + s_p & \text{if } d(p) = +1 \end{cases} \tag{4.5}$$

with d as a function mapping from $dims_k(n_k(a_{x,k})$ to $\{-1, 0, +1\}$,
so that $d(p) \neq 0 | p \in dims_k(n_k(a_{x,k})) \neq 0$ and $D(dims_k(n_k(a_{x,k}))$ as the set of such functions. So each manipulation affects a subset of the dimensions by the span s_p with the directions given in d. Δ_k poses as the set of such manipulations. Having these definitions a complete and covering set of ladders $L(n_k(a_{x,k}))$ can be defined.
The region n_k can be automatically determined if the region of previous layer n_{k-1} and the functions Φ_k and Ψ_k are given with satisfying the restrictions from equation 4.2. The activation value $a_{x,k}(p)$ of $p \in P_k$ is computed from the activation values of a subset in P_{k-1} according to function Φ_k with $Vars(p) \subseteq P_{k-1}$ as such a set of perceptrons. Following formula can be formed from the restrictions mentioned before:

$$\forall a_{y,k-1} \in n_k(a_{x,k-1}) : a_{y,k-1} \in \Psi_k(n_k(a_{x,k})) \tag{4.6}$$

The values of the variables V_k gained from satisfiability of equation 4.6 represent the definition of manipulations using equation 4.5. But since the resulting values of s_p do not necessarily satisfy the "refinement by layer" relation there is a need to adapt the values for V_k as well as retain the region $n_k(a_{x,k})$. The "refinement by layer" relation described in chapter 4.1.1.4 could be rewritten into a formula which can be solved by a SMT solver, but may result in a huge amount of comutations needed to verify the relation. To reduce computational cost, a new notion is defined including a precision ε. Therefore, a manipulation $\delta_{k-1}(a_{y,k-1})$ is refinable wit a precision $\varepsilon > 0$ if there is a sequence of activations and valid manipulations such that $a_{y,k} = a_{x_0,k}, \delta_{k-1,k}(a_{y,k-1}) \in rec(a_{x_{j-1},k}, a_{x_j,k}), dist(a_{x_{j-1},k}, a_{x_j,k}) \leq \varepsilon$, and $a_{x_t,k} = \delta_k^t(a_{x_{t-1},k})$ for $1 \leq t \leq j$. For a neural network N and an input x, the set of manipulations Δ_k are a refinement by layer of $n_k, n_{k-1}, \Delta_{k-1}$ with the precision ε, if for all activations in the region all its manipulations are refinable in layer L_k with precision ε.To find suitable values for V_k teh variable h is introduced which represents the maximal number of manipulations in layer L_k used to express a manipulation in the previous layer. Having these definitions following formula can be formed which then can be used with a SMT solver:

$$\begin{aligned} &\forall a_{y,k-1} \in n_k(a_{x,k-1}) \forall d \in D(dims_k(n_k(a_{x,k-1}) \forall \delta_{k-1}^d \in V_{k-1}(a_{y,k-1}) \\ &\exists a_{y_0,k}, ..., a_{y_h,k} \in n_k(a_{x,k}) : a_{y_0,k} = a_{y,k} \wedge \bigwedge_{t=0}^{h-1} a_{y_{t+1},k} = \delta_k^d(a_{y_t,k}) \wedge \\ &\bigvee_{t=0}^{h-1} (\delta_{k-1,k}^d(a_{y,k} \in rec(a_{y_t,k}, a_{y_{t+1},k}) \wedge dist(a_{y_t,k}, a_{y_{t+1},k}) \leq \varepsilon) \end{aligned} \tag{4.7}$$

4.1.1.7 Mapping to Input Layer

If a region in a hidden layer is manipulated, it is necessary to have a mapping back to the input layer to generate an input image which resulted in the misclassification due to the introduced manipulations in the hidden layer. Checking the 0-variation of region $n_k(a_{x,k})$ involves the computation of $diff_n(a_{x,n}, a_{y,n}$ for many points in the region. Since $a_{x,n}$ is already known, only $a_{y,n}$ needs to be calculated. It is done by looking for a point in the input layer which is in the computed mapping of $Pre_0(a_{y,k})$. Since all points in the mapping have the same class, any found point with this constraint is adequate. For this purpose the functions $\Psi_k, \Psi_{k-1}, ..., \Psi_1$ are used to compute $a_{y,0}$, so that $a_{y,j-1} = \Psi_j(a_{y,j}) \wedge a_{y,j-1} \in N_{j-1}(a_{x,j-1})$ for $1 \leq j \leq k$. For this computation a SMT solver is required to encode the functions $\Psi_k, \Psi_{k-1}, ..., \Psi_1$ if they are piecewise linear or taking the inverse functions if they are sigmoid functions. It is possible that the SMT solver cannot find any points for these constraints, which means that the regarded point has no corresponding point in the input layer.

4.1.2 Implementation

4.1.2.1 Framework

The safety verification tool was implemented as proposed in [HKW+16] using the programming language python, which provides a huge variety of libraries for artificial neural networks. Keras is used as the central deep learning library for this application [Ker]. It is a high-level neural networks API running with TensorFlow, CNTK or Theano. For this application the Theano backend is used [The]. Keras enables easy and fast prototyping through modularity and extensibility. The used SMT solver is Z3. The frameworks workflow is depicted in figure 4.5.

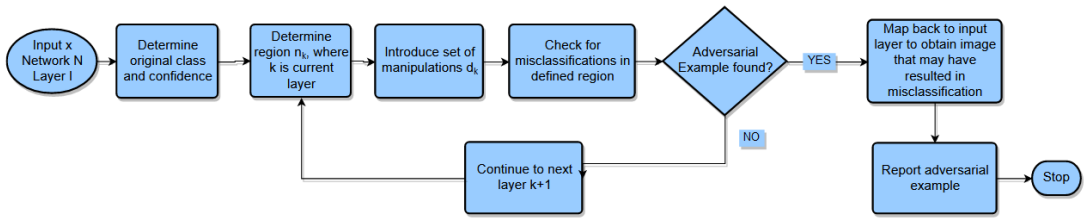

Figure 4.5: Flowchart Verification Framework

4.1.2.2 Neural Networks

For experimentation several pre trained neural networks were used [Mni] [Cif]. The following figures depict the convolutional neural networks used for testing and evaluation.

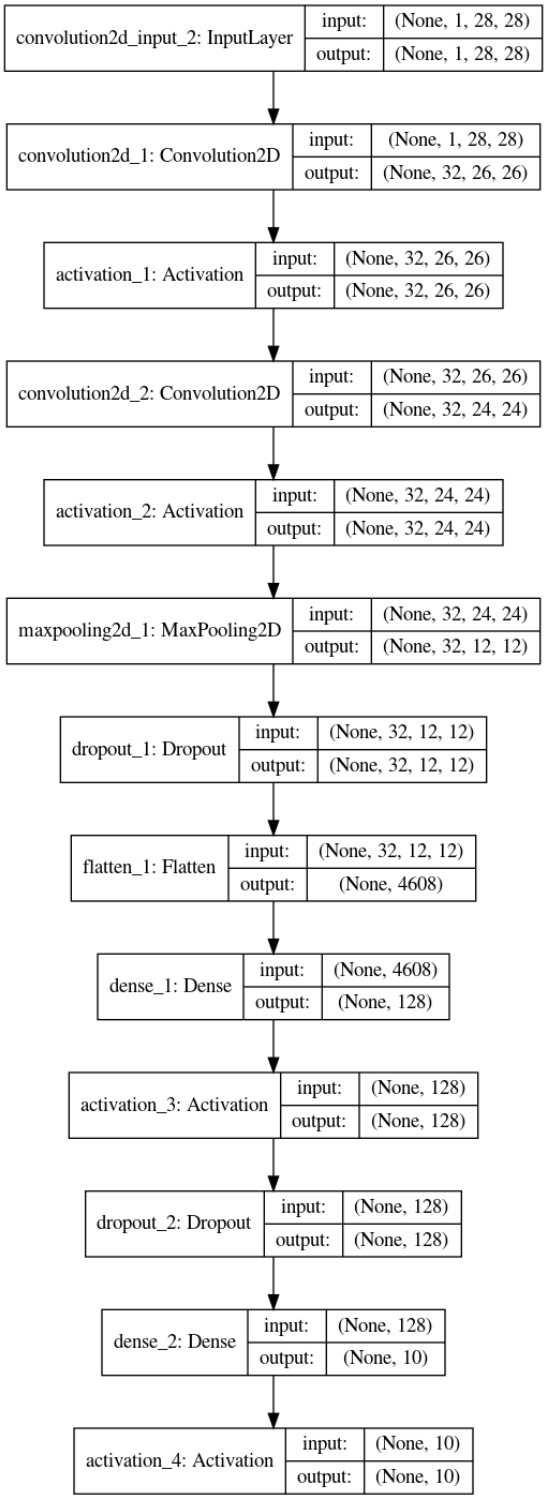

Figure 4.6: CNN Architecture for MNIST

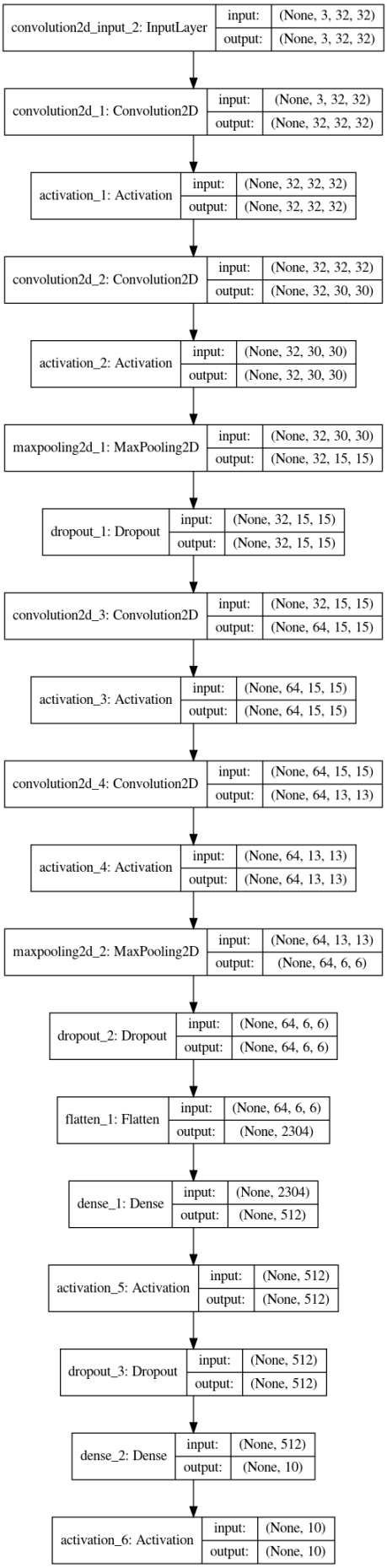

Figure 4.7: CNN Architecture for CIFAR-10

Evaluation and Optimisation

The following chapter discusses the performance metrics that were used in evaluating the framework as well as configurations used. Furthermore the applied optimization steps are described in detail and the results are discussed.

5.1 Performance Metrics

The framework generates a report when finishing its computations. This report includes several results which can be used to evaluate the run of the safety verification. The following metrics can be summarized from the report:

Running time : Describes the time the framework was running until an adversarial example was found.

Manipulation Percentage : Describes the amount of manipulation to the image which lead to the misclassification.

Euclidean distance : Euclidean distance to the adversarial perturbation found from the original image

L1 distance : L1 distance to the adversarial perturbation found from the original image

Confidence :Confidence of the classification decision

Additional average statistics are generated if multiple runs are done.

5.2 Configuration

Additionally to an image and a neural network, the framework has several configuration parameters. The following chapter describes important parameters and highlights the configurations used for the performed experiments.

5.2.1 Framework Parameters

Name	Type	Range	Description
l	Integer	[0,n]	indicates starting layer L_l
$dims_l$	Integer	≥ 1	maximum number of dimension to be considered in Layer L_l
ε	Integer	$[0, \text{inf})$	error bound for manipulation refinement between layers
$dims_{k,f}$	Integer	≥ 1	number of dimensions for each feature

Further settings are:

dataset

Which dataset to work on (mnist,cifar10,imageNet...)

dataProcessing

To work on single image or a batch

checkingMode

specific layer or stepwise

exitCondition

exit when first example found, or when all are found

controllDistance

which norm to use for distance to perturbation

5.2.2 Experiment Configuration

For experiments with the neural networks described earlier, following configurations were used:

Table 5.1: Configuration for MNIST CNN

Parameter	Value
l	1
$dims_l$	150
ε	1.0
$dims_{k,f}$	5

Table 5.2: Configuration for CIFAR-10 CNN

Parameter	Value
l	1
$dims_l$	500
ε	1.0
$dims_{k,f}$	5

Table 5.3: Configuration for ImageNet CNN

Parameter	Value
l	2
$dims_l$	20000
ε	1.0
$dims_{k,f}$	5

5.3 Preliminary Results

After implementing a first stable build of the safety verification framework several experiments were carried out to check the functionality and performance, to identify possible issues and find possibilities for optimisation. The experiments were conducted on a system with a 4-Core Ryzen 5 1500X processor at 3.5 GHz and 16 GB memory.

5.3.1 Runs on MNIST Dataset

For this experiment the pre trained neural network described in figure 4.6 was used with the basic configuration from table 5.1. The first several runs took up to 5 minutes, which is rather long considering that the neural network is not very big or complex as well as working on only one rather small image as input.

Report on MNIST run

running time: (4, -1):309.632999897

manipulation percentage: (4, -1):0.00892857142857

Euclidean distance: (4, -1):2.64575131106

L1 distance: (4, -1):7.0

confidence: (4, -1):0.55623925

average running time: 309.632999897

average manipulation percentage: 0.00892857142857

average euclidean distance: 2.64575131106

average L1 distance: 7.0

average L0 distance: 7

success rate: 1.0

The report as well as the following few figures are showing similar results as described in [HKW+16].

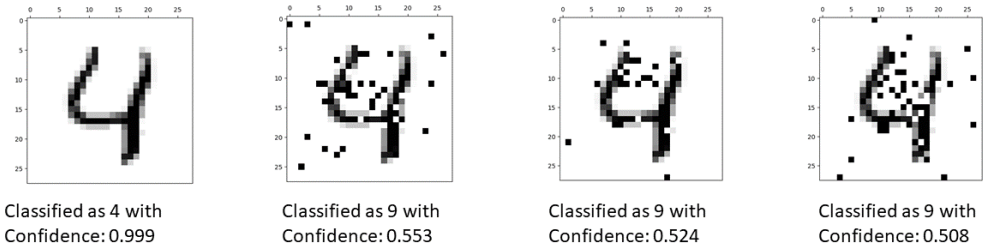

Figure 5.1: Misclassification Examples of handwritten 4

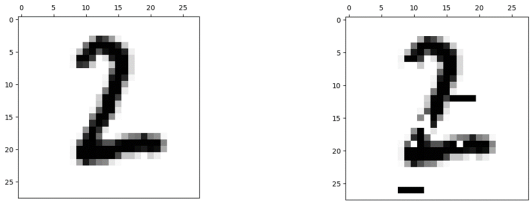

Figure 5.2: 2 misclassified as 3 with confidence: 0.669

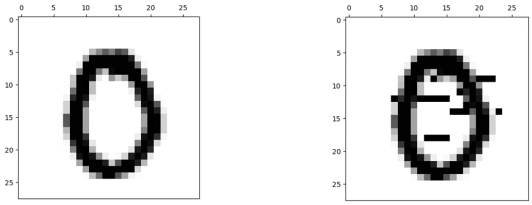

Figure 5.3: 0 misclassified as 8 with confidence: 0.604

But using this framework to ensure safety for a deep neural network, there is a need to check for multiple adversarial examples and for multiple input images as well as more complex neural networks and larger input data and therefore the computation time might rise drastically.

The following two figures summarize an example logging output of the framework.

```
C:\Users\Eugen\Documents\SVF\SVF33\Scripts\python.exe C:/Users/Ich/Documents/SVF/SVF.py
Using Theano backend.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
Start loading model ...

____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
convolution2d_1 (Convolution2D)  (None, 32, 26, 26)    320         convolution2d_input_1[0][0]
____________________________________________________________________________________________________
activation_1 (Activation)        (None, 32, 26, 26)    0           convolution2d_1[0][0]
____________________________________________________________________________________________________
convolution2d_2 (Convolution2D)  (None, 32, 24, 24)    9248        activation_1[0][0]
____________________________________________________________________________________________________
activation_2 (Activation)        (None, 32, 24, 24)    0           convolution2d_2[0][0]
____________________________________________________________________________________________________
maxpooling2d_1 (MaxPooling2D)    (None, 32, 12, 12)    0           activation_2[0][0]
____________________________________________________________________________________________________
dropout_1 (Dropout)              (None, 32, 12, 12)    0           maxpooling2d_1[0][0]
____________________________________________________________________________________________________
flatten_1 (Flatten)              (None, 4608)          0           dropout_1[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 128)           589952      flatten_1[0][0]
____________________________________________________________________________________________________
activation_3 (Activation)        (None, 128)           0           dense_1[0][0]
____________________________________________________________________________________________________
dropout_2 (Dropout)              (None, 128)           0           activation_3[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 10)            1290        dropout_2[0][0]
____________________________________________________________________________________________________
activation_4 (Activation)        (None, 10)            0           dense_2[0][0]
====================================================================================================
Total params: 600,810
Trainable params: 600,810
Non-trainable params: 0
____________________________________________________________________________________________________
Model loaded!

processing input of index 4 in the dataset:
the shape of the input is (28L, 28L)
directly handling the image ...
13 keypoints are found.
The pixels are partitioned with respect to keypoints.
160 manipulations have been initialised for keypoint (8,16), whose response is 0.0841175988317.
79 manipulations have been initialised for keypoint (8,7), whose response is 0.0409617908299.
28 manipulations have been initialised for keypoint (10,14), whose response is 0.0679738298059.
90 manipulations have been initialised for keypoint (10,7), whose response is 0.0901470184326.
16 manipulations have been initialised for keypoint (11,11), whose response is 0.0413023792207.
41 manipulations have been initialised for keypoint (14,19), whose response is 0.0724877491593.
26 manipulations have been initialised for keypoint (15,14), whose response is 0.0626098662615.
106 manipulations have been initialised for keypoint (17,9), whose response is 0.0454806014895.
127 manipulations have been initialised for keypoint (17,22), whose response is 0.107759676874.
10 manipulations have been initialised for keypoint (17,9), whose response is 0.0456902161241.
28 manipulations have been initialised for keypoint (19,17), whose response is 0.0524917431176.
73 manipulations have been initialised for keypoint (21,13), whose response is 0.0467853844166.
the number of all manipulations initialised: 784
```

Figure 5.4: Example run on MNIST Dataset

```
the number of moves we have made up to now: 0
euclidean distance 0.0
L1 distance 0.0
L0 distance 0
manipulated percentage distance 0.0
manipulated dimensions []
making a move into the new root 1, whose value is 0.0 and visited number is 1
confidence: 0.99993354
the number of moves we have made up to now: 1
euclidean distance 0.0
L1 distance 0.0
L0 distance 0
manipulated percentage distance 0.0
manipulated dimensions []
making a move into the new root 37, whose value is 37.443138122558594 and visited number is 27
confidence: 0.99993354
the number of moves we have made up to now: 2
euclidean distance 0.0
L1 distance 0.0
L0 distance 0
manipulated percentage distance 0.0
manipulated dimensions []
making a move into the new root 53, whose value is 37.443138122558594 and visited number is 14
confidence: 0.9999519
the number of moves we have made up to now: 3
euclidean distance 0.643137285618
L1 distance 0.6431373
L0 distance 1
manipulated percentage distance 0.00127551020408
manipulated dimensions [(16, 15)]
making a move into the new root 70, whose value is 103.2431411743164 and visited number is 161
confidence: 0.9999519

found an adversary image within prespecified bounded computational resource. The following is its information:
difference between images: [(6, 12), (6, 14), (9, 13), (9, 14), (11, 16), (12, 14), (24, 5)]
number of adversarial examples found: 4
euclidean distance 2.64575131106
L1 distance 7.0
L0 distance 7
manipulated percentage distance 0.00892857142857
class is changed into 9 with confidence 0.55623925

Please refer to the file data/mnist_statistics/dataCollection.txt for statistics.
--- 313.435000181 seconds ---

Process finished with exit code 0
```

Figure 5.5: Example run on MNIST Dataset continued

5.3.2 Runs on CIFAR-10 Dataset

For this experiment the pre trained neural network described in figure 4.7 was used with the basic configuration from table 5.2. Following report shows the results for finding the misclassification of a ship as an automobile. Despite having a bigger neural network as well as bigger input image size, the framework took less time to determine the misclassification due to first improvements using the OpenBLAS library for computation described in section 5.4.

Report on CIFAR-10 run
running time: (1, -1):186.572000027
manipulation percentage: (1, -1):0.0009765625
Euclidean distance: (1, -1):1.7297896523
L1 distance: (1, -1):2.9960785
confidence: (1, -1):0.59953994
average running time: 186.572000027
average manipulation percentage: 0.0009765625
average euclidean distance: 1.7297896523
average L1 distance: 2.9960784912109375
average L0 distance: 3
success rate: 1.0

The figures 5.6 to 5.8 show some examples for detected misclassification with the respective manipulated input images.

One observation during some runs was that the deeper the layer, the longer the framework takes to find a misclassification, which confirms Huang's statement that the more and bigger dimensions need to be checked, the longer the framework needs. This concluded that an parallelisation of the check within a layer is needed as a first step, rather than an overall optimisation of the framework. The parallelisation attempt is described in chapter 5.4.2.

Figure 5.6: Misclassification of boat with higher confidence than original classification

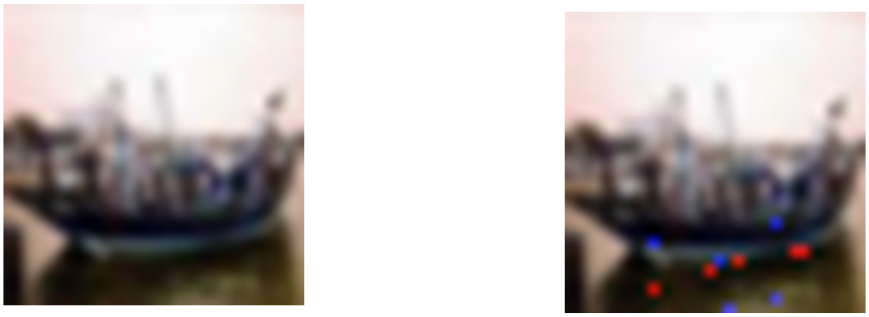

Figure 5.7: Ship to Plane with conf.: 0.887

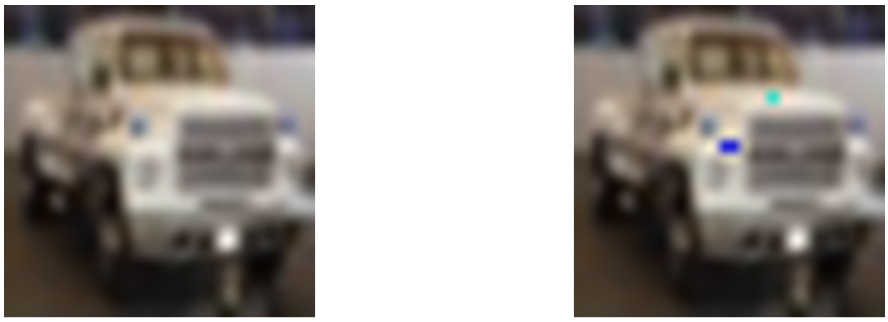

Figure 5.8: Truck to Car with conf.: 0.796

5.4 Optimisation

5.4.1 BLAS Library

The first issue found was that the theano backend was using the Numpy C-API based implementation for BLAS functions as seen in in the logs in figure 5.4. This explained the slow computation in the first experiment with the MNIST Dataset. Some research and experiments have shown that using a different library for computations should improve the performance significantly. Therefore attempts where made to set up a new environment using MKL library by Intel, since it showed promising results in several benchmarks [Beu].

Default BLAS & LAPACK

```
Dotted two 4096x4096 matrices in 64.22 s.
Dotted two vectors of length 524288 in 0.80 ms.
SVD of a 2048x1024 matrix in 10.31 s.
Cholesky decomposition of a 2048x2048 matrix in 6.74 s.
Eigendecomposition of a 2048x2048 matrix in 53.77 s.
```

ATLAS

```
Dotted two 4096x4096 matrices in 3.46 s.
Dotted two vectors of length 524288 in 0.73 ms.
SVD of a 2048x1024 matrix in 2.02 s.
Cholesky decomposition of a 2048x2048 matrix in 0.51 s.
Eigendecomposition of a 2048x2048 matrix in 29.90 s.
```

Intel MKL

```
Dotted two 4096x4096 matrices in 2.44 s.
Dotted two vectors of length 524288 in 0.75 ms.
SVD of a 2048x1024 matrix in 1.34 s.
Cholesky decomposition of a 2048x2048 matrix in 0.40 s.
Eigendecomposition of a 2048x2048 matrix in 10.07 s.
```

OpenBLAS

```
Dotted two 4096x4096 matrices in 3.97 s.
Dotted two vectors of length 524288 in 0.74 ms.
SVD of a 2048x1024 matrix in 1.96 s.
Cholesky decomposition of a 2048x2048 matrix in 0.46 s.
Eigendecomposition of a 2048x2048 matrix in 32.95 s.
```

Figure 5.9: Benchmarks of different BLAS Libraries [Beu]

But using MKL with numpy lead to only minimal performance improvements. Further research showed that MKL does not run well on AMD processors like the Ryzen 5 1500X the experiments were carried out on [Kin]. For this reason the usage of OpenBLAS was considered, which showed improvements in the experiments.

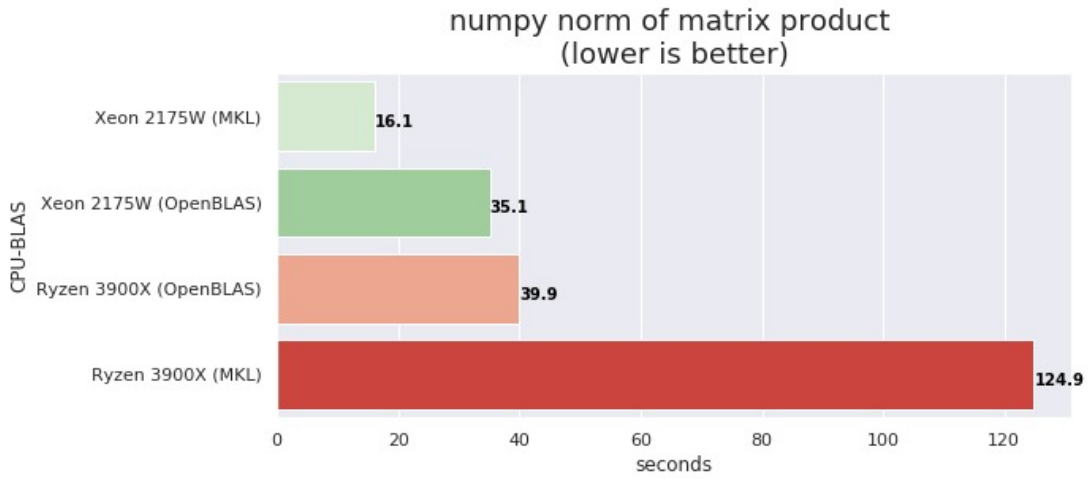

Figure 5.10: Comparison of MKL vs OpenBLAS on Intel and AMD processors [Kin]

After setting up the environment using the new library for computation, further optimisation attempts were done regarding the safety check. Using the framework to find all adversarial examples it will propagate deeper through the layers. When using the the CNN for the MNIST dataset a safety check for 32 regions in the first convolution layer takes in average 2 seconds, while checking 32 regions in the second convolutional layer might take up to 16 seconds. The reason for this is that for the first convolutional layer one input is considered. Propagating to the next convolutional layer the framework needs to check 32 inputs for each region. Therefore an attempt to parallelize the safety check was done. Since regions contain one feature as defined in chapter 4.1.1.5, they can be checked independently and therefore in parallel.

5.4.2 Multithreading vs Multiprocessing

Usually the go to approach regarding parallelism in programming is multithreading. But in python it does not suffice in optimizing the problem described earlier. Multithreading in python cannot be used for parallel CPU computation but rather for I/O operations. Python was not designed considering computers having multiple cores. The global interpreter lock (GIL) ensures that only one thread runs at a time. So multithreading in python does not promise parallel execution. Here the multiprocessing library comes in handy, which is as easy to use as multithreading, but works with actual processes. Since every process runs in its own context and cannot affect another process, a strict separation of the logic and data to work on is necessary. Due to the relations between layers, it is complex to parallelise on layer basis, since information has to be shared by multiple processes. But parallelisation can be achieved within a layer. Since the algorithm is a heuristic search, it iterates through the image points and checks the regions around for manipulations resulting in misclassification. Here the safety check can be done for each point individually. The safety check for each layer is done in a while loop with a counter of rounds limited by the configuration parameter $dims_l$ described in chapter 5.2. This loop runs until an adversarial example was found or the exit condition is reached, then the algorithm proceeds to the next layer. An example round is summarized like follows:

One round of safety check

==========

start checking the safety of layer 0

==========

Round 1 of layer 0 for image 4

(1) get a manipulated input ...

current layer: 0.

current index: (0, -1).

current operated image is saved into data/mnist_pic/temp.png

(2) synthesise region from [(10, 20), (10, 19), (11, 19), (15, 18), (15, 17)]...

convolutional layer, synthesising region ...

found the region to work

(3) safety analysis ...

the right class is 4

the confidence is 0.9999335

safety analysis for layer 0 ...

32 regions need to be checked.

ran througn the neural network for 32 times.

(4) no adversarial example is found in this round.

euclidean distance 1.06090785394

L1 distance 1.3775647

L0 distance 5

manipulated percentage distance 0.00637755102041

==========

...

Round 33 of layer 0 for image 4

...

The algorithm had to be refactored to enable computation in separate processes. For this reason the initialisation of the search tree, single rounds and respective regions was extracted. Following code snippets summarizes the parallel algorithm:

Listing 5.1: Check Safety for Layer Method

```
1  ...
2  from multiprocessing import Pool
3  ...
4  def checkSafetyForLayer
5    searchTree = SearchTree(image,layerIndex)
6    pool = Pool()
7    for(i in range(numFeatures))
8      round = InitializeNewRound(i,...)
9      pool.apply_async(processOneRound,args=[round],callback
         ↪ = checkResult)
10   ...
11   pool.join()
12   ...
```

The method in the code listing above is executed for each layer in a loop. The method initializes a search tree and a process pool. In a for loop the method calls with respective input parameters are transmitted to the process pool. To ensure a non blocking behaviour this is done asynchronously. Additionally to the input parameter, a callback function is passed to the pool, to ensure an exit condition. As soon as one process is done checking a round, the callback function is called checking the result if an adversarial example was found. If so, a report is generated and the process pool is terminated. Then the next layer can be checked. The checkResult method can be seen in code snippet 5.2.

Listing 5.2: Check Result Method for exit condition

```
1 def checkResult(result)
2   if result == result.found:
3   generateReport(result)
4   pool.terminate()
```

The method to process the safety check on one round, is basically the same as described earlier in 5.4.2

Listing 5.3: Actual method to check safety for one round

```
1 def processOneRound(round)
2   ...
3   result = checkSafety()
4   return result
```

The overall results of the optimisation and a comparison to the initial implementation are further discussed in chapter 5.5

5.4.3 Parallel SMT Solver

Further research showed that the used SMT solver Z3 already supports a functionality to solve in parallel [BMN+]. On the official github page of Z3 an example showed a similar approach to parallelisation as described in 5.4.2 using a process pool for each solver calculation [Z3g]. The Z3 solver can be parametrised with the *set_param()* method with following:

parallel.enable

Is a boolean which activates the parallel mode of the solver.

parallel.threads.max

Caps the maximal number of threads for parallel solving.

Experimenting with these parameters lead to the assumption that the used tactics and solvers are not compatible with the internal implementation for parallel solving in Z3, since activating this option led to worse performance overall.

5.4.4 Running on GPU

Since Keras supports the usage of GPU, an attempt was made to carry out the calculations on a GPU so that it could later be linked to the learning process. Research and some experiments lead to the conclusion that enabling the safety verification process to run on GPU would be very time consuming and would go beyond the time frame of the project. Nevertheless a combination with training on GPU and safety verification on CPU is possible and does not generate a lot of overhead.

5.5 Discussion

This chapter discusses the optimisation results. The following tables and respective diagram depict the achieved speed-up through the optimisation steps described in chapter 5.4 for the convolutional neural network for the CIFAR-10 dataset. Since finding an adversarial example in the first convolutional layer took already only a few seconds pre optimisation, the experiments were carried out for the second convolutional layer. Here an improvement would be more clearly visible. The experiments with the CIFAR-10 dataset were done after already using the OpenBLAS library and therefore represent the speed up only through parallelising as described in chapter 5.4.2.
Table 5.4 summarizes several runs on one image with same manipulations checking for adversarial perturbations in the second convolutional layer.

Table 5.4: Initial results on CIFAR-10 checking second convolutional layer

Running Time (s)	Manipulation (%)	Euclidean Distance	L1 Distance	Confidence
186.572000027	0.0009765625	1.7297896523	2.9960785	0.59953994
170.820999861	0.0009765625	2.65588367404	9.019608	0.32505915
187.8317029481	0.0009765625	1.30069629289	2.25098	0.534579
169.8136169911	0.0009765625	1.52324045325	2.62353	0.723574
184.824858904	0.0009765625	1.31644441906	2.26667	0.919093

Table 5.5 summarizes several runs on the same image with same manipulations checking for adversarial perturbations in the second convolutional layer after parallelising.

Table 5.5: Results on CIFAR-10 checking second convolutional layer after parallelising

Running Time (s)	Manipulation (%)	Euclidean Distance	L1 Distance	Confidence
83.3600001335	0.00162760416667	2.2360679775	5.0	0.750232
82.9860000611	0.00162760416667	2.2360679775	5.0	0.750232
85.8410000801	0.00162760416667	2.2360679775	5.0	0.98030376

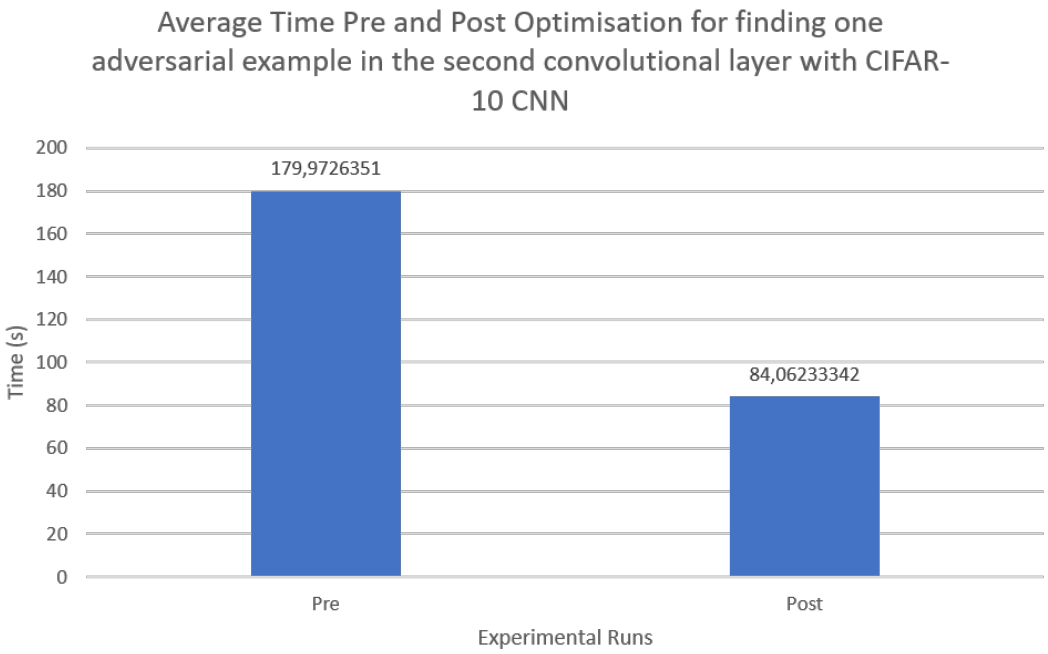

Figure 5.11: Comparison of average run-times pre and post optimisation for CIFAR-10

For the experiment with the convolutional neural network for CIFAR-10 dataset a speed up of an average of 53% could be achieved regarding only the second convolutional layer. Running the safety check through all layers took an average of 306.843 seconds, which seems reasonable if comparing the time a neural network takes to train and validate. Therefore this safety verification framework could be incorporated in the training process of a neural network without an overly huge performance loss.

The initial experiments with the MNIST dataset were done before using the OpenBLAS library and were inconsistent and could not be used for comparison and validation of the overall speed up through the parallelisation steps. For this reason, new experiments were done to get a baseline to compare to. Carrying out the experiments analogously to the experiments with the CIFAR-10 CNN, the results were as expected having achieved a significant speed up.

Table 5.6 summarizes several runs on one image with same manipulations checking for adversarial perturbations in the second convolutional layer.

Table 5.6: Initial results on MNIST checking second convolutional layer

Running Time (s)	Manipulation (%)	Euclidean Distance	L1 Distance	Confidence
11.1211271286	0.0573979591837	6.70588339224	44.9843	0.672662
12.1056849957	0.0573979591837	6.7082039325	45.0	0.601526
11.7709162235	0.0573979591837	6.7082039325	45.0	0.623106
11.3156499863	0.0573979591837	6.7082039325	45.0	0.623064
11.3661100864	0.0573979591837	6.7082039325	45.0	0.599762

Table 5.7 summarizes several runs on the same image with same manipulations checking for adversarial perturbations in the second convolutional layer after parallelising.

Table 5.7: Results on MNIST checking second convolutional layer after parallelising

Running Time (s)	Manipulation (%)	Euclidean Distance	L1 Distance	Confidence
8.18499994278	0.0573979591837	6.69190733949	44.890198	0.8366375
8.14800000191	0.0573979591837	6.7082039325	45.0	0.5614343
8.06299996376	0.0573979591837	6.7082039325	45.0	0.67067885
8.24900007248	0.0573979591837	6.7082039325	45.0	0.49200147
8.28600001335	0.0573979591837	6.7082039325	45.0	0.558569

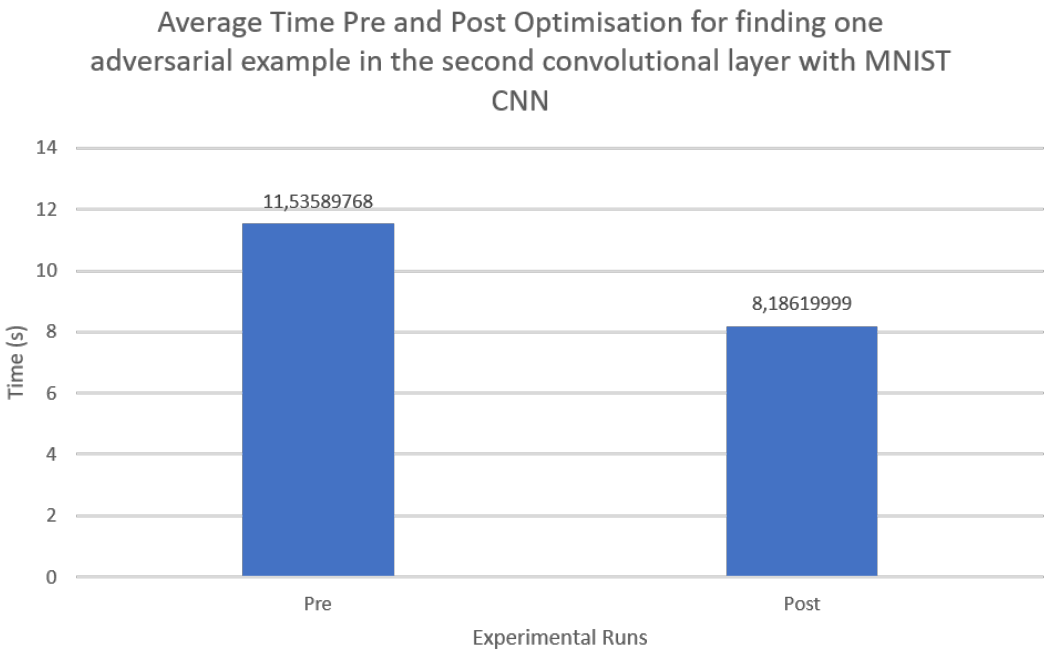

Figure 5.12: Comparison of average run-times pre and post optimisation for MNIST

For the experiment with the convolutional neural network for MNIST dataset a speed up of an average of 29% could be achieved regarding only the second convolutional layer. The speed up is lower compared to the results with the CIFAR-10 network as expected. Due to lower dimensionality and smaller inputs, the run time was already quicker from the beginning.

Conclusion

This dissertation aimed to analyse and optimise a safety verification framework for deep neural networks. Through research and analysis of existing approaches, a safety verification framework based on the research by Huang et al. was implemented and initially validated and compared with the experimental results. After a comparable state of the framework was reached, further research was carried out to find ways to optimize the framework in terms of run time performance. For this purpose, various BLAS libraries were tried out and the decision fell on OpenBLAS, since it was compatible with the present system the experiments were carried out and had promising benchmark results. After setting up an experimental environment and optimizing the framework through the usage of more suitable external libraries, several experiments were performed to have a baseline to compare against after optimising further. Further familiarization with the framework showed a possibility for parallelisation of the safety check within a layer. This was implemented through multiprocessing using a process pool and strictly separating the contexts of the framework to prevent segmentation fault or race hazards. After refactoring the framework to an acceptable state the same experiments were carried out to have a direct comparison. The experiments showed an improvement up to 53% with the CNN for the CIFAR-10 dataset and an improvement of 29% with the MNIST dataset when checking the safety of a single layer. Propagating a safety check through all layers of a deep neural network showed reasonable run times as well. Since this project was limited in time, no further experiments with other neural networks or configurations could be performed. These further experiments and possible other optimisation approaches are handled in the following chapter.

6.1 Further Work

Even though several experiments with CNNs for MNIST and CIFAR-10 datasets showed promising performance gains, evaluating the optimised framework with other neural networks like the VGG16 using the ImageNet dataset are still open and would be interesting to know how the framework will perform with more complex networks and larger input images with the incorporated improvements. Further refactoring of the framework could open up new opportunities for parallelisation, in terms of checking multiple layers concurrently.

Furthermore the safety verification could be incorporated into the training process of a neural network to automatically tune the network to be safe against certain adversarial perturbations. Another possible improvement might be to run the safety check on GPU with NVIDIA CUDA, which however requires an implementation of an SMT solver for CUDA.

In terms of performance additional refinements are possible using a different programming language to better utilise computational resources, and parallelise the framework even further.

Bibliography

[AHS+14] F. Agostinelli, M. Hoffman, P. Sadowski, et al. *Learning Activation Functions to Improve Deep Neural Networks*. 2014. arXiv: 1412.6830 [cs.NE].

[AOS+16] D. Amodei, C. Olah, J. Steinhardt, et al. *Concrete Problems in AI Safety*. 2016. arXiv: 1606.06565 [cs.AI].

[APF00] L. Ambrosio, D. Pallara, and N. Fusco. "Functions of bounded variation and free discontinuity problems". In: 2000.

[Beu] M. Beuckelmann. *Boosting numpy: Why BLAS Matters*. https://markus-beuckelmann.de/blog/boosting-numpy-blas.html. Accessed: 2019-12-25.

[BIL+16] O. Bastani, Y. Ioannou, L. Lampropoulos, et al. "Measuring Neural Net Robustness with Constraints". In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee, M. Sugiyama, U. V. Luxburg, et al. Curran Associates, Inc., 2016, pp. 2613–2621.

[BMN+] N. Bjørner, L. de Moura, L. Nachmanson, et al. *Programming Z3*. https://theory.stanford.edu/~nikolaj/programmingz3.html. Accessed: 2019-12-25.

[Cif] *CIFAR10 CNN MODEL*. https://github.com/fchollet/keras/blob/master/examples/cifar10cnn.py. Accessed: 2019-12-25.

[CIS+08] G. Carlsson, T. Ishkhanov, V. Silva, et al. “On the Local Behavior of Spaces of Natural Images”. In: *Int. J. Comput. Vision* 76.1 (Jan. 2008), 1–12.

[Dac17] J. Dacombe. *An introduction to Artificial Neural Networks (with example)*. [Online; accessed September 16, 2019]. 2017.

[Ehl17] R. Ehlers. “Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks”. In: *CoRR* abs/1705.01320 (2017). arXiv: 1705.01320.

[FFF15] A. Fawzi, O. Fawzi, and P. Frossard. *Analysis of classifiers' robustness to adversarial perturbations*. 2015. arXiv: 1502.02590 [cs.LG].

[GSS14] I. J. Goodfellow, J. Shlens, and C. Szegedy. *Explaining and Harnessing Adversarial Examples*. 2014. arXiv: 1412.6572.

[GSS15] I. Goodfellow, J. Shlens, and C. Szegedy. “Explaining and Harnessing Adversarial Examples”. In: *International Conference on Learning Representations*. 2015.

[HKW+16] X. Huang, M. Kwiatkowska, S. Wang, et al. *Safety Verification of Deep Neural Networks*. 2016. arXiv: 1610.06940 [cs.AI].

[HKW+17] X. Huang, M. Kwiatkowska, S. Wang, et al. “Safety Verification of Deep Neural Networks”. In: *Computer Aided Verification*. Ed. by R. Majumdar and V. Kunčak. Cham: Springer International Publishing, 2017, pp. 3–29.

[KB00] S. Kustrin and R. Beresford. “Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research”. In: *Journal of pharmaceutical and biomedical analysis* 22 (July 2000), pp. 717–27.

[KBD+17] G. Katz, C. Barrett, D. Dill, et al. *Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks*. 2017. arXiv: 1702.01135 [cs.AI].

[Ker] *Keras: The Python Deep Learning library*. https://keras.io/. Accessed: 2019-12-27.

[Kin] D. Kinghorn. *AMD Ryzen 3900X vs Intel Xeon 2175W Python numpy - MKL vs OpenBLAS*. https://www.pugetsystems.com/labs/hpc/AMD-Ryzen-3900X-vs-Intel-Xeon-2175W-Python-numpy---MKL-vs-OpenBLAS-1560/. Accessed: 2010-01-04.

[Kri07] D. Kriesel. *A Brief Introduction to Neural Networks*. 2007.

[KW17] J. Z. Kolter and E. Wong. "Provable defenses against adversarial examples via the convex outer adversarial polytope". In: *CoRR* abs/1711.00851 (2017). arXiv: 1711.00851.

[LB98] Y. LeCun and Y. Bengio. "Convolutional Networks for Images, Speech, and Time Series". In: *The Handbook of Brain Theory and Neural Networks*. Cambridge, MA, USA: MIT Press, 1998, 255–258.

[LBH15] Y. LeCun, Y. Bengio, and G. Hinton. "Deep learning". English (US). In: *Nature Cell Biology* 521.7553 (May 2015), pp. 436–444.

[Mal16] S. Mallat. "Understanding deep convolutional networks". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 374 (2016).

[Mni] *MNIST CNN MODEL*. https://github.com/fchollet/keras/blob/master/examples/mnistcnn.py. Accessed: 2019-12-25.

[MW+14] S. B. Maind, P. Wankar, et al. "Research paper on basic of artificial neural network". In: *International Journal on Recent and Innovation Trends in Computing and Communication* 2.1 (2014), pp. 96–100.

[NYC14] A. Nguyen, J. Yosinski, and J. Clune. *Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images*. 2014. arXiv: 1412.1897 [cs.CV].

[PMG+16] N. Papernot, P. D. McDaniel, I. J. Goodfellow, et al. “Practical Black-Box Attacks against Deep Learning Systems using Adversarial Examples”. In: *CoRR* abs/1602.02697 (2016). arXiv: 1602.02697.

[SAMR18] W. Schwarting, J. Alonso-Mora, and D. Rus. “Planning and Decision-Making for Autonomous Vehicles”. In: *Annual Review of Control, Robotics, and Autonomous Systems* 1.1 (2018), pp. 187–210. eprint: https://doi.org/10.1146/annurev-control-060117-105157.

[Sch14] N. Schraudolph. *A Simple Artificial Neuron)*. [Online; accessed September 16, 2019]. 2014.

[SNA17] M. Sarker, S. Noor, and U. Acharjee. “Basic Application and Study of Artificial Neural Networks”. In: *SK International Journal of Multidisciplinary Research Hub* 4 (May 2017), pp. 1–12.

[SVS17] J. Su, D. V. Vargas, and K. Sakurai. “One pixel attack for fooling deep neural networks”. In: *CoRR* abs/1710.08864 (2017). arXiv: 1710.08864.

[The] *Theano*. http://deeplearning.net/software/theano/. Accessed: 2019-12-27.

[WHK18] M. Wicker, X. Huang, and M. Kwiatkowska. “Feature-Guided Black-Box Safety Testing of Deep Neural Networks”. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by D. Beyer and M. Huisman. Cham: Springer International Publishing, 2018, pp. 408–426.

[Z3g] *Z3 Documentation*. https://github.com/Z3Prover/z3/wiki/Documentation. Accessed: 2019-12-25.

[ZHS09] J. Zou, Y. Han, and S.-S. So. “Overview of Artificial Neural Networks”. In: *Artificial Neural Networks: Methods and Applications*. Ed. by D. J. Livingstone. Totowa, NJ: Humana Press, 2009, pp. 14–22.

Appendix

A.1 Interim Report

DEPARTMENT OF ELECTRONIC AND COMPUTER ENGINEERING

DISTRIBUTED COMPUTING SYSTEMS ENGINEERING MSC

INTERIM REPORT

Safety Verification of Deep Neural Networks

Year of Submission: 2019

Contents

List of Figures

Acronyms

ANN Artificial Neural Network.

DNN Deep Neural Network.

ReLu Rectified Linear Units.

SIFT Scale Invariant Feature Transform.

MCTS Monte Carlo Tree Search.

MNIST database Modified National Institute of Standards and Technology Database.

SMT Satisfiability Modulo Theories.

Introduction

ANNs have been evolving greatly since the 1940's and are now used in many areas, such as autonomous driving. Image classification is used to detect street signs or traffic lights. But despite further developments, ANN are still mostly unstable due to adversarial perturbations. These adversarial perturbations are minimal changes to the input image which lead to misclassification. For applications in assisted driving or autonomous cars, these misclassification can lead to disastrous outcomes, for example forcing a car of the road for detecting a sign for turning left instead of right. Therefore to ensure safety in decision making a way to verify the classification of the ANN is needed. For this purpose several approaches were proposed using Satisfiability Modulo Theory to implement a verification framework or by using SIFT and MCTS. This thesis project focuses on evaluating existing approaches and developing a new verification framework based on the analysis and further optimisation of known methods. This report describes the overall topic of the thesis. Chapter 2 gives general insight in the practical and technical background. Chapter 3 discusses existing approaches. The following chapter states the aims and objectives of the project and describes presents possible methods and approaches for the project. Finally, the deliverables and expected outcomes will be determined and a detailed time plan for the project will be presented.

Project Background

In this chapter important background information for the project is provided.

2.1 Artificial Neural Network

Artificial Neural Networks are modelled to loosely mimic the human brain [Kri07]. They consist of multiple artificial neurons which are mostly connected unidirectionally. These neurons have an activation function which is used to process the input. The connections between neurons are so called edges, which pose to have weights for learning purposes. Neurons are structured in layers. Especially in Deep Neural Networks multiple layers are used. The neurons are connected between layers. In contrary of systems using an algorithmic approach, ANNs learn by example [MW+14]. A depiction of the basic structure of an ANN can be seen in following figure 2.1.

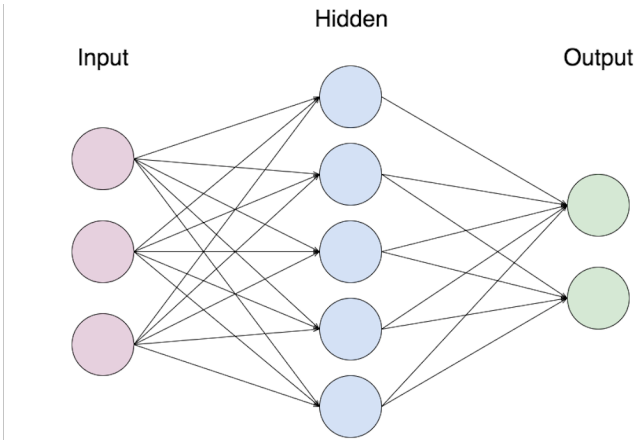

Figure 2.1: Simple Example of an ANN [Dac17]

2.1.1 Neurons

A neuron can have multiple inputs which are summed up in the first step. The sum is then processed with the activation function leading to the output of the neuron [ZHS09]. The basic structure of a neuron is depicted in figure 2.2.

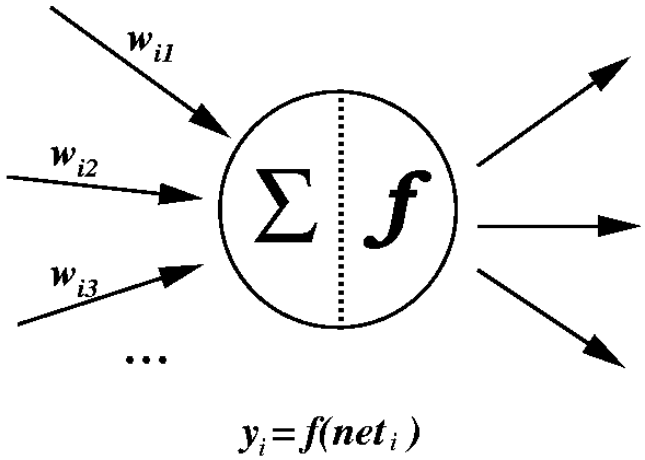

Figure 2.2: Simple Example of a Neuron [Sch14]

Common activation functions are following [AHS+14]:

- *Binary Step:* It is a threshold based function outputting a signal to the next layer when a value is above or below a certain threshold.
- *Linear:* The inputs are multiplied by a factor and therefore the output is linear to to the weighted input.
- *Sigmoid:* It is non linear and prevents jumps in the output values. The output values are bound between 0 and 1.
- *Rectified Linear Unit:* ReLu is very efficient and allows the network to converge quickly. Despite it looking like a linear function it is not and allows for back-propagation.

The activation function is selected depending on the input data and the expected outcome.

2.1.2 Types

There are several types of Artificial neural networks, which are mainly distinguished by the way neurons and layers are connected with each other [SNA17]. ANNs are often split in two categories, the supervised and unsupervised. The supervised training of neural networks is done by having labelled input data. The network is trained and compares against the labels to check if its classification matches the label. These networks are mostly used for classification problems and regression problems.

Unsupervised training has unlabelled data. The network has to automatically find structure in the data by extracting features and analysing them. It is often used when labelled data is too expensive or hard to gather. Since there are no labels to compare to, it is hard to measure the accuracy of a network trained unsupervised. The most common and simplest is the feed forward network. Here the process is unidirectional from input over hidden layers to the output. Another type is the Convolutional Neural Network, which is mainly used for image processing and image classification.

2.1.3 Layers

Depending on the type of network the structure of the layers in a neural network might vary. There is usually one input and one output layer in an ANN. The number of hidden layers and how they are connected with each other often determines the type of network in interaction with the activation function used. The input layer simply passes the information to the neurons in the first hidden layer. The inputs to the layers are often weighted to enable more adjust-ability. The neurons in a hidden layer process the data and pass it to next layer. The output layer combines the results and forms the output [KB00].

2.2 Practical Background

Using ANNs in highly automated systems like autonomous driving poses safety risks due to the unstable nature of those with respect to adversarial perturbations. Having scratches on a cameras lens or perturbations due to snow or rain or even an active manipulation of input data by an attacker can lead to misclassification of an image. Several reports in recent years stated that systems in self-driving cars misinterpreted traffic signs and eventually lead to accidents. There are several research papers concretising the problems with regard to safety in artificial neural networks or AI in general [AOS+16] [SAMR18]. Robustness, accuracy and confidence of a network are often not sufficient to ensure safety [NYC14]. For minimising the risk of misclassification and to verify ANNs, several approaches were introduced and two of them are highlighted in chapter 3.

2.3 Data Acquisition

For applying the verification algorithm, a pre trained network is required, since training and testing a network can take a long time. Therefore several pre-trained networks for different datasets will be considered, such as MNIST Handwritten Dataset, an ImageNet Dataset or a Dataset for traffic signs. Having a well trained network diminishes the probability of having errors in the verification process due to a badly trained network.

2.4 OpenCV

OpenCV is an open source computer vision and machine learning library [Bra00]. It provides a common infrastructure for computer vision applications as well as more than 2500 optimized algorithms. Since OpenCV 3.1 a DNN module was introduced. This module enables the library to use pre-trained deep networks using common frameworks like Tensorflow or Caffe.

Initial Survey

As already mentioned there are several approaches for safety verification of deep neural networks proposed in recent years. This chapter highlights two existing approaches giving an overview and general understanding of the matter. The first approach presented by Guy Katz et al. focuses on verifying certain properties of a DNN using ReLu activation functions. The second approach introduced by Xiaowei Huang et al. is focused on adversarial examples. The information given in the following two sections is entirely based on the research papers published by the groups mentioned [KBD+17] [HKW+16].

3.1 Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks

The project around Guy Katz from Stanford University USA introduced a new, scalable and efficient technique for verifying properties of deep neural networks with ReLus. Past methods of verifying DNNs with ReLus had the need of simplification, like only considering small input regions leading to only be able to verify an approximation of the DNNs property. Based on the simplex method it is extended to be able to handle the Rectified Linear Unit activation function. Therefore this technique is called Reluplex. This technique considers the whole neural network without simplifying it. The main goal of this method is to verify if input and output match certain constraints. The basic approach is to split the ReLu nodes in the hidden layer into two variables. A weighted sum variable and an activation function variable. Equations for the activation function variable and bounds according to the set constraints for the checked property are introduced.

Example Equations:

$$x_{2w} - x_1 = x_5 \tag{3.1}$$

$$x_{3w} + x_1 = x_6 \tag{3.2}$$

$$x_4 - x_{3a} - x_2 a = x_7 \tag{3.3}$$

Example Bounds:

$$\begin{aligned} &x_1 \in [0,1] \\ &x_4 \in [0.5,1] \\ &x_{2w}, x_{3w} \in [-\infty, \infty] \\ &x_{2a}, x_{3a} \in [0, \infty] \end{aligned} \tag{3.4}$$

The Reluplex then assigns the variables to solve the equations, but violating the ReLu constraints. These violations are then fixed on the go by updating the assignments to satisfy the bounds. Additionally to updating the assignments it might be necessary to manipulate the initial equations to satisfy some constraints. This will be done in several iterations until all bounds and constraints are satisfied and therefore the property to be checked is verified. In some cases the algorithm might run into a loop where case splitting may solve it.

To efficiently implement this algorithm the team introduced bound tightening and use a satisfiability modulo theory framework to manage case splits and bound tightening. This approach was evaluated using the ARCAS Xu which is a system using multiple DNNs for the decision making in collision avoidance for drones. They proved local robustness using this approach. For global robustness this worked out using small networks. The improvements of scalability are left for future work.

3.2 Safety Verification of Deep Neural Networks

The team around Xiaowei Huang of the University of Oxford developed an automated verification framework feed forward deep neural networks. Their basic approach is to search for perturbations in each region of an image by discretization and propagate the analysis layer by layer. In this method the safety verification is reduced to a search for adversarial example and therefore safety can be achieved if no misclassification are found in all layers. Also falsification can be detected which then can be used to tune the network or to be examined by a human. The method defines a region around a given point and a set of manipulations. These manipulations have to be specified for a certain classification problem. For layer by layer analysis a mapping between the layers is needed to determine how a manipulation in one layer affects the next. A region has to be determined such that it satisfies the mapping between consecutive layers. The refine-ability of manipulations has to be defined such that a manipulation in the previous is re-finable in the current layer if there exists a sequence of manipulations in the current layer which implements the manipulation in the previous layer. Having these preconditions and definitions the verification proceeds as follows: With k being the index of the current investigated layer, N being the network, x being the input, ηk being the region and Δk the set of manipulations.

Verify whether N, ηk, $\Delta k \models x$,

$if\ N$, ηk, $\Delta k \models x$ then

report that N is safe at x with respect to $\eta k(ax,k)$ and Δk, and continue to layer $k+1$;

if N, ηk, $\Delta k6 \models x$, then report an adversarial example.

This algorithm was implemented using SMT solvers. Experiments were done on several neural network classifiers for MNIST or traffic sign classification and led to found adversarial examples in seconds when working on few dimensions. But the verifying process is exponentially big regarding the number of features and therefore has a high complexity. They state that this approach is open to improvements through parallelisation regarding scalability and performance.

Organisation

This chapter outlines the organisational topics of the thesis project. Section 4.1 deals with aims and objectives. Section 4.2 describes the experimental methods to be used throughout the project. The last two sections contain the time plan of the project and determine deliverables and describe the overall expected outcomes of the project.

4.1 Aims and Objectives

The aim of the project is to evaluate existing methods and implement a new verification framework based on the analysis with possible improvements in scalability and performance.

The essential objective is to design an improved verification framework for deep neural networks based on the analysis of existing methods. Another objective is the evaluation of the framework and comparison to previous approaches.

4.1.1 Analysing and Developing a Verification Algorithm

Based on the research done throughout the project a verification framework will be developed using OpenCV. Since OpenCV works with different frameworks such as Tensorflow or Caffe, it is easier to test the newly developed verification against different networks and datasets, than implementing it for each of those. Having a implemented a framework or verification algorithm for OpenCV also enables to compare and evaluate against existing solutions and leaves room for improvements and optimisation.

4.1.2 Performance Evaluation

Evaluating the implementation is done by analysing point wise robustness and guarantee of decision safety. Robustness evaluation could be done by computing expected confidence diameter weighted by the test data distribution. Having precise manipulations, enables to use accuracy of the verification to evaluate the results. Introduction of an own metric for safety measuring might be possible.

4.2 Experimental and Investigative Methods

The initial survey brought insight and general understanding of the problem. The further procedure is to analyse the prototypical approaches in more detail and to implement a prototype in order to have a basis for testing, evaluating and comparing the newly developed framework. On this basis possible improvements and further optimisations regarding scalability and performance can be carried out. The resulting algorithm or framework can then be tested with various datasets and evaluation can take place. To distinguish the improvements of the algorithm a detailed comparison to existing methods will be done.

4.3 Time Plan

A time plan has been created to ensure efficient time management. Further sections will describe the time plan depicted in 4.1

4.3.1 Development

The first steps of development are designing the system and setting up the environment accordingly. When everything is set up and the data in terms of pre trained networks is prepared, the implementation commences. First tests are expected to be carried out during the end phase of implementation. During testing, the result will also be validated before evaluation can start. The final steps will be evaluating the results and comparing it to approaches and methods it is based on. The development should be completed by the beginning of January.

4.3.2 Writing

The writing parts will commence in parallel to the implementation. Introduction and Fundamentals can be written independently of the implementation. Writing about the implementation and evaluation will be commenced after the correspondent part of the implementation is done. The writing shall be concluded 2-3 weeks prior to submission to review and correct.

4.4 Deliverables and Specific Outcomes

The expected outcome is the research and study of safety verification for deep neural networks, as well as to determine limitations and possible optimisations of certain approaches and a possibly improved and optimized framework based on the research.

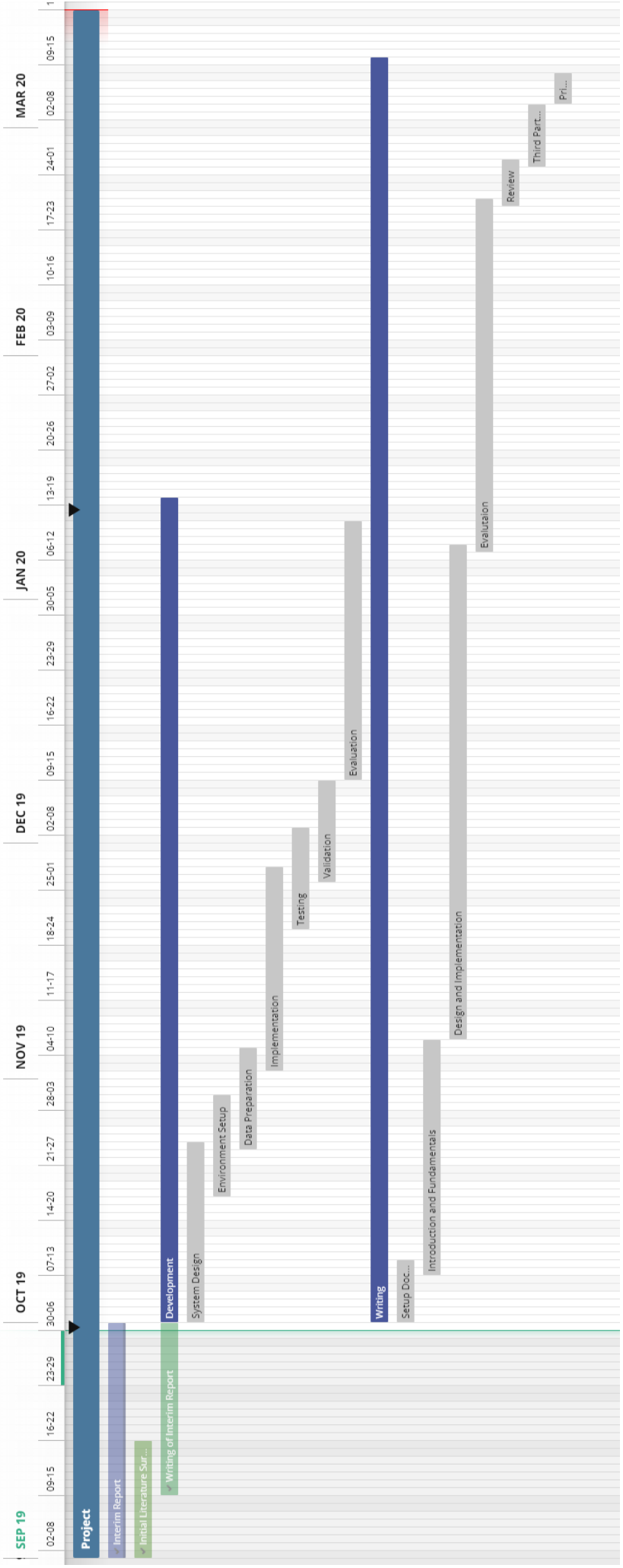

Figure 4.1: Time Plan

Bibliography

[AHS+14] F. Agostinelli, M. Hoffman, P. Sadowski, et al. *Learning Activation Functions to Improve Deep Neural Networks*. 2014. arXiv: 1412.6830 [cs.NE].

[AOS+16] D. Amodei, C. Olah, J. Steinhardt, et al. *Concrete Problems in AI Safety*. 2016. arXiv: 1606.06565 [cs.AI].

[Bra00] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[Dac17] J. Dacombe. *An introduction to Artificial Neural Networks (with example)*. [Online; accessed September 16, 2019]. 2017.

[HKW+16] X. Huang, M. Kwiatkowska, S. Wang, et al. *Safety Verification of Deep Neural Networks*. 2016. arXiv: 1610.06940 [cs.AI].

[KB00] S. Kustrin and R. Beresford. "Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research". In: *Journal of pharmaceutical and biomedical analysis* 22 (July 2000), pp. 717–27.

[KBD+17] G. Katz, C. Barrett, D. Dill, et al. *Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks*. 2017. arXiv: 1702.01135 [cs.AI].

[Kri07] D. Kriesel. *A Brief Introduction to Neural Networks*. 2007.

[MW+14] S. B. Maind, P. Wankar, et al. "Research paper on basic of artificial neural network". In: *International Journal on Recent and Innovation Trends in Computing and Communication* 2.1 (2014), pp. 96–100.

[NYC14] A. Nguyen, J. Yosinski, and J. Clune. *Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images*. 2014. arXiv: 1412.1897 [cs.CV].

[SAMR18] W. Schwarting, J. Alonso-Mora, and D. Rus. "Planning and Decision-Making for Autonomous Vehicles". In: *Annual Review of Control, Robotics, and Autonomous Systems* 1.1 (2018), pp. 187–210. eprint: https://doi.org/10.1146/annurev-control-060117-105157.

[Sch14] N. Schraudolph. *A Simple Artificial Neuron)*. [Online; accessed September 16, 2019]. 2014.

[SNA17] M. Sarker, S. Noor, and U. Acharjee. "Basic Application and Study of Artificial Neural Networks". In: *SK International Journal of Multidisciplinary Research Hub* 4 (May 2017), pp. 1–12.

[ZHS09] J. Zou, Y. Han, and S.-S. So. "Overview of Artificial Neural Networks". In: *Artificial Neural Networks: Methods and Applications*. Ed. by D. J. Livingstone. Totowa, NJ: Humana Press, 2009, pp. 14–22.