Electronics and Computer Science
Faculty of Physical and Applied Sciences

University of Southampton

Konstantinos Karras Kallidromitis

30th April 2019

Adversarial Examples and security of Deep Neural Networks

Project supervisor: Dr Srinandan Dasmahapatra
Second examiner: Dr Xu Fang

A project report submitted for the award of
BEng in Electrical and Electronic Engineering

Abstract

The project investigates the impact of adversarial examples on two security critical scenarios: traffic sign classification and facial recognition. The attacks and defences on the networks are considered realistically and based on their feasibility in the real world. Traffic sign recognition is initially tested on a ResNet-34 network, against a variety of graffiti patterns that can cause untargeted and targeted misclassifications. The graffiti attacks can however be completely negated by applying adversarial training and including graffiti images in the learning process. An adversarial patch, which can be used as a sticker and placed on signs is also tested. Defensive distillation proves to be ineffective against well trained patches and only able to prevent attacks close to classification boundaries. Facial recognition is investigated on the SphereFace implementation and evaluated on the LFW Benchmark. Initially, different types of noises are sampled and the impact on the pair accuracy is tested. Defensive denoising models are employed which improve the accuracy of the model and extract discriminative features from the images. Then a black-box boundary attack is used as a targeted attack to gain momentum. Several input transformations are used including bit-depth reduction and JPEG compression which mitigate the attack.

Statement of Originality

- I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.
- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.
- I have acknowledged all sources, and identified any content taken from elsewhere.
- I did all the work myself, or with my allocated group, and have not helped anyone else.
- The material in the report is genuine, and I have included all my data/code/designs.
- I have not submitted any part of this work for another assessment.
- The www.planetb.ca website was used to format the code into the word document.
- My work did not involve human participants, their cells or data, or animals.
- Open libraries used in software development: PyTorch, Matplotlib, SciPy, Pandas, NumPy, Jupiter Notebook, ImageMagic, Mtcnn, OpenCV, Foolbox, Cuda, Advertorch

Acknowledgements

I would like to thank my supervisor Dr Dasmahapatra for all his support and guidance in the machine learning and mathematical concepts required for this project. I would also want to thank my family and friends, especially Sophia, for their support and patience through this project.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

Deep Neural Networks (DNNs) have been leading the technological revolution with increasing applications in security-critical systems [1]. Advances in trained classifiers have led to models with high accuracy and sophistication. However, recent studies have shown the limitations of neural networks, when maliciously crafted inputs can easily mislead the models and cause them to misclassify outputs [2][3].

Adversarial attacks are defined as perturbated inputs in a model that have been specifically created to cause an error in the model [4]. These attacks are frequent in image classification tasks, were images can appear imperceptible to the human observer but can still cause a mistake in the classification [2]. With the increasing applications of neural networks across multiple industries, adversaries can take advantage of inputs and give rise to misleading results with dangerous consequences. Examples that express the importance of defending against such attacks can range from the misinterpretation of a stop sign from a self-driving vehicle to an error in facial recognition from security cameras [5].

The research of adversarial inputs has so far been related to abstract games and elementary datasets, with unrealistic boundaries between the model and the adversary. These examples cannot be used to express real-world security concerns and are detached from the defined rules of adversarial research defined by *Goodfellow et al.* [6]. This paper analyses adversarial examples of real-world scenarios and attempts to make robust, reliable defensive networks that can be implemented for realistic problems.

1.2 Project Scope

In previous research (*Kurakin et al., 2018* [4]; *Huang et al., 2017* [7]) adversarial scenarios are investigated in the context of a game between an attacker and a defender. For one side to win it is not enough to have an effective strategy, but also anticipate the actions of the opposing party. The scope of the project is to act both as an adversary and defender in order to investigate the impact of adversarial examples on specific real-world security critical systems and attempt to defend against a plethora of different attack types.

The first system is a street sign classifier created to inform self-driving vehicles which traffic sign type is detected. With the rise of automation, errors in recognition of vital traffic control indicators such as stop signs can prove fatal. The second system is based on the concept of facial recognition. The adversarial example (attack) generation will be approached differently in each scenario. Attacking a traffic sign will be focused on the physical environment and will consider multiple limitations that exist in the real world [5]. On the other hand, facial recognition attacks will be based on previous work on sophistically generated from a computer and are usually undetectable from the human eye. These attacks are based on decision and optimization methods which have been extensively used to show the dangers and weaknesses of neural networks [3][8]. A range of adversarial attacks is considered, with the aim to examine their impact on state-of-the-art neural network architectures, while considering if these can be mitigated in the real world.

Chapter 2

Background and Literature Review

2.1 Artificial Neural Networks

The following section explains necessary concepts that are used in image recognition to significantly improve the network's performance. These methods are used in the models that will be examined for both systems and are vital in order to understand how the attacks and defences work. Moreover, the section provides information on the characteristics of different attack and defence methods and briefly introduces which methods will be examined.

2.1.1 Convolutional Layers

In recent years the success of CNNs in computer vision has led the accuracy to unprecedented levels [9]. The state-of-the-art network structures considered contain multiple convolutional layers, which allow them to learn more complex features of an image. Following the convolutional layers is always a batch normalization layer which reduces the covariance shift and speeds up the training process [10].

Figure 1 shows the operation of two successive convolutional layers on a training image with three input channels (RGB). The picture is divided into feature maps that are projected on the first layer. The feature maps are extracted by applying image filters to the inputs which are located on each convolution [11]. For each successive convolution, the filters become more complicated and specialised to detect specific classes. A deep understanding of the methods a framework uses to distinguish images is a prerequisite for the creation of methods that are resistant to adversarial attacks.

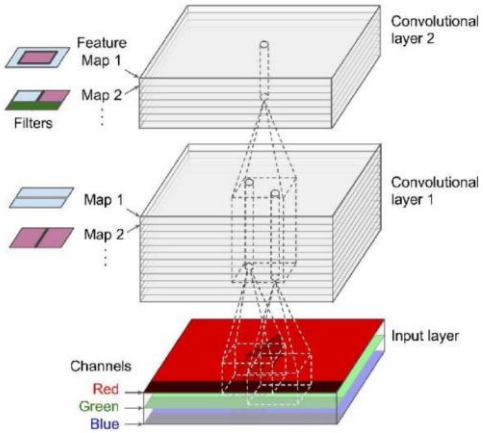


Figure 1: *Successive convolution layers with feature map representation. Reproduced from [11]*

2.1.2 Residual Networks

The latest trend in deep learning is Residual Networks (ResNets), which contain skip connections (shortcuts) in their architecture. Traditional networks reach a limit in the test accuracy as additional layers are added since the chance of overfitting to the training data also increases. The residual shortcuts help networks learn and optimise at a faster pace than traditional DNNs allowing the networks to stack multiple layers [12][13]. This is achieved since a network which models a function $F(x)$ uses residual learning by placing a skip connection in the input x making the structure now model a function $F(x) + x$ shown in Figure 2.

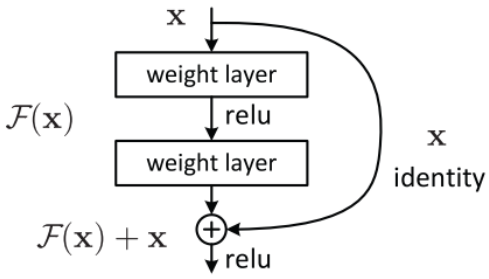


Figure 2 : *Single Residual (skip) Connection. Reproduced from [13].*

Because the weights are initialised close to zero, the target will initially resemble the identity function. Skip connections will equal the input to the output (identity), and the signal can directly travel several layers, which make the learning process faster [11].

2.1.3 Traffic Sign Classification Network

Figure 3 shows the 34-layer ResNet architecture which is implemented in the physical world, to classify traffic signs for a self-driving vehicle (Chapter 3). The arrows represent skip connections, and the dotted lines imply an additional increase in dimensionality. An increase in the dimension is caused when the destination layer of the shortcut, has different dimension size from the original layer, such as 64 to 128 [14]. In the final network the fully connected (FC) layer (shown as a white box), is modified from 1000 to 43 outputs to represent the different number of classes that the network is trained on.

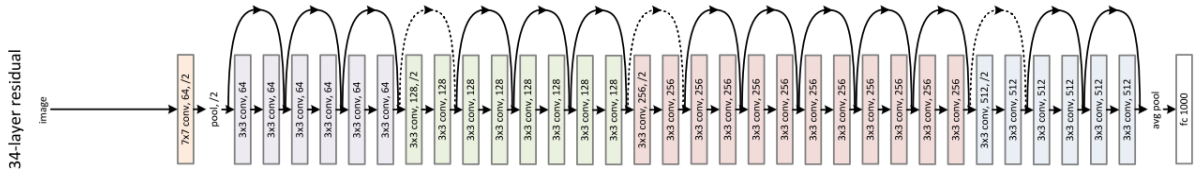


Figure 3: *Indicative deep residual 34-layer network architecture. Reproduced from [13]*

2.1.4 Facial Recognition Network

Facial Recognition (FR) is a significantly more complex problem than street sign classification. Each traffic sign has distinct features which makes it easily distinguishable and there are few sign types compared to billions of individual faces. A practical approach to face recognition also considers open-set testing. Open set testing is when the network is evaluated not only on different data but also classes. This implies that the network should be able to cluster similar faces without assigning them to a specific person. This method makes the evaluation of the network harder, but it is a more realistic approach to the problem.

The limitations of the system require a more sophisticated network architecture to achieve a high accuracy value. SphereFace is a deep hypersphere embedding approach which uses Angular (A) SoftMax loss to learn facial features [15]. The network diagram is similar to the ResNet-34, employing residual connections and stacking convolutional layers. However, it has a distinct A-SoftMax loss function that is specifically designed to learn discriminative features, on a hyperspace which intrinsically matches the facial characteristics of the data.

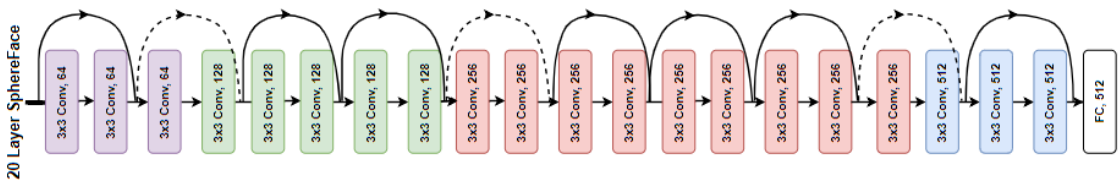


Figure 4: *SphereFace 20-layer network architecture. Diagram was created using draw.io software.*

2.1.5 Face Detection Network

In facial recognition problems, a major issue is the background interference which can influence the network and prevent it from focusing on the actual face area. To mitigate the impact of the surroundings, the input images are first placed on a face detection assembly of networks called Multitask Cascaded Convolutional Networks (MTCNN) [16]. The network can distinguish the area of the face, which is afterwards used for recognition. A pre-trained version of MTCNN is considered to pre-process the data before they are inserted in the main classifier.

MTCNN is a state-of-the-art approach to extract the bounding box (rectangular boundary) of a face in an input image. The first network is the Proposal Network (P-Net) which is used to detect Regions of Interests (RoI). These are possible regions that a face can be located in and are computed by inputting in the network various sizes of the input image (image pyramid). An important step after every network output is non-max suppression (NMS) which minimizes the number of ROIs by voting candidate bounding boxes according to the variances of neighbouring boxes [17]. The Refine Network (R-Net) is employed to reject false candidate locations and calibrate the bounding boxes. The Output Network (O-Net) focuses on regions with more supervision and detect high probability facial landmarks (eyes, nose, mouth). An example of how object detection operates is shown in Chapter 4, Figure 20.

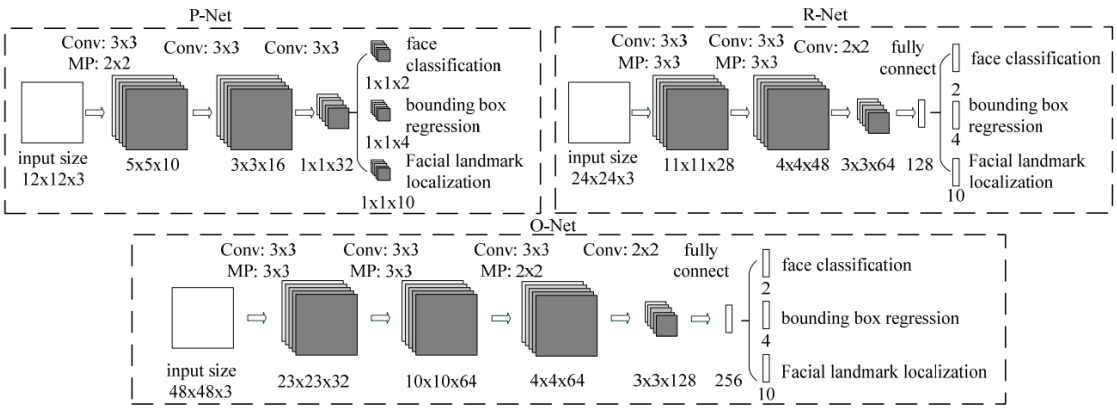


Figure 5: *Multitask Cascaded Convolutional Networks Architecture. Max Pooling (MP) and Convolution Layers (Conv) are displayed for each network. Reproduced from [14].*

2.2 Adversarial Examples

2.2.1 Evaluation Metrics

In this project, there are two different types of networks considered. The first one is a classification network which assigns an output to a class with the highest probability. The second model is a recognition network which outputs a vector for each input. Multiple inputs can be compared and clustered according to the distances of their vectors.

2.2.1.1 Classification Confidence

Adversarial attacks and defences on a classification network will be examined based on two metrics: accuracy and confidence. Accuracy relates to a correct or incorrect classification and it is used to define a successful attack. On the other hand, confidence is the probability assigned to the network prediction. An attack that is extremely effective on a network architecture would not only misclassify an input but cause the network to also have a confident (high probability) wrong prediction.

Confidence of a prediction is measured using the SoftMax function on the output labels of the network [18]. Based on its inputs, SoftMax outputs a distribution of probabilities that add up to 1. Assuming an array of N numbers, the probability of the n^{th} number (P[n]) in the array is expressed by equation (1):

$$\mathrm{P}[n] = \frac{e^{Q_n}}{\sum_{i=1}^{N} e^{Q_i}}, n = 1, \dots, N \quad (eq.1)$$

To find the probabilities for the whole array, the equation is applied to all the numbers $n \in \{1, N\}$, where $Q(n)$ is the value of the n^{th} position in the vector.

2.2.1.2 Cosine Similarity

The challenge in evaluating a recognition network stems from the different datasets that the network is trained and tested on (open-set). In the real world, there are billions of different people and it would be impossible to have a dataset with every face worldwide. This implies that the networks should be able to separate people regardless if they are in the training data. This is achieved by calculating the cosine of the angle between the output vectors using cosine similarity [19]. Two identical vectors would overlap and have an angle of 0° and $cos(\theta) = 1$ which is the maximum value of cosine similarity. The equation (2) is used to calculate the similarity between two vectors (A, B) where A_i, B_i are the i^{th} numbers in the vectors [20].

$$\cos(\theta) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}} \quad (eq.2)$$

2.2.2 Characteristics

When implementing perturbations in real-world scenarios, the adversarial theory requires the following categories to be defined to simulate the behaviour of the attacker and the defender. Moreover, the danger and effectiveness of the methods will be described and judged based on these characteristics.

2.2.2.1 Classification Target (Goals)

An untargeted attack aims to make the model make any mistake when classifying an input. A targeted attack is designed by the adversary to misclassify an input image as a specific class [21]. However, a targeted attack usually involves knowing the model parameters and gradient to point the classification in a specific direction. An additional goal of the attacker can be to lower the confidence of the prediction instead of a mistake in classification.

2.2.2.2 Knowledge of model parameters

The knowledge of the attacker is vital to determine the methods by which adversarial examples will be generated. A white-box attack is when the adversary has full knowledge of the parameters of the network including trained weights, optimisation methods and the model architecture [6]. Those parameters can be used from methods such as FGSM (Fast Gradient Sign Method) to enhance the attack and increase the chance of misclassification. In the black-box with probing scenario the adversary does not have the knowledge of the model parameters but can sample inputs and inspect the results. The last scenario is the black-box without probing in which the model is entirely unknown, and outputs cannot be observed. In this case, the adversarial examples are based on transferability and include general attacks that will impact most neural networks [22].

2.2.2.3 Types of Attacks

Digital attacks include scenarios in which the adversary has access to the digital input data used in the model. An example of a real-world situation would be using software to modify facial recognition input images. A physical attack is when direct changes or addition of objects (object transplant) occur in the physical environment of an input image [23][4]. The last category, which will also be examined is computer simulated physical attacks which involve using software to simulate an attack that can affect the physical layer.

2.2.2.4 Action space (Magnitude)

Action space is one of the most important parameters to measure the performance of attack and defence methods. The action space defines the scale of the technique. An adversarial example that is unrecognisable from the original image is significantly harder to classify compared to an L-BFGS attack which uses the minimum, imperceptible perturbation that can misclassify the input [2]. The less action space an attack needs to be effective, the stronger the attack.

The categorisation of action space is examined in [6] and will be used to distinguish between the scale of attacks. The different magnitudes of adversarial examples are indistinguishable perturbation, content-preserving perturbation, non-suspicious input, content-constrained input and unconstrained input.

2.2.3 Related Work

2.2.3.1 Physical Environment Attacks

Adversarial examples implemented on physical systems will be examined in the context of traffic sign recognition. Most of the research in adversarial examples is focused on small perturbations which require "digital-level" access to the network inputs. However, such a scenario in the real-world has been widely criticized as unrealistic since small changes in the image can remain undetected from a vehicle sensor (camera) or lose their effectiveness when viewed from different angles and distances [24][25]. In addition, the attacker especially in the context of self-driving does not have digital access to the network and can only influence the physical layer [26]. These limitations decrease the possible attacks that can be implemented successfully on a physical system.

Recent adversarial research papers have proposed new types of attacks that can be effectively placed on the physical environment. The first example of such an attack was from *Kurakin et al.* [8], who was able to successfully misclassify images, by printing adversarial examples. However, it should also be mentioned that there were also significant errors in the printing process which forced the attack to produce larger perturbations to be effective. Following this research, there have been numerous other attack types such as transferable stickers [27], graffiti [5] and algorithms robust to physical world transformations [26][28].

2.2.3.2 Computer Generated Attacks

Computer-generated attacks have been heavily researched in the context of adversarial machine learning. However, this project will focus on computer-generated attacks specifically in facial recognition. Facial recognition uses two networks, a detector and a recognition network. This implies that attacks are harder to investigate since changes will affect both models. Moreover, the recognition network is evaluated on different data than the ones it was trained on. Attacks must focus on reducing the calculated similarity of images, which is harder to compute than simply misclassifying an input [29].

A major attack that commonly occurs in the real-world revolves around security cameras. Since cameras are recording for large durations each day, the video has a bad quality and a lot of noise. Noise can appear in different categories and each type is an individual attack requiring a different solution. Moreover, it is important to consider a black-box scenario with probing, where the adversary has access to the network and is allowed to sample inputs and generate decision-based adversarial examples such as a boundary attack [22].

2.3 Adversarial Defences

2.3.1 Characteristics

Adversarial defences tend to be significantly harder to implement to achieve robust neural networks compared to attacks. This can be explained because as seen in the previous section, adversarial attacks have many different categories which are focused on influencing specific parts of the network. Therefore, it is much easier to cause an error in the network than to neutralize all possible scenarios. This is further supported by the research of *Warren et al.* [30], where it is shown that multiple defences do not always coexist to form a robust model. Moreover, machine learning models are required to successfully predict many classes with high confidence which can make them further vulnerable to adversarial examples [31].

2.3.1.1 Defences on Training

Training defences are focused on modifying the learning procedure of a network. The most common method is adversarial training which includes mixing adversarial inputs in the training data. However, there are significantly more complex processes such as regularization schemes for ReLU networks [32] or even methods such as Defensive distillation which involve training multiple networks [33]. These methods have proven effective when countering indistinguishable perturbations and small-scale attacks that are undetectable by a human observer [34].

2.3.1.2 Defences on Testing

Testing defences are generally categorized into detectors and manipulators. Detectors are additional networks that are focused on identifying tampered inputs and preventing them from entering the network [35]. These networks can come in various proposed forms, from a MagNet [36] to the latest implementation of Generative Adversarial Models [37]. Manipulators can transform the test images in a way that mitigate the changed inputs. These transformations come in the form of randomization [38] and denoising models [39].

2.3.2 Related Work

2.3.2.1 Real-World Defences

Adversarial examples in a physical-world scenario tend to be unconstrained and unpredictable. The difficult nature of the attacks, combined with the lack of research papers on real-world defences makes it harder to predict which methods will ensure the robustness of the network [8]. Nonetheless, there are still some methods that can be considered to have an effective impact on mitigating physical-layer adversarial attacks.

For large scale attacks, such as graffiti, a detector network or adversarial training can counteract the changes in the image. However, this is only feasible if the data in the training process have experienced a similar attack shape and position. More sophisticated attacks such as Expectation over Transformation (EOT) can prove significantly harder to mitigate [28]. A possible solution could be defensive distillation which is a form of label smoothing which can widen the boundary and improve the classification.

2.3.2.2 Computer Simulated Defences

In the past few years, with the increase in adversarial example research, there have been multiple papers on black-box defences. These defences are harder to implement since black box attacks are unpredictable [6]. The attacks are not generated as a minimum perturbation but are transferable or decision-based.

Decision-based attacks are iteratively implemented until a specific criterion is satisfied (usually misclassification). In a black box scenario, they are implemented on specific images or models and without using the gradient or other model parameters. These attacks are commonly found in a security camera scenario in the form of noisy images. Additive noise attacks can benefit from denoising networks which smooth pixel values. However, if there is enough noise in an image, it is possible that a denoising network which attempts to remove the noise will destroy discriminative features and make a prediction worse. Input transformations such as bit-depth reduction and JPEG compression are also employed in defensive scenarios, to limit attacks that are robust to the traditional denoising models [40].

Chapter 3

Traffic Sign Classification

3.1 Traffic Sign Classifier

The scenario of street sign classification has been increasingly more relevant with the rise of self-driving vehicles. Currently, automotive companies have implemented partial self-driving software that can be used for simple driving tasks such as automatic lane detection. However, complete automation requires a significant increase in the sophistication and robustness of the algorithms. One of the major problems facing the automotive industry is street sign recognition and the life-threatening problems that can arise from a misclassification in the network.

The Pytorch library is used to train the ResNet-34 classifier since it creates the networks dynamically making it faster to code and debug. A classification network receives input data and outputs class labels representing the types of traffic signs. As seen in Figure 7, the centre and largest part of each image is a traffic sign and the background interferences are minimized. Adversarial attacks will thus be generated on the traffic signs' area, which will isolate the attacks and improve the investigation without the influence of external objects.

3.1.1 German Traffic Sign Recognition Benchmark (GTSRB)

The dataset includes 39,209 training and 12,630 testing images with 43 different traffic signs. It has been widely used in numerous papers in the field of automated driving and can be employed to measure the impact of adversarial examples on a self-driving vehicle. Due to class imbalances (Appendix B) the accuracy will vary between different classes. Most of the adversarial examples will be centred around vital and commonly found signs in traffic control. It is also important to consider the impact an attack would have in real life, for example, a stop sign error can cause a severe accident, which makes the correct classification of stop signs vital.

Figure 6: *Samples from the 43 classes included in the GTRSB dataset. The images represent the class numbers from class 0 to (top-left) class 42 (bottom-right).*

3.1.2 Data Pre-processing

The GTRSB dataset is pre-processed to ensure the inputs are used to effectively train the Resnet-34 network. Initially, a validation set is created by randomly taking 20% of the training images in each class, to ensure that the validation set includes all dataset classes (Figure 6) and is not affected by imbalances. Then using the NumPy random number generator the training and validation data are shuffled, which is important to ensure the network is constantly trained on the variety of different signs in the data.

All inputs need to be at the same size since they are processed in parallel when trained in batches of 200 images to speed up the training. To ensure minimum loss in information the average aspect ratio(width/height) is initially calculated (~ 1), and the images are normalized and then reshaped according to the ratio ($112 \times 112\, pixels$). The Cuda library is used to train the model on a GPU which greatly accelerates the learning process.

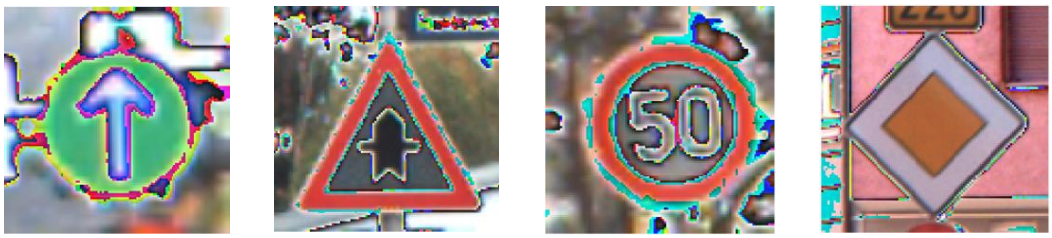


Figure 7:*Normalization and resizing of traffic sign images before they are used in training*

3.1.3 Training the Network

Figure 8 shows the training process of the ResNet34 network, which can achieve a high accuracy (94.82%), using the convolutional and residual methods discussed in Chapter 2. The input images are inserted in batch sizes ($N = 200$) and then there is a series of convolutional layers, with the first convolution of size 7x7. The last step includes a fully connected layer with output matrix dimensions $N \times 43$ (batch size $\times$ # classes). A SoftMax equation is used to output the class probabilities and then the results are inserted into a cross entropy loss function. The cross-entropy loss function will be further discussed in the defensive distillation method.

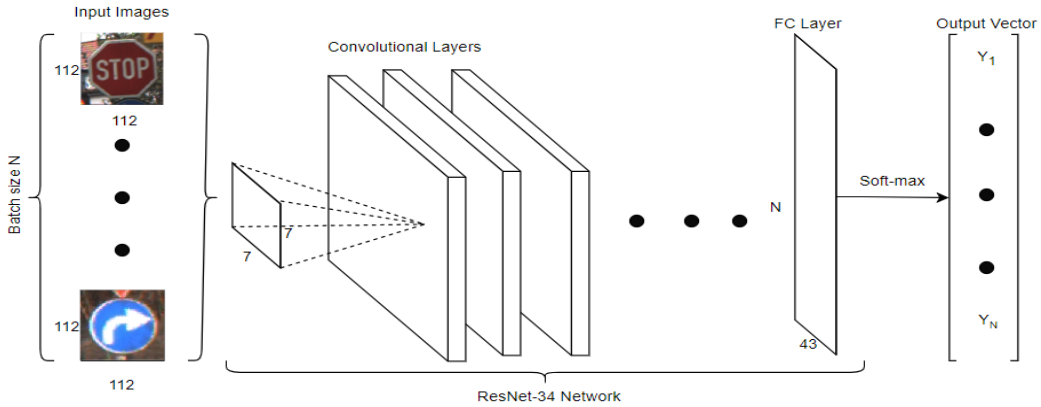


Figure 8: *Indicative structure of the training process of the ResNet-34 Network. The diagram was created using the draw.io software.*

To achieve high accuracy on a test set it is vital for the network to be able to generalise on data it has not been trained on. A validation set is used every epoch (learning iteration) and the model with the highest validation accuracy is used, not the one that trained for more iterations. This can be seen in Figure 9, where the maximum accuracy value occurs before the final iteration in the network. This methods aids the learning process and significantly mitigates the tendency of neural networks for overfitting after multiple iterations [41].

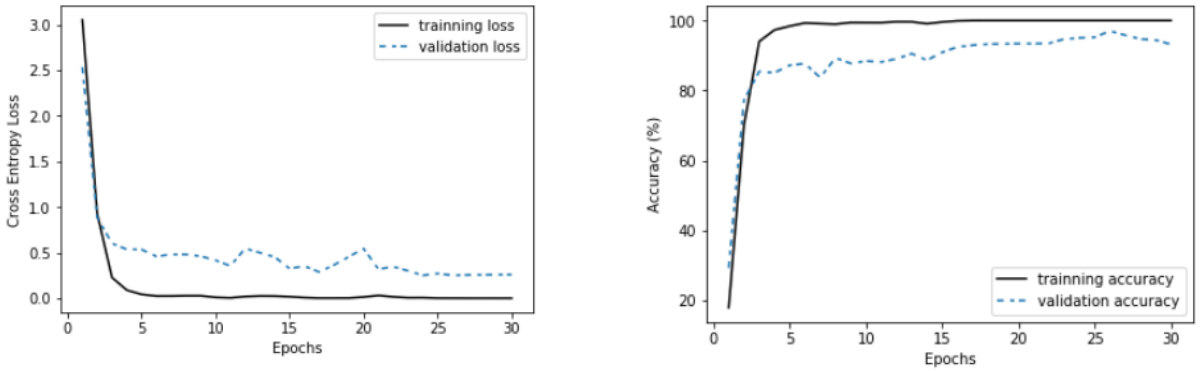


Figure 9: *ResNet-34 loss (left) and accuracy (right) diagrams for GTSRB. Diagram are generated using matplotlib.*

3.2 Adversarial Examples

3.2.1 Graffiti Attacks

Graffiti can be considered as human-generated attacks and are commonly found on traffic signs in the physical world. They ten to be very hard to predict by a neural network since they significantly vary in shape and colour. In the context of adversarial machine learning, graffiti attacks can be characterised as physical black-box attacks, since no knowledge of the model's parameters is required. The graffiti attacks that will be considered relate to content-constraint scale, meaning that a human observer can recognize the original sign and at the same time clearly see the attack. However, most of the graffiti would be unsuspicious to humans. To properly examine the impact of the graffiti, different areas, colours and positions will be implemented.

The attacks displayed in the figures are generated on correctly labelled images with 100% SoftMax confidence from the classification network. The images have already been resized to (3,112,112) pixels to ensure the attack has the same size and position on all images. Moreover, the attacks are computer generated to simulate a real-life scenario and only the signs that fit the attack within the sign area are considered.

Table 1: *Characteristics of different graffiti attacks examined*

	Size (Pixel Area)	Colours	Number of Individual parts	Total Accuracy (%)
Attack 1	900	Black	3	2.61
Attack 3	400	White	1	57.42
Attack 2	500	Black/White	4	7.44

3.2.1.1 Attack 1

The first graffiti attack is similar to the one sampled by *et al. Eykholt* [5] and it is used to investigate the confidence of the network's predictions. It uses black paint and it covers an area of 900 pixels (7.2% of total area). In Table 1, the first attack was shown to be the most effective in reducing the accuracy of the classifier. The low probability can be explained since most sign types depend on a black symbolling.

More specifically, Figures 10a and 10c show that signs most vulnerable to this attack include "speed limits" and "triangular signs". These sign types have multiple classes each ("speed limit" signs 30,50,70), which have close discriminative boundaries (similar features). A new design will confuse the network which will assign the input to the closer class from its training data. However, since the new design, which includes the black graffiti attack does not match perfectly the training samples, the confidence of the network will be significantly lower. In Figures 10a and 10c, the misclassified signs have a smaller probability value (65.27% and 63.72%).

Blue directional signs such as the "keep left" sign in Figure 10b were overall less affected by graffiti attack 1. This can be explained since all the similar signs involve only white and blue colours and a change in some pixel values to black is not enough to push the sample images over their class boundary.

10a ***10b*** ***10c***

Figure 10: *Graffiti attack 1 changing the classifier sign predictions. 10a: misclassified as "speed limit 80" sign (65.27%), 10b: classified as "keep left" sign (99.99%) and 10c: misclassified as "beware of ice" sign (63.72%). Misclassified signs are shown top-left from source: road-signs.uk.com.*

3.2.1.2 Attack 2

Attack 2 was created as a targeted attack. This graffiti pattern was specifically designed to attack the "speed limit 50" signs (class 2). The purpose of the attack is to hide part of the number 50 and cause the network to misclassify it as a "speed limit 30" sign (class 1). It is important to note that speed limit signs have fewer risks involved to cause an accident, due to a misclassification, which makes the attack less threatening. However, the success of the technique shows that an adversary can create a targeted attack in a black box scenario without probing. Figure 11 shows that the attack was successfully able to misclassify 75% of the "speed limit 50" signs and with high accuracy (99.99%). However, overall the attack was significantly less effective in the remaining traffic signs (Figure 11b) and thus the total accuracy, shown in Table 1, is much higher than the rest at 57.42%.

An unsuspected and dangerous effect of the attack was to also misclassify 64% of the test set stop signs as "no entry signs". Stop signs have very distinct features which make them naturally very robust against adversarial attacks. Even the stop sign misclassified in Figure 11 has low confidence in the prediction (67.24%). Nonetheless, a human observer would be able to clearly identify the stop sign from the distinct shape and letters. Adversarial examples have the potential to prevent the network from prioritizing more distinctive features of an object and instead focus on the attack (stop signs were specifically classes as no entry signs).

11a ***11b*** ***11c***

Figure 11: *Graffiti attack 2 changing the classifier sign predictions. 11a: misclassified as "speed limit 30" sign (99.99%), 11b: classified as "no vehicles" sign (100%) and 11c: misclassified as "no entry" sign (67.24%).*

3.2.1.3 Attack 3

The third attack combines two colours and multiple positions and was generated to simulate the aftereffects of removing a poster or a sticker from a traffic sign. This attack has been natively observed in the training data and occurs often in real life. The attack was able to misclassify a large amount of traffic signs and decrease the total accuracy to 7.44%. This makes the attack highly dangerous and shows that multiple colours and orientations can further confuse the classifier and are likely to cause accidents in the real world. Most of the traffic signs that were misclassified are triangular signs (Figure 12b), which include similar designs in the centre resulting in a high false confidence (92.6%).

12a ***12b*** ***12c***

Figure 12: *Graffiti attack 3 changing the classifier sign predictions. 12a: classified as 'no entry sign' (99.99%), 12b: misclassified as 'road work sign (92.6%) and 12c: classified as 'stop sign' (99.93%).*

The graffiti and poster aftereffects are not purposely placed on traffic signs to trick classifiers but have the potential to drop the accuracy to dangerous levels. Their different variations and high occurrence can make them especially dangerous and likely to cause accidents if they are not prevented. More diverse graffiti patterns such as the third attack (Figure 12) are especially malicious since they only occupy 4% of the total image area and appear unsuspicious to humans.

3.2.2 Expectation over Transformation

As mentioned in Chapter 2 there are multiple limitations when considering adversarial examples in the real world in generating but also in detecting an attack. The Expectation over Transformation (EoT) method proposed by *et al. Athalye* [28] has proven to overcome these difficulties. The method is based on the principle that real world inputs occur at different angles, rotations and distances. This means that a robust attack needs to be able to overcome these transformations and consistently misclassify the result [42]. A relevant example is when the car approaches a street sign, the classifier needs to be constantly fooled for a successful attack.

Traditional white-box adversarial examples, like L-BFGS, are generated by finding a changed input x' which maximizes the log-likelihood of the misclassified target class y_t (eq. 3):

$$\underset{x'}{\operatorname{argmax}}\, log\ P(y_t|x') \qquad (eq.3)$$

The adversarial examples are generated to meet the L_∞ norm bound $\|x' - x\|_\infty \leq \varepsilon$ which is used to bound the new input to ensure there is only a small difference with the original image [43].

The expectation over transformation method is employed to model the different transformations that can occur within the optimization process. A distribution (T) of transformation functions t is used between the input x', controlled by the attacker and $t(x')$ which is the actual input to the model. By taking random transformations into account in the perturbation calculation, the adversarial image becomes resistant to these changes in the physical world [28]. This changes the above optimization problem to eq.4. where the expectation of the log probability of the adversarial label $y_t$is calculated.

$$\underset{x'}{\operatorname{argmax}}\, \mathbb{E}_{t\sim T}[log\ P(y_t|t(x'))] \qquad (eq.4)$$

Furthermore, instead of the L_∞ norm bound the distance of the two inputs is constrained by considering the transformations as eq. 5:

$$\delta = \mathbb{E}_{t\sim T}\big[dist\big(t(x'), t(x)\big)\big] \qquad (eq.5)$$

The bound uses $t(x')$ instead of x' since the important part is to limit the *Expected* effective distance of the perceived input to the model.

3.2.2.1 Adversarial Patch

In the traffic sign scenario, a variation of the EoT attack generated by Google research will be examined, which is ideal for an adversary to influence a self-driving system. An adversarial patch is a sticker that can be applied on the surface of traffic signs to cause a misclassification. The patch is trained to optimize eq. 6:

$$\hat{p} = \underset{x'}{\operatorname{argmax}}\, \mathbb{E}_{x\sim X, t\sim T, l\sim L}[log\ P(y_t|A(x', l, t)] \quad (eq. 6)$$

The attack A for the training data X does not only consider a transformation distribution T, but also a distribution over the locations in the image L. This expectation is over multiple images, which implies that the patch is a universal attack that works on multiple backgrounds.

From Figure 13 it can be inferred that the effectiveness of the attack occurs due to the inherent weakness of classification networks to assign one value on each image. The patch operates by creating a region which causes the filters to focus on it and ignore other features on the image [27]. Figure 13b shows that a simple sticker well positioned on a traffic sign can completely misclassify it with a high probability (91.39%).

The adversarial patch area can be further decreased based on the boundary of each sign. Figure 13c shows that a sticker which takes only 0.5% of the total figure area is enough to misclassify traffic signs with a 96.4% confidence. Since all speed limit signs share common features such as shape and colours, a smaller change is enough to change the classification.

13a ***13b*** ***13c***

Figure 13: *Adversarial patch generation.13a: list of stickers generated to misclassify traffic signs. 13b: misclassified as ‘ no entry‘ sign (91.3%). 13c: misclassified as’ speed limit 80’ sign (96.4%).*

3.2.2.2 Patch Transferability

The adversarial patch paper [27] includes patch images that have the potential for transferability on specific models. To compare the effectiveness with the ones algorithmically generated they are also examined as stickers on traffic signs (Figure 14). Since the stickers were generated on a different dataset and model, they require a significantly larger area to cause a misclassification.

14a ***14b*** ***14c***

Figure 14: *Transferability of camouflaged adversarial patches. 14a: misclassified as 'turn left ahead' sign (56.44%). 14b: misclassified as 'turn left ahead' sign (66.43%). 14c: misclassified as 'no entry' sign (67.19%)*

The adversarial patch is a highly dangerous attack, due to its robustness and ability to cause an error in the model from multiple distances and angles. It is also able to make confident false positive predictions (Figure 13) and especially in its camouflaged form (Figure 14), it would pose a serious security threat to the future of self-driving vehicles. It is also vital to consider that the patch can generate a targeted attack. This allows adversaries to generate misclassifications with dangerous consequences.

3.3 Adversarial Defences

3.3.1 Adversarial Training

The ingenuity of machine learning and more specifically deep learning frameworks is that based on the provided training data the algorithm can learn new and different features. Adversarial training is a defence method which allows the defender to mix perturbated images in the training data. In theory, the network can learn the features of adversarial inputs and decrease the misclassification caused by the attack. Adversarial training is an extremely effective defence method, but only for the specific attacks, that it is trained against.

Figure 15 displays the two methods of adversarial training that are considered for defence. The first method trains the existing model with only adversarial data. Each input is randomly assigned one of the three previous graffiti attacks and then is used for training. The second method only attacks 1 out of 5 data points. This ensures the model does not overfit in the adversarial data and lose its accuracy in the original images. Both methods perform exceptionally well and within 2 epochs can predict all the graffiti attacks and original test data with 94.22% accuracy for one on one method and 88.71% for one on five.

In the case where multiple different graffiti attacks were considered, the network would require a larger number of iterations to achieve satisfactory accuracy. In such a case, training with only adversarial data would be less effective since the original accuracy would decrease. This is further reinforced from the fact that after the third iteration the accuracy of one on one training is falling.

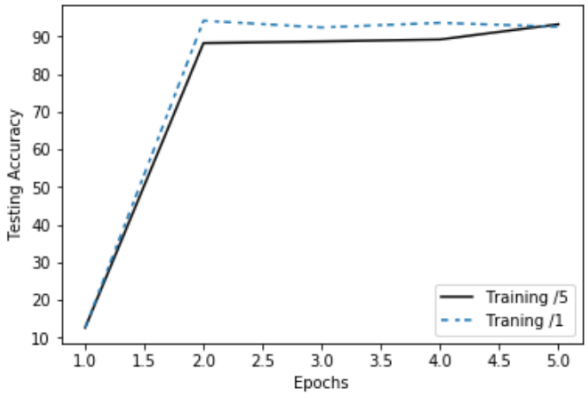


Figure 15: *Testing accuracy of adversarial training with only adversarial examples, one on one perturbation (/1) and with one out of five (/5).*

3.3.2 Defensive Distillation

Defensive distillation is a label smoothing method that takes advantage of the final SoftMax layer of neural networks to improve the classification of adversarial inputs. The function is used to assign probability (confidence) values to multiple classes [18]. The revised SoftMax equation 7 calculates the probability vector predictions F(X) by normalising the vector of outputs Z(X) that was produced in the last hidden layer. The temperature (T) determines the level of confidence in the predictions. It is usually set to 1 to make definite (high) predictions, which is why eq.2 in Chapter 2, which is used for discriminative confidence calculations, did not include the temperature variable.

$$F(X) = \frac{e^{\frac{z_i(X)}{T}}}{\sum_{l=0}^{N-1} e^{\frac{z_l(X)}{T}}} \qquad (eq.7)$$

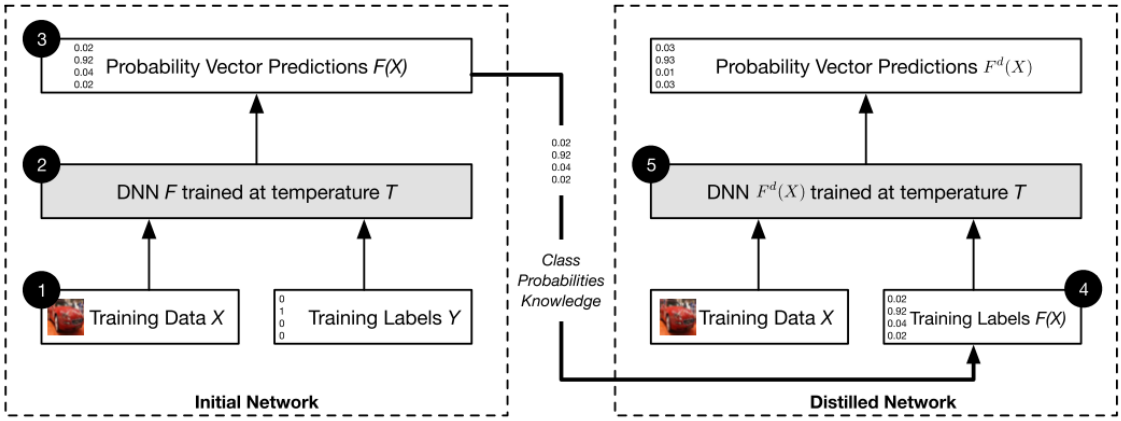


Figure 16: *Defensive Distillation method showing the initial and distilled Networks. Reproduced from [33].*

Defensive distillation is achieved by changing the temperature of the SoftMax function to a large value (model trained with 20). The network will output a set of probabilities F(X) describing the input image instead of a discrete class value. As shown in Figure 16, the probability vector will then be used as the training labels for the distilled network. In the final layer, a temperature value of 1 will be used to gain a confident prediction for the class label [33].

Defensive distillation mitigates adversarial perturbations by smoothing the decision boundary. Adversarial examples directed towards finding the minimum perturbation can easily be mitigated by widening the boundary.

When calculating the weights, the gradient of the cross-entropy loss function E(w) is minimised [10]. Equation 8 calculates the gradient for an activation function $a_n = w^T \varphi_n$. Because the input labels (vector of probabilities) of the distilled network are not discrete, a wrong prediction (p_n), will not have a zero value and can still contribute to the decision process.

$$\nabla E(w) = \sum_{n=1}^{N} (F_n(X) - p_n) \times \varphi_n \qquad (eq. 8)$$

Figure 17 displays the training process of the defensive distillation models that was used in the project. The outputs from the first model and second model are show the impact of the temperature in SoftMax to control the confidence of probabilities. Both models are ResNet-34 and but have different temperature values. In order to code the change in temperature, custom cross entropy loss functions were generated and the SoftMax function was manually changed.

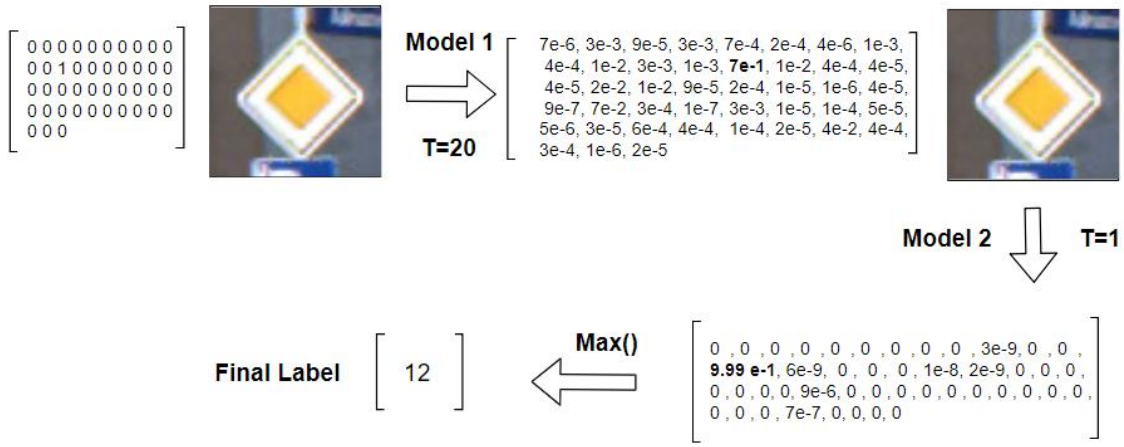


Figure 17*:The defensive distillation training process for a traffic sign of class 12. Temperature value is 20 and 1 respectively. Diagram was created using draw.io.*

Defensive Distillation is tested against the expectation over transformation adversarial patch. EoT is a very difficult attack to counteract because it is universal for all images and can thus extend the adversarial value much further than the boundary of the original y with the adversarial label y_t. Defensive distillation was not able to successfully predict most of the larger size adversarial patches. However, it was successfully able to defend against small patches of size 0.5% of the total image area (Figure 13c). These stickers were trained with few images "speed limit" signs and the perturbations were kept to a minimum close to the boundaries. Defensive distillation was able to predict with high accuracy (over 90%) these attacks, which shows that label smoothing is an effective technique, but is limited on the magnitude of the attack.

Chapter 4

Facial Recognition

4.1 Facial Recognition Network

Facial recognition (FR) has been increasingly relevant in multiple societies across the world. It is widely considered a harder problem to approach than the classification of street signs. FR requires the network to learn numerous discriminative features and perform well in open-data testing. In recent years, there have been larger and more complex datasets which are a necessity for such a task. The facial recognition network considered will be a 20-layer SphereFace which is pre-trained in PyTorch on the VGGFace2 Dataset. Facial recognition revolves around clustering unknown individuals to the network based on the similarity of the vectors. The LFW pair benchmark will be used for open testing and to evaluate the accuracy of the model.

4.2 Face Datasets

4.2.1 VGGFace2

VGGFace2 is a large dataset with 3.31 million images and 9131 different people. The dataset is unique because it has an average of 362.6 images per subject making it ideal to train a network that identifies similarity between faces [44]. The images are downloaded from google search and are different in age, illumination, ethnicity and even pose. This makes the networks trained on this dataset able to generalise on different faces. The dataset includes a test set, but it will be evaluated using the LFW Benchmark which is a standard in the field of facial recognition.

Figure 18: *Images found in the VGGFace2 Dataset displaying the variety of different people in the data.*

4.2.2 Labelled Faces in the Wild

The Labelled Faces in the Wild (LFW) Dataset was one of the first large scale datasets collected for unrestrained facial recognition. It includes more than 13,000 images with 5749 different people. The LFW pair benchmark includes a list of 6000 pairs which can be used to evaluate a recognition network. Each pair can be the same person and have a label of 1 or two different people (label=0). The network uses a distance method such as cosine similarity and calculates the threshold to achieve the best accuracy on the test [45].

19a ***19b***

Figure 19: *Different pairs included in the LFW pair Benchmark. 19a: Same person pair with label 1 (Matt Damon), 19b: Different person pair with label 0 (Debra Yang, Yekaterina Guseva)*

4.3 Data Pre-processing

The images considered above include people in different backgrounds, poses and illuminations. The first step in the data is to extract the faces from the images using the pretrained face detection network MTCNN. Figure 21 shows the different steps of the detection network in detecting a face from LFW. Only the face identified in the final network is inputted in the recognition network. Afterwards, every cropped face is at the same size (3,112,96) and the images are normalized by subtracting 127.5 and diving 128 from the pixel values [0 , 255] .

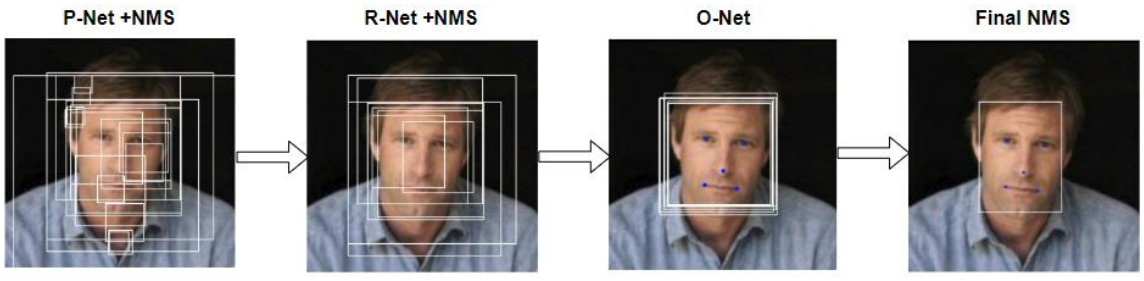


Figure 20: *Facial Detection output of different networks in MTCNN on the Aaron Eckhart image from LFW to minimize background interference.*

The cosine similarity results from the LFW pair benchmark are inputted in a loop that calculates a range of accuracies for different thresholds and selects the threshold with the highest accuracy. The threshold is the value which separates the classifications of cos similarity: same image (above threshold) and different images (below threshold). The SphereFace implementation has an LFW pair accuracy of 99.22% with an optimal threshold of 0.305.

4.4 Adversarial Attacks

4.4.1 Additive Noise

4.4.1.1 Gaussian Noise

Gaussian (or electronic) noise is often found in amplifiers and detectors. It is caused by natural resources such as thermal vibration [46]. It is expressed by the Probability Density function (PDF) with respect to the grey level g, where μ is the mean and σ the standard deviation [47].

$$P(g) = \sqrt{\frac{e^{\frac{-(g-\mu)^2}{2\sigma^2}}}{2\pi\sigma^2}} \qquad (eq.9)$$

Gaussian noise is widely used to represent an approximation of a practical noise scenario that is likely to occur in real life. As seen in Figure 21, this attack is especially harmful because it is classified as an indistinguishable perturbation, making it undetectable by a human observer. The attack is implemented using the Foolbox library, in a decision-based scenario. This implies that the mean and standard deviation are calculated to express the minimum possible noise that decreases the confidence below a threshold. It is also important to consider the impact on noise in the face detection networks. A noisy image generates a heightened uncertainty which in turn increases the number of bounding boxes in each neural network of the MTCNN. Because of the robust training of the detection network (MTCNN) on low quality and perturbated images, it is still able to detect the face accurately but with a decreased confidence.

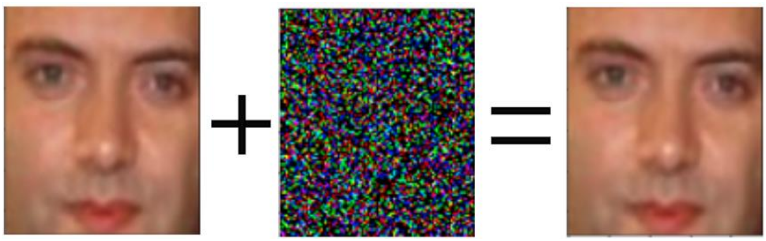


Figure 21: *Gaussian Noise in LFW. Even though the noise is added across the entire image it is not visible by a human observer.*

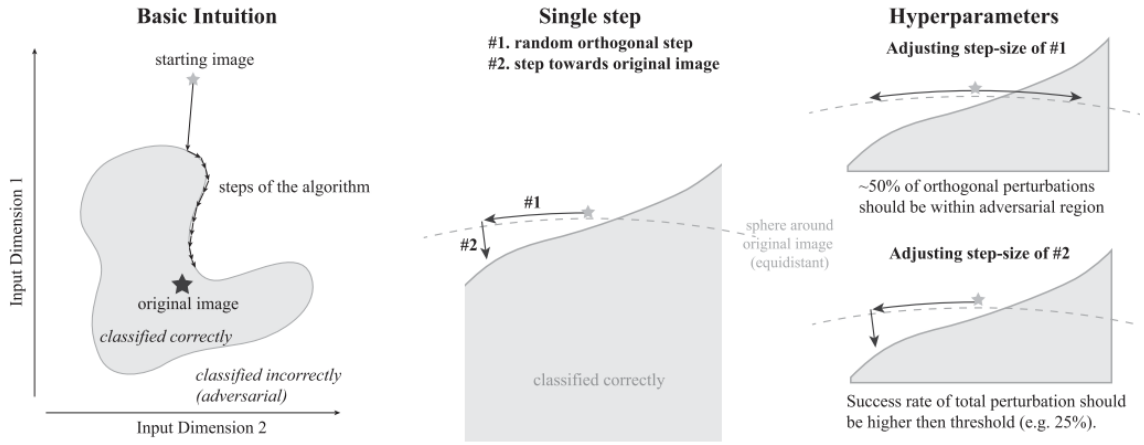
4.4.1.2 Salt and Pepper

Salt and Pepper (Impulse valued) noise does not fully corrupt the image but only specific pixel values [48]. The noise commonly occurs in telecommunications and data transmission were specific pixels erroneously receive the maximum or minimum value (255 or 0 respectively). The attack is generated by adding noise in random positions across the image. Because the network is not trained for classification but recognition, the performance of the attack is measured using the LFW pair benchmark and is able to decrease the total accuracy to 85.56%.

***Figure 22**:Salt and Pepper additive noise in LFW input image. Noise is generated using random pixel locations and setting them to min or max.*

4.4.2 Boundary Attack

Boundary attack is one the most powerful decision based, black-box adversarial examples. The attack starts with an adversarial image x' that is already misclassified (adversarial). Afterwards the goal of the algorithm as shown in Figure 23a is to approach as much as possible to the boundary of the original image while remaining adversarial [49]. This will ensure that the final boundary attack does not significantly change the perturbated image (Figure 24).



23a ***23b*** ***23c***

***Figure 23:**The operation of the Boundary Attack. Reproduced from [48]*

Figure 23b shows the adversarial change in each algorithmic iteration, which includes one random orthogonal move along the sphere projection of the original image, and one step towards the target image. To ensure that the attack finds the minimum distance towards the original image, the hyperparameters (#1 orthogonal, #2 vertical) are adjusted based on the geometry of the boundary (Figure 23c).

The boundary attack does not require knowledge of the model's parameters and it is thus classified as a black-box attack with probing. The magnitude of the attack is always different, as it relates to the relative distance of the adversarial and target class. One limitation of the attack is that it needs to iteratively enter the network to approximate the boundary and direction of the attack which can be slow for targeted misclassification. The attack is generated by setting a random class target to each input and using the iterative approach of the attack to approximate the boundary. The adversarial example is evaluated on the LFW benchmark and can lower the total accuracy of the model to 23.53% with a fixed threshold.

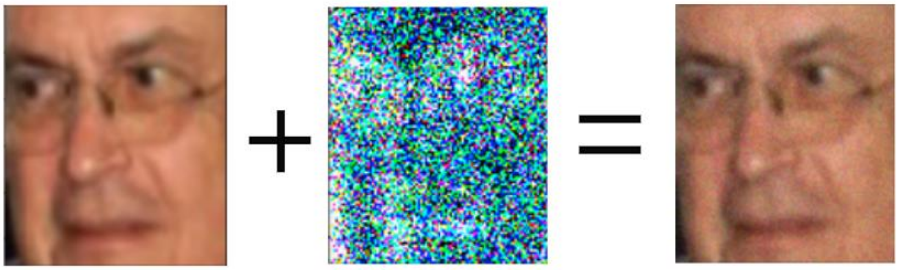


Figure 24: *Boundary attack until successful misclassification on the LFW Abel_Pacheco_0001 image.*

4.5 Adversarial Defences

4.5.1 Denoising Models

4.5.1.1 Gaussian smoothing

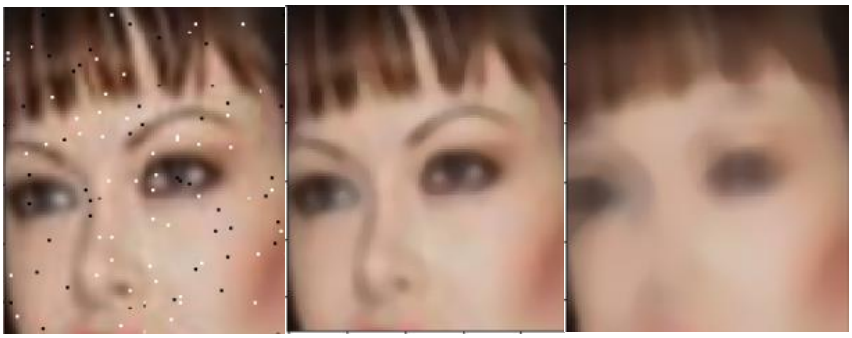
One of the most popular filters in computer graphics is the gaussian blur. It is specifically designed to mitigate noise in the form of a Gaussian distribution. The method is commonly used in edge detection since it operates as a low pass filter and can retain sharp edges in an image. Smoothing can vary according to the standard deviations of the distribution. Figure 25b shows the result of the image with standard deviations in the X and Y directions equal to 7. Figure 25c shows the difference between the adversarial image and the original after the denoising method. The distinct facial characteristics imply that most of the noise has been neutralized and the image can be used for feature extraction. A large smoothing (Figure 25d) can mitigate higher levels of noise, but at the same time give a worse result since the network will not be able to detect the main features that allow recognition between faces [50].

25a ***25b*** ***25c*** ***25d***

Figure 25: *Gaussian Blur as a defence method to prevent additive gaussian noise. 25a: Image with gaussian noise. 25b:Gaussian blur using a 7x7 standard deviations. 25c:Difference between original and blurred image. 25d: Gaussian blur using a 27x27 standard deviations.*

4.5.1.2 Median Filtering

A median filter is an image processing method which slides a window area across the image. Each time, the pixel values inside the window are replaced with the median of the entire area. This technique ensures the smoothing of the pixels while preserving the sharp edges of the image. Spiky noise and especially salt and pepper are extremely susceptible to this method. Figure 26 shows that the impulse noise has been entirely negated ,but the trade-off was a significant loss in the resolution of the image. When removing large amounts of noise from an image, it is likely that the defence method will also remove discriminative features and lower the accuracy of the model (Figure 26c). Using the OpenCV library, a median blur is applied to the salt and pepper noise, and it can increase the accuracy of the LFW pair benchmark to 88.78%.



26a ***26b*** ***26c***

Figure 26: *Impact of median filtering on an image. 26a: Salt-Pepper noise on image. 26b: Median smoothing on 50% noise 26c: Median smoothing on 75% noise*

Denoising models have a lot of limitations as a defence method but are constantly used in computer vision because they are extremely effective in noise reduction. Gaussian blur was able to directly extract the features of the image and median blur was similarly capable of increasing the accuracy of the benchmark. Based on the success of these techniques, any real-world model would and especially security cameras would greatly benefit and enhance their robustness.

4.5.2 Input Transformations

Input transformations is a collection of transformation functions, that are combined to prevent adversarial perturbations from influencing a network. It is a testing defensive method that manipulates the images before they are inserted in the network. Two main transformations are examined: JPEG compression and bit-depth reduction. JPEG compression can remove small perturbations by decreasing the image quality. Similarly, bit-depth reduction operates a quantization in the pixels (Figure 27d), which is specifically targeted towards small perturbations. By degrading the image quality using these techniques, real-world transformations are applied to the data [40]. The boundary attack is a technique that aims at finding minimal perturbations (closest to the boundary). The transformations can influence the adversarial examples and significantly impact the performance of the boundary attack. Towards small boundary perturbations, the accuracy is increased to 64.92%. On the other hand, an adversary can use targeted boundary attacks, which take significantly longer to iterate, but can have a wider classification margin. In these cases, it is difficult to achieve an image quality that reduces perturbations and ensures sufficient quality for feature extraction.

27a ***27b*** ***27c*** ***27d***

Figure 27: *Input transformations to mitigate the effects of a boundary attack. 27a: Boundary attack on LFW image. 27b: Reducing JPEG compression to 10%. 27c: Reducing bit-depth to 3. 27d: Reducing bit-depth to 1*

Chapter 5
Conclusion and Future Work

5.1 Results Interpretation

The deep learning models considered can achieve high accuracies and even detect a wide variety of data, such as different angles and illuminations. However, adversarial examples are shown to be effective even in real-life models with multiple limitations such as traffic sign classification and facial recognition. The graffiti attacks proved to be effective in misclassifying images and decreased the accuracy of the network significantly. On the other hand, adversarial training was able to partially mitigate the attacks by combing the in original training data some adversarial samples. The EoT attack was not completely mitigated by defensive distillation because of a universal perturbation includes adversarial samples that are far away from the boundary. Facial recognition was investigated using noise functions which managed to impact the accuracy with only small perturbations. The denoising models were afterwards able to partially reconstruct the images, which resulted in improved accuracy for the model. Lastly, the boundary attack was highly successful and decreased the accuracy of the model in an iterative black-box scenario, but its impact was minimized using input transformation techniques.

In conclusion, even though there are ways to significantly mitigate the impact of adversarial examples, especially physical-layer attacks are difficult to predict. There are still many dangers and limitations with adversarial research since the neural networks are only able to compare based on the training data and lack the ability to understand the complexities of the real world, that humans are easily accustomed to. Nevertheless, adversarial research is a step in the right direction in significantly decreasing accidents and protecting security-critical systems.

5.2 Future Work

With the ever-increasing automation in modern society, adversarial research will become increasingly more relevant in multiple areas were security and robustness of the networks is crucial. In self-driving cars, one of the most difficult issues is out-of-distribution attacks on traffic sign detection. These attacks can confuse the network which identifies signs in images where they do not exist. This problem can be especially tricky when considering reflections on surfaces, which might require a network to not only focus on the region of interest (area of traffic sign), but also evaluate if the background increases the possibility of a sign. Furthermore, attacks such as the adversarial patch, which is resistant to transformations in the real world, pose significant threats when disguised as peace sign stickers which can pass undetected from a human observer. A possible countermeasure would be a well-trained GAN (Generative Adversarial Network) detector that is able to remove any excess stickers and recreate the image [37].

In facial recognition, the major issue is additive noise and low image quality. After using a denoiser function, a possible solution would be a super-resolution network. Such a network can predict the value of pixels and significantly enhance the image quality. This would imply that facial features are better extracted and mitigate the impact of noise. On the other hand, a boundary attack would be much harder to counter. A possible solution could be defensive distillation, but it would require the network to overcome the limitations of facial recognition. Defensive distillation needs to train two networks, which would take tremendous amount of time and power considering the sizes of facial recognition datasets. Moreover, the datasets contain thousands of different subjects and are tested on completely different people. This would probably backfire on the label smoothing part of defensive distillation and result in an indecisive network. A better solution would likely be PixelDefend which uses generative models to possible attacks and purifies the image by moving it backwards towards the distribution that might have caused the attack [51].

There are many different attack and defence types, each with its strengths and weaknesses. To ensure in the future robust security-critical systems, it is important to understand the risks and impacts of common attacks that are likely to occur and set system specific defences that are able to mitigate and more importantly, take advantage of the big data and deep learning techniques to learn different attack methods when they are encountered.

Chapter 6
Project Management

The project included a lot of background research in order to decide on the specific scenarios, attacks and defences that would be applicable for a real-world investigation. Most of the research was conducted in the first semester, but the attacks and defences were not finalised until March. Each method had its own challenging problems in the programming implementation and some also in data collection. This meant constantly adjusting the schedule according to the technical progress. Moreover, a useful step in the project was writing the report in parallel which was especially useful in setting clear targets and observing exactly what was the progress until that point. Appendix A includes the Gantt charts and risk mitigation strategies which were used to ensure the project was successfully completed in time.

Word Count: 9306

The word count was calculated including headings and figure labels. Only the sections marked as chapters in the report are considered in the counting.

Bibliography

[1] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards Deep Learning Models Resistant to Adversarial Attacks,” Jun. 2017.

[2] C. Szegedy *et al.*, “Intriguing properties of neural networks,” Dec. 2013.

[3] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and Harnessing Adversarial Examples,” Dec. 2014.

[4] A. Kurakin *et al.*, “Adversarial Attacks and Defences Competition,” Mar. 2018.

[5] K. Eykholt *et al.*, “Robust Physical-World Attacks on Deep Learning Models,” Jul. 2017.

[6] J. Gilmer, R. P. Adams, I. Goodfellow, D. Andersen, and G. E. Dahl, “Motivating the Rules of the Game for Adversarial Example Research,” 2018.

[7] L. Huang, A. D. Joseph, B. Nelson, B. I. P. Rubinstein, and J. D. Tygar, *Adversarial Machine Learning* *. 2011.

[8] A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial examples in the physical world,” Jul. 2016.

[9] A. Krizhevsky and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” *Neural Inf. Process. Syst.*, 2012.

[10] Christopher M. Bishop, *Pattern Recognition and Machine Learning*. Springer.

[11] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. OReilly, 2017.

[12] K. He, X. Zhang, S. Ren, and J. Sun, “Identity mappings in deep residual networks,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9908 LNCS, pp. 630–645, 2016.

[13] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” Dec. 2015.

[14] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, “Aggregated residual transformations for deep neural networks,” *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017–Janua, pp. 5987–5995, 2017.

[15] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song, “SphereFace: Deep Hypersphere Embedding for Face Recognition.”

[16] and Y. Q. Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, Senior Member, IEEE, “Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks,” 2015.

[17] Y. He, C. Zhu, J. Wang, M. Savvides, and X. Zhang, “Bounding Box Regression with Uncertainty for Accurate Object Detection,” Sep. 2018.

[18] P. Reverdy and N. E. Leonard, “Parameter Estimation in Softmax Decision-Making Models With Linear Objective Functions,” *IEEE Trans. Autom. Sci. Eng.*, vol. 13, no. 1, 2016.

[19] F. Rahutomo, T. Kitasuka, and M. Aritsugi, “Semantic Cosine Similarity.”

[20] Pratap Dangeti, *Statistics for Machine Learning*, 1st ed. Birmingham: Packt

Publishing Ltd., 2017.

[21] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar, “Can Machine Learning Be Secure?,” 2006.

[22] Y. Liu, X. Chen, C. Liu, and D. Song, “Delving into Transferable Adversarial Examples and Black-box Attacks,” no. 2, pp. 1–24, 2016.

[23] S. Clark, “Elephant in the room,” *New Sci.*, vol. 230, no. 3074, pp. 29–31, 2016.

[24] K. Eykholt *et al.*, “Physical Adversarial Examples for Object Detectors,” Jul. 2018.

[25] J. Lu, H. Sibai, E. Fabry, and D. Forsyth, “NO Need to Worry about Adversarial Examples in Object Detection in Autonomous Vehicles,” Jul. 2017.

[26] C. Sitawarin, A. N. Bhagoji, A. Mosenia, M. Chiang, and P. Mittal, “DARTS: Deceiving Autonomous Cars with Toxic Signs,” Feb. 2018.

[27] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer, “Adversarial Patch,” Dec. 2017.

[28] A. Athalye, L. Engstrom, A. Ilyas, and K. Kwok, “Synthesizing Robust Adversarial Examples,” Jul. 2017.

[29] Y. Wen, K. Zhang, Z. Li, and Y. Qiao, “A Discriminative Feature Learning Approach for Deep Face Recognition.”

[30] W. He, J. Wei, X. Chen, N. Carlini, and D. Song, “Adversarial Example Defenses: Ensembles of Weak Defenses are not Strong,” Jun. 2017.

[31] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, and D. Mukhopadhyay, “Adversarial Attacks and Defences: A Survey,” Sep. 2018.

[32] F. Croce, M. Andriushchenko, and M. Hein, “Provable Robustness of ReLU networks via Maximization of Linear Regions,” Oct. 2018.

[33] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, “Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks,” in *Proceedings - 2016 IEEE Symposium on Security and Privacy, SP 2016*, 2016, pp. 582–597.

[34] L. Schott, J. Rauber, M. Bethge, and W. Brendel, “Towards the first adversarially robust neural network model on MNIST,” May 2018.

[35] N. Carlini and D. Wagner, “Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods,” May 2017.

[36] D. Meng and H. Chen, “MagNet: a Two-Pronged Defense against Adversarial Examples,” May 2017.

[37] B. Hitaj, G. Ateniese, and F. Perez-Cruz, “Deep Models Under the GAN: Information Leakage from Collaborative Deep Learning,” Feb. 2017.

[38] C. Xie, J. Wang, Z. Zhang, Z. Ren, and A. Yuille, “Mitigating Adversarial Effects Through Randomization,” Nov. 2017.

[39] W. Xu, D. Evans, and Y. Qi, “Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks,” Apr. 2017.

[40] C. Guo, M. Rana, M. Cisse, and L. van der Maaten, “Countering Adversarial Images using Input Transformations,” Oct. 2017.

[41] D. Pedamonti, “Comparison of non-linear activation functions for deep neural

networks on MNIST classification task."

[42] B. Luo, Y. Liu, L. Wei, and Q. Xu, "Towards Imperceptible and Robust Adversarial Example Attacks against Neural Networks."

[43] Y. Dong *et al.*, "Boosting Adversarial Attacks with Momentum," Oct. 2017.

[44] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman, *VGGFace2: A dataset for recognising faces across pose and age*. .

[45] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments."

[46] A. Kumar Boyat and B. Kumar Joshi, "A REVIEW PAPER: NOISE MODELS IN DIGITAL IMAGE PROCESSING," *An Int. J.*, vol. 6, no. 2, 2015.

[47] A. D. W. Robert N. McDonough, *Detection of Signals in Noise*, 2nd ed. London: Academic Press Limited, 1971.

[48] A. Joshi, A. K. Boyat, and B. K. Joshi, "Impact of Wavelet Transform and Median Filtering on removal of Salt and Pepper Noise in Digital Images," in *2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*, 2014, pp. 838–843.

[49] W. Brendel, J. Rauber, and M. Bethge, "Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models," Dec. 2017.

[50] T. S. Huang, *Two-Dimensional Digital Signal Prcessing II*. Berlin: Springer Berlin Heidelberg, 2014.

[51] Y. Song, T. Kim, S. Nowozin, S. Ermon, and N. Kushman, "PixelDefend: Leveraging Generative Models to Understand and Defend against Adversarial Examples," Oct. 2017.

Appendix A: Project Management

A.1 Gantt Charts

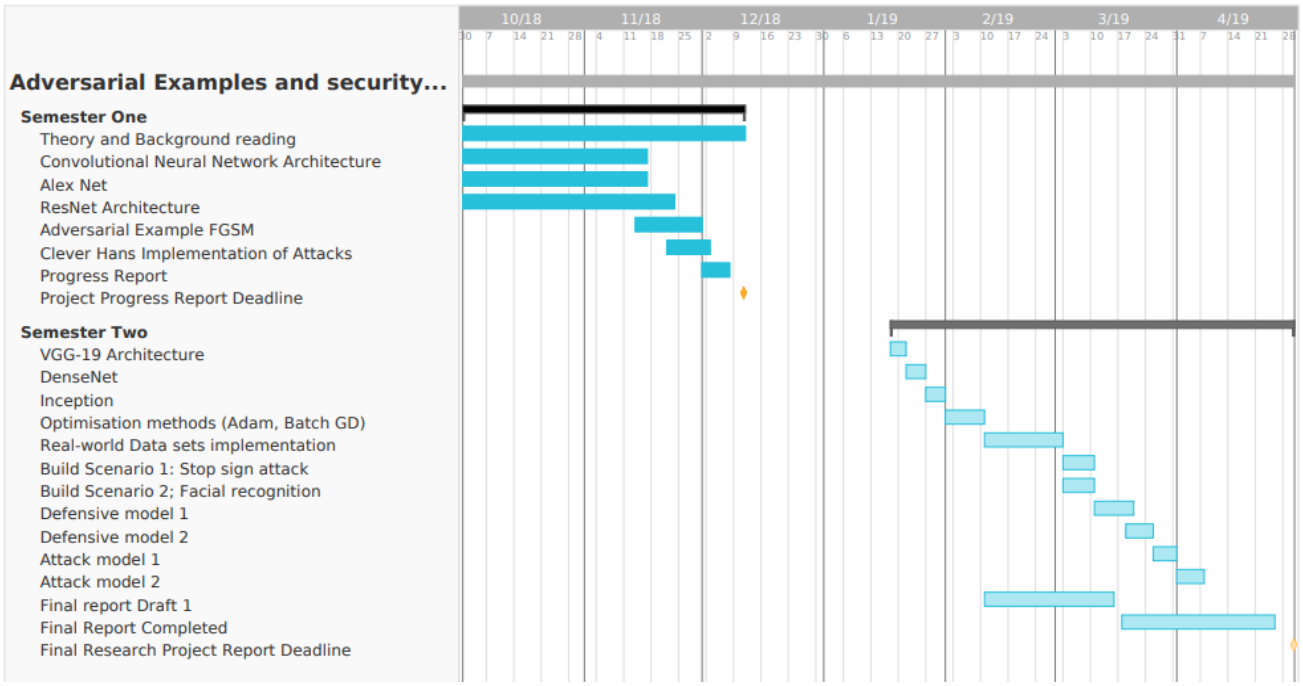


Figure A 2:*Initial Gantt chart for project management in semester 1. Created using Teamgantt.com service.*

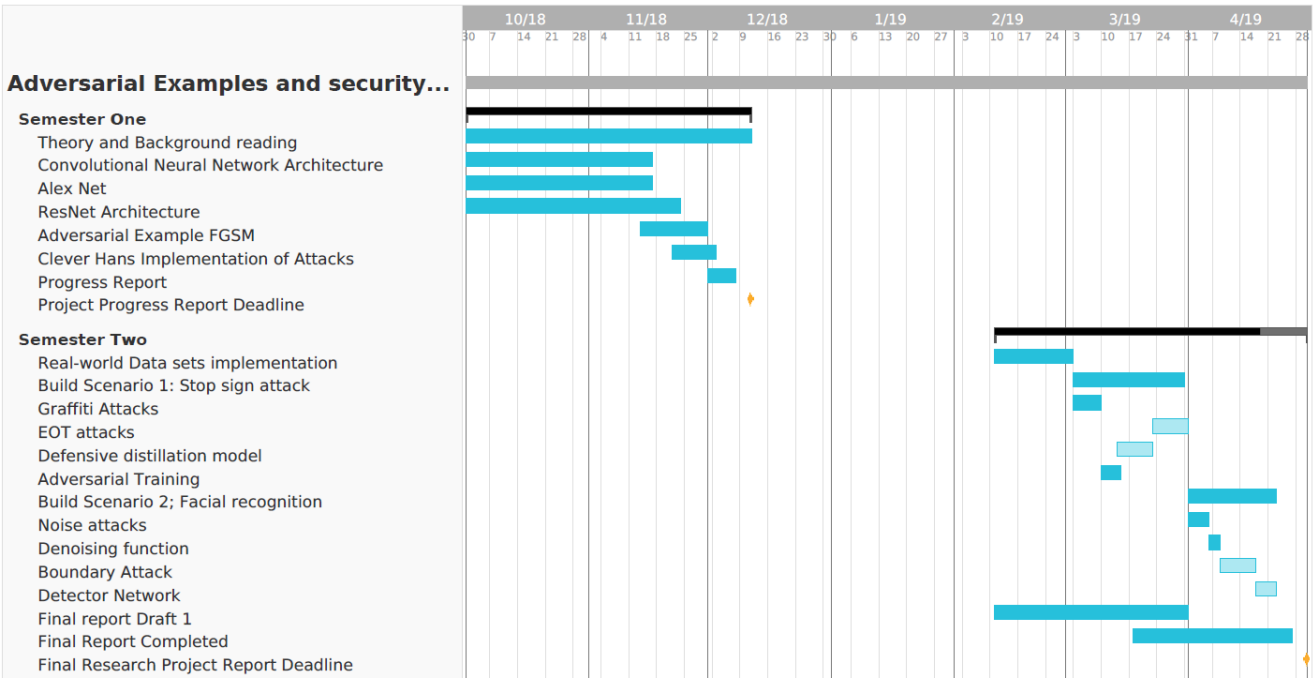


Figure A 1:*Revised Gantt chart for project management in semester 2. Created using Teamgantt.com service.*

A.2 Risk Mitigation

Risk Management	Probability score	Impact score	Mitigation strategies
Technical Difficulties	4/5	4/5	Regular meetings with supervisor, spend a lot of time in semester 1 studying theory
Programming Difficulties	5/5	3/5	Ability to ask PhD students for assistance, flexible schedule
Possibility to go off topic	3/5	2/5	Regular meetings with supervisor, writing progress in parallel with code
Loss of work	1/5	5/5	Regular saves on usb and cloud
Different versions of software required	3/5	3/5	Using dual boot of Linux and windows, and virtual environment
Supervisor unavailable when needed	2/5	3/5	Weekly meetings and collection of queries
Illness or inability to work	2/5	3/5	Flexible scheduling and Gantt chart

Appendix B: GTRSB Distribution

Label count 2250	Label: 2 Speed limit (50km/h)
Label count 2220	Label: 1 Speed limit (30km/h)
Label count 2160	Label: 13 Yield
Label count 2100	Label: 12 Priority road
Label count 2070	Label: 38 Keep right
Label count 2010	Label: 10 No passing for vechiles over 3.5 metric tons
Label count 1980	Label: 4 Speed limit (70km/h)
Label count 1860	Label: 5 Speed limit (80km/h)
Label count 1500	Label: 25 Road work
Label count 1470	Label: 9 No passing
Label count 1440	Label: 7 Speed limit (100km/h)
Label count 1410	Label: 3 Speed limit (60km/h)
Label count 1410	Label: 8 Speed limit (120km/h)
Label count 1320	Label: 11 Right-of-way at the next intersection
Label count 1200	Label: 18 General caution
Label count 1200	Label: 35 Ahead only
Label count 1110	Label: 17 No entry
Label count 780	Label: 14 Stop
Label count 780	Label: 31 Wild animals crossing
Label count 689	Label: 33 Turn right ahead
Label count 630	Label: 15 No vechiles
Label count 600	Label: 26 Traffic signals
Label count 540	Label: 28 Children crossing
Label count 510	Label: 23 Slippery road
Label count 450	Label: 30 Beware of ice/snow
Label count 420	Label: 6 End of speed limit (80km/h)
Label count 420	Label: 16 Vechiles over 3.5 metric tons prohibited
Label count 420	Label: 34 Turn left ahead
Label count 390	Label: 22 Bumpy road
Label count 390	Label: 36 Go straight or right
Label count 360	Label: 20 Dangerous curve to the right
Label count 360	Label: 40 Roundabout mandatory
Label count 330	Label: 21 Double curve
Label count 300	Label: 39 Keep left
Label count 270	Label: 24 Road narrows on the right
Label count 270	Label: 29 Bicycles crossing
Label count 240	Label: 27 Pedestrians
Label count 240	Label: 32 End of all speed and passing limits
Label count 240	Label: 41 End of no passing
Label count 240	Label: 42 End of no passing by vechiles over 3.5 metric tons
Label count 210	Label: 0 Speed limit (20km/h)
Label count 210	Label: 19 Dangerous curve to the left
Label count 210	Label: 37 Go straight or left

Appendix C: File Navigation

Files are separated on two different folders based on the two different security-critical systems investigated:

- **Street Sign Classification:** Folder for scenario 1
    - **Classification Network.py:** Training and Testing of ResNet 34
    - **Graffiti Attacks and Adversarial Training.py:** Generation of graffiti attacks and adversarial training defence method
    - **DD Network 1 and Label Gen.py:** Defensive Distillation network 1 which generates label smoothing
    - **DD Network 2 and Testing.py:** Defensive Distillation network 1 which is used to classify images
    - **Adversarial Testing.py:** Testing of adversarial trained networks to compute the accuracies
    - **Collect and Convert images.py:** script used to collect and change the format of images for adversarial training
    - **EOT Patch.py:** Generation of Adversarial Patch based on Github respiratory: github.com/jhayes14/adversarial-patch

- **Facial Recognition:** Folder for scenario 2
    - **LFW Benchmark Evaluation.py:** Evaluation of LFW pairs based on Github respiratory: github.com/clcarwin/sphereface_pytorch
    - **Adversarial Attack Generation and Denoising.py:** Generation of gaussian noise, salt-pepper noise and denoising functions
    - **Boundary Attack and Transformation input.py:** Generation of boundary attacks and inputs transformations
    - **Bounding Boxes and Image Resizing.py:** Facial recognition using the pretrained MTCNN for face detection and pre-processing

Appendix D: Selective Listings

D.1 Street Sign Classifier

D.1.1 Data Pre-processing

```
import random
s = np.random.choice((100), 5, replace=False)
print(s)
k1=s[:]
k2=s[:]
np.random.shuffle(k1)
np.random.shuffle(k2)
print(k1,k2)

temp=Counter(train_labels).most_common()
order=[[] for _ in range(43)]
for l,c in temp:
    order[int(l)]=c

val_images=[]
val_labels=[]
tr_labels=[]
tr_images=[]
n1,n2=0,0
ratio=0.2

print(order)

for n in order:
    n2+=n
    #print(n1,n2)
    val_labels.extend(train_labels[n1:int(n1+n*ratio)])
    tr_labels.extend(train_labels[int(n1+n*ratio):n2])
    val_images.extend(train_images[n1:int(n1+n*ratio)])
    tr_images.extend(train_images[int(n1+n*ratio):n2])
    n1=n2
#print(val_labels)

ts_images=test_images
ts_labels=test_labels

def shuffle(images,labels):
    s = np.random.choice(range(len(images)), len(images), replace=False)
    i=0
    temp_labels = labels
    temp_images = images
    for n in s:
        temp_labels[i]=labels[n]
        temp_images[i]=images[n]
        i+=1
    return temp_images,temp_labels

val_images,val_labels=shuffle(val_images,val_labels)
tr_images,tr_labels=shuffle(tr_images,tr_labels)
#ts_images,ts_labels=shuffle(ts_images,ts_labels)

num=int(np.random.choice(len(val_images), 1, replace=False))
fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(3,3))
plt.imshow(val_images[num])
plt.tick_params(axis=u'both', which=u'both',length=0)
```

```
plt.show()
print(val_labels[num],sign_names[int(val_labels[num])],'Number=',num)

av=0
for i in train_images:
    av+=i.shape[0]/i.shape[1]

for i in test_images:
    av+=i.shape[0]/i.shape[1]
av=av/(len(test_images)+len(train_images))
print('Average Ratio of Images',av)

#Training Processing

transform = transforms.Compose([transforms.Resize((112,112)),transforms.ToTensor(),transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
toPIL=torchvision.transforms.ToPILImage(mode=None)
print(tr_images[0].shape)
for i in range (len(tr_labels)):
    tr_images[i]=toPIL(tr_images[i])
    tr_images[i]=transform(tr_images[i])
    tr_images[i]=tr_images[i].unsqueeze_(0)
    print(i,end="\r")
print(tr_images[0].shape)

tr_labels=list(map(int, tr_labels))
tr_labels=torch.tensor(tr_labels)
print(tr_labels[:10])

tr_images=torch.cat(tr_images, dim=0, out=None)
print(tr_images.shape,tr_labels.shape)

batch_size=200
workers=4
tr=torch.utils.data.TensorDataset(tr_images,tr_labels)
load_tr=torch.utils.data.DataLoader(tr, batch_size=batch_size, num_workers=workers)
print(load_tr)

torch.save(tr_images, r'/home/konstantinoskk//Documents/linux_ip/Street/tr_images.pt')
torch.save(tr_labels, r'/home/konstantinoskk//Documents/linux_ip/Street/tr_labels.pt')
#torch.save(load_tr, r'/home/konstantinoskk//Documents/linux_ip/Street/load_tr.pt')


#Validation Processing

        print(val_images[0].shape)
        for i in range (len(val_labels)):
            val_images[i]=toPIL(val_images[i])
            val_images[i]=transform(val_images[i])
            val_images[i]=val_images[i].unsqueeze_(0)
            print(i,end="\r")
        print(val_images[0].shape)

        val_labels=list(map(int, val_labels))
        val_labels=torch.tensor(val_labels)
        print(val_labels[:10])

        val_images=torch.cat(val_images, dim=0, out=None)
        print(val_images.shape,val_labels.shape)

        val=torch.utils.data.TensorDataset(val_images,val_labels)
        load_val=torch.utils.data.DataLoader(val, batch_size=batch_size, num_workers=workers)
        print(load_val)
```

D.1.2 Classification Network Initialisation

```
modelf=torchvision.models.resnet34(pretrained=True)
for param in modelf.parameters():
    param.requires_grad = True
modelf.fc = nn.Linear(512, 43)
modelf.avgpool= nn.AdaptiveAvgPool2d(1)
print(modelf)
modelf=modelf.to(device)

epochs=30
total_step = len(tr_images)
print('Total step =',total_step,'Epochs =',epochs)

d=batch_size
step=(int((total_step/d)+1)*epochs)
print()
step_pe=int(step/epochs)
print('Number of steps per epoch' ,step_pe,'for batch =',d)

loss_list = np.zeros((step,1))
eloss=np.zeros((epochs,1))
tacc=[]
vacc=[]
vloss=np.zeros((epochs,1))
val_step = len(val_images)
test_step = len(ts_images)
```

D.1.3 Training the Network

```
def Net(model, criterion, optimizer, scheduler, num_epochs):
    start = time.time()
    tloss=0

    k,st=0,0
    vtemp=0

    for e in range(num_epochs):
        correct = 0
        total = 0
        for i,(inputs, labels) in enumerate(load_tr,0):

            optimizer.zero_grad()
            inputs = inputs.to(device)
            labels = labels.to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            tloss = loss.item()
            ac=labels.shape[0]
            st+=ac
            inputs = inputs.to('cpu')
            labels = labels.to('cpu')

            print('Epoch:',e+1,'/',epochs,'Step:',st,'/',total_step,'Step Loss:
  ',tloss, end="\r")
            loss_list[k,0]=tloss
            k+=1
            tloss=0

```

```
            if st==total_step:
                tac=(100 * correct / total)
                tacc.append(tac)
                correct = 0
                total = 0
                with torch.no_grad():
                    for inputs,labels in load_val:
                        inputs, labels = inputs.to(device), labels.to(device)
                        outputs = modelf(inputs)
                        _, predicted = torch.max(outputs.data, 1)
                        vtloss = criterion(outputs, labels)
                        vtloss = vtloss.item()
                        vloss[e,0]+=vtloss
                        total += labels.size(0)
                        correct += (predicted == labels).sum().item()
                    vac=(100 * correct / total)
                    vloss[e,0]=vloss[e,0]/(len(val_images)/d)
                    vacc.append(vac)
                if vac>=vtemp:
                    torch.save(modelf.state_dict(), r'/home/konstantinoskk//Documents/linux_ip/model/model.pth')
                    vtemp=vac
                    ef=e+1
        st=0
        pe=step_pe*(e+1)
        eloss[e,0]= np.mean(loss_list[(pe-step_pe):(pe),0])
        print('----------------','Epoch:',e+1,'Loss:',round(eloss[e,0],10),'Tac',tac,'Vac',vac,'----------------')

    time_elapsed = time.time() - start
    print('Training complete in {:.0f}m {:.0f}s'.format(time_elapsed // 60, time_elapsed % 60))
    print('Final Model produced at Epoch',ef)
    return model
```

D.1.4 Testing Phase

```
print(ts_images[0].shape)
for i in range (len(ts_labels)):
    ts_images[i]=toPIL(ts_images[i])
    ts_images[i]=transform(ts_images[i])
    ts_images[i]=ts_images[i].unsqueeze_(0)
    print(i,end="\r")
print(ts_images[0].shape)
ts_labels=list(map(int, ts_labels))
ts_labels=torch.tensor(ts_labels)
ts_images=torch.cat(ts_images, dim=0, out=None)
print(ts_images.shape,ts_labels.shape)

ts=torch.utils.data.TensorDataset(ts_images,ts_labels)
load_ts=torch.utils.data.DataLoader(ts, batch_size=batch_size, num_workers=workers)
print(load_ts)

correct = 0
total = 0
with torch.no_grad():
    for inputs,labels in load_ts:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = modelf(inputs)
        #print(inputs.shape)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        #print(labels,predicted,(predicted == labels).sum().item())
print('Accuracy : %d %%' % (100 * correct / total))
```

D.2 Graffiti Adversarial Attacks

D.2.1 Preload Trained Model

```
PATH = r'/home/konstantinoskk//Documents/linux_ip/model/model94.pth'
modelf=torchvision.models.resnet34(pretrained=True)
modelf.fc = nn.Linear(512, 43)
modelf.avgpool= nn.AdaptiveAvgPool2d(1)
modelf.load_state_dict(torch.load(PATH))
modelf=modelf.to(device)
```

D.2.2 Evaluate Model and Calculate Class Accuracy

```
correct = 0
total = 0
c=[]
class_correct=[[0]*43]
class_total=[[0]*43]
for i in range(len(ts_images)):
    inputs=ts_images[i]
    labels=ts_labels[i]
    inputs.unsqueeze_(0)
    labels.unsqueeze_(0)
    #print(i,inputs.shape,labels.shape)
    inputs, labels = inputs.to(device), labels.to(device)
    outputs = modelf.eval()(inputs)
    out=outputs.cpu()
    out=out.permute(1,0)
    out=out.detach().numpy()
    out = [i[0] for i in out]
    softmax=np.exp(out)/sum(np.exp(out))
    c.append(max(softmax*100))
    _, predicted = torch.max(outputs.data, 1)
    total += labels.size(0)
    class_total[0][labels]+=1
    if predicted==labels:
        class_correct[0][labels]+=1
    #print(labels,predicted)
    correct += (predicted == labels).sum().item()
print('Accuracy : %d %%' % (100 * correct / total))
c_acc=np.divide(class_correct,class_total)
```

D.2.3 Graffiti Attacks

```
def g_attack1(img,lab):
    img=toPIL(img)
    img=resize(img)
    #print(img.shape)
    img[:,75:85,25:85]=255
    img[:,25:35,45:55]=255
    img[:,50:70,25:35]=255
    img=toPIL(img)
    plt.imshow(img)
    plt.show()
    correct=0
    img=transform(img)
    img.unsqueeze_(0)

    img,lab=img.to(device),lab.to(device)
    outputs = modelf.eval()(img)
    _, predicted = torch.max(outputs.data, 1)
    correct = (predicted == lab).sum().item()
    print(predicted,correct)
```

```
    out=outputs.cpu()
    out=out.permute(1,0)
    out=out.detach().numpy()
    out = [i[0] for i in out]
    softmax=np.exp(out)/sum(np.exp(out))
    print('Attack 1 Confidence',max(softmax*100))
    img=torch.reshape(img,(3,112,112))
    if enable==1:
        return img
    else:
        return

def g_attack2(img,lab):
    img=toPIL(img)
    img=resize(img)
    #print(img.shape)
    img[:,50:60,30:70]=1
    img=toPIL(img)
    plt.imshow(img)
    plt.show()
    correct=0
    img=transform(img)
    img.unsqueeze_(0)

    img,lab=img.to(device),lab.to(device)
    outputs = modelf.eval()(img)
    _, predicted = torch.max(outputs.data, 1)
    correct = (predicted == lab).sum().item()
    print(predicted,correct)
    out=outputs.cpu()
    out=out.permute(1,0)
    out=out.detach().numpy()
    out = [i[0] for i in out]
    softmax=np.exp(out)/sum(np.exp(out))
    print('Attack 2 Confidence',max(softmax*100))
    img=torch.reshape(img,(3,112,112))
    if enable==1:
        return img
    else:
        return

def g_attack3(img,lab):
    img=toPIL(img)
    img=resize(img)
    #print(img.shape)
    img[:,75:80,65:85]=1
    img[:,25:35,40:50]=1
    img[:,50:65,25:35]=255
    img[:,55:70,65:75]=255
    img=toPIL(img)
    plt.imshow(img)
    plt.show()
    correct=0
    img=transform(img)
    img.unsqueeze_(0)

    img,lab=img.to(device),lab.to(device)
    outputs = modelf.eval()(img)
    _, predicted = torch.max(outputs.data, 1)
    correct = (predicted == lab).sum().item()
    print(predicted,correct)
    out=outputs.cpu()
    out=out.permute(1,0)
    out=out.detach().numpy()
    out = [i[0] for i in out]
    softmax=np.exp(out)/sum(np.exp(out))
    print('Attack 3 Confidence',max(softmax*100))
    img=torch.reshape(img,(3,112,112))
```

```
    if enable==1:
        return img
    else:
        return
```

D.2.4 Individual Attacks for Data Collection

```
attacks={3}
enable=0
img=mpimg.imread(r'/home/konstantinoskk//Documents/linux_ip/Street/adversarial_ex/test_graffiti/00593.ppm')
lab=torch.tensor(2)
if 1 in attacks:
    g_attack1(img,lab)
if 2 in attacks:
    g_attack2(img,lab)
if 3 in attacks:
    g_attack3(img,lab)
```

D.3 Adversarial Training

D.3.1 Collect and Convert Images

```
index=[]
for i, row  in labels.iterrows():
    index.append(int(row))
print(index)
rc={} #Hashmap datastructure for faster search
for i in range (len(index)):
    if int(index[i])<= 10000:
        index[i]='0'+str(index[i])
        if int(index[i])<= 1000:
            index[i]='0'+str(index[i])
        if int(index[i])<= 100:
            index[i]='0'+str(index[i])
    else:
        index[i]=str(index[i])
    index[i]=str(index[i])+'.ppm'
    rc[index[i]]=1
print(index)
PATH=r'/home/konstantinoskk/Documents/linux_ip/Street/Test'
DEST=r'/home/konstantinoskk/Documents/linux_ip/Street/adversarial_ex/adv_tr/test_stop_signs'
count=0
for filename in os.listdir(PATH):
    if filename in rc:
        file=str(filename)
        file=file[:5]
        file=str(DEST)+str(file)
        filename=PATH+str(filename)
        print(filename)
        #print(file)
        !convert echo {filename} to echo {file}.jpg
```

D.3.2 Training Attacks

```
def g_attack1(img,lab):
    img=toPIL(img)
    img=resize(img)
    #print(img.shape)
    img[:,75:85,25:85]=255
    img[:,25:35,45:55]=255
    img[:,50:70,25:35]=255
    if enable==1:
        return img
    else:
        return

def g_attack2(img,lab):
    img=toPIL(img)
    img=resize(img)
    #print(img.shape)
    img[:,50:60,30:70]=1
    if enable==1:
        return img
    else:
        return

def g_attack3(img,lab):
    img=toPIL(img)
    img=resize(img)
    #print(img.shape)
    img[:,75:80,65:85]=1
    img[:,25:35,40:50]=1
    img[:,50:65,25:35]=255
    img[:,55:70,65:75]=255
    if enable==1:
        return img
    else:
        return
```

D.3.3 Adversarial Network Training

```
       enable=1
def Net(model, criterion, optimizer, scheduler, num_epochs):
    start = time.time()
    tloss=0

    k,st=0,0
    vtemp=0

    for e in range(num_epochs):

        for i,(inputs, labels) in enumerate(load_tr1,0):

            if (i % 5==0):
                num=np.random.randint(3)

                inputs=torch.reshape(inputs,(3,112,112))
                if num==0:
                    inputs=g_attack1(inputs,labels)
                elif num==1:
                    inputs=g_attack2(inputs,labels)
                elif num==2:
                    inputs=g_attack3(inputs,labels)
                inputs.unsqueeze_(0)
            #print(inputs.shape)
```

```
            optimizer.zero_grad()
            inputs = inputs.to(device)
            labels = labels.to(device).long()
            outputs = model(inputs)
            #print(labels)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            tloss = loss.item()
            ac=labels.shape[0]
            st+=ac
            inputs = inputs.to('cpu')
            labels = labels.to('cpu')

            print('Epoch:',e+1,'/',epochs,'Step:',st,'/',total_step,'Step Loss:',tloss, end="\r")
            loss_list[k,0]=tloss
            k+=1

        torch.save(modelf.state_dict(), r'/home/konstantinoskk//Documents/linux_ip/model/'+str(e)+'.pth')
        pe=step_pe*(e+1)
        eloss[e,0]= np.mean(loss_list[(pe-step_pe):(pe),0])
        print('----------------','Epoch:',e+1,'Loss:',round(eloss[e,0],10),'----------------')

    time_elapsed = time.time() - start
    print('Training complete in {:.0f}m {:.0f}s'.format(time_elapsed // 60, time_elapsed % 60))
    return model
```

D.3.4 Adversarial Testing

```
def testacc(model,load_ts):
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs,labels in load_ts:
            if (i % 5==0):
                num=np.random.randint(3)
                inputs=torch.reshape(inputs,(3,112,112))
                if num==0:
                    inputs=g_attack1(inputs,labels)
                elif num==1:
                    inputs=g_attack2(inputs,labels)
                elif num==2:
                    inputs=g_attack3(inputs,labels)
                inputs.unsqueeze_(0)

            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            #print(inputs.shape)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()


        print(correct,total)
    return (100 * correct / total)
```

D.4 Defensive Distillation

D.4.1 Training of First Network

```
def Net(model, criterion, optimizer, scheduler, num_epochs):
    start = time.time()
    tloss=0
    k,st=0,0
    vtemp=0
    for e in range(num_epochs):

        for i,(inputs, labels) in enumerate(load_tr,0):

            optimizer.zero_grad()
            inputs = inputs.to(device)
            labels = labels.to(device)
            outputs = model(inputs)
            outputs=fun.log_softmax(outputs/40, dim=-1)
            ll=outputs.cpu().detach().numpy()
            ll=torch.as_tensor(ll)
            #ll=ll.to(device).long()
            #print(outputs.shape,ll.shape)

            outputs=outputs.to(device)
            loss=criterion(outputs,labels)

            loss.backward()
            optimizer.step()

            tloss = loss.item()
            ac=labels.shape[0]
            st+=ac
            inputs = inputs.to('cpu')
            labels = labels.to('cpu')
    return model

criterion = nn.NLLLoss()
optimizer =torch.optim.Adam(modelf.parameters(), lr=0.01)
exp_lr_scheduler = lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.1)
modelf = Net(modelf, criterion, optimizer, exp_lr_scheduler,num_epochs=epochs)
```

D.4.2 Generate Distillation Labels

```
correct = 0
total = 0
count=[0,200]
new_labels_tr=np.zeros((len(tr_labels),43))

with torch.no_grad():
    for inputs,labels in load_tr:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = modelf(inputs)
        print(outputs.shape)
        ll=outputs.cpu().detach().numpy()
        new_labels_tr[count[0]:count[1],:]=ll
        count[0]+=200
        count[1]+=200
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        #print(labels,predicted,(predicted == labels).sum().item())
print('Accuracy : %d %%' % (100 * correct / total))
new_labels_tr=torch.as_tensor(new_labels_tr)
```

```
new_labels_tr=fun.softmax(new_labels_tr/40,1)
#%store new_labels
torch.save(new_labels_tr,r'/home/konstantinoskk//Documents/linux_ip/Street/new_
    labels_tr.pt')
```

D.4.1 Custom Loss Function

```
def cross_entropy_loss(out, labels):
    logsoft = nn.LogSoftmax(dim=0)
    return torch.mean(torch.sum(- labels * logsoft(out), 0))
```

D.5 LFW Evaluation

D.5.1 Load Pairs

```
def load_pairs(txt_name,PATH):
    path = open(str(txt_name))
    lines = path.readlines()
    labs=[]
    for i in range(len(lines)):
        lines[i]=' '.join(lines[i].split())
        ll=lines[i].split(' ')

        #print(i)
        if len(ll)==3:
            labs.append(PATH+ll[0]+'/'+ll[0]+fill(ll[1]))
            labs.append(PATH+ll[0]+'/'+ll[0]+fill(ll[2]))

        elif len(ll)==4:
            labs.append(PATH+ll[0]+'/'+ll[0]+fill(ll[1]))
            labs.append(PATH+ll[0]+'/'+ll[0]+fill(ll[3]))


    return labs
def load_img(PATH):
    images=[]
    names=[]
    for filename in os.listdir(PATH):
        filename=PATH+str(filename)
        print(filename)
        img=mpimg.imread(filename)
        images.append(img)
        names.append(filename)
    return images,names
```

D.5.2 Calculate Optimal Threshold

```
def accuracy(th,cosim,labs):
    correct=0
    for i in range(len(cosim)):

        if cosim[i]>=th:
            if labs[i]==1:
                correct+=1

        elif cosim[i]<th:
            if labs[i]==0:
                correct+=1
    return correct
```

D.6 Facial Attacks and Defences

D.6.1 Adversarial Attacks

```
model=foolbox.models.PyTorchModel(net, (0,1), 512, preprocessing=(0, 1))
import torchvision.transforms as transforms
resize1=transforms.Resize((112,96))
randcrop=transforms.RandomCrop((112,96))
grey=transforms.Grayscale(num_output_channels=1)
transform = transforms.ToTensor()
toPIL=torchvision.transforms.ToPILImage(mode=None)
#attack = foolbox.attacks.FGSM(model)
criterion=foolbox.criteria.Misclassification()
attack=foolbox.attacks.BoundaryAttack(model=model, criterion=criterion,threshold=0.01)
attack=foolbox.attacks.GaussianAdditiveNoiseAttack(model=model, criterion=criterion,threshold=0.01)
attack=foolbox.attacks.SaltPepperAttack(model=model, criterion=criterion,threshold=0.01)
adversarial = attack(im,lab)
```

D.6.2 Adversarial Defences

```
from advertorch.defenses import MedianSmoothing2D
from advertorch.defenses import BitSqueezing
from advertorch.defenses import JPEGFilter

bits_squeezing = BitSqueezing(bit_depth=17)
median_filter = MedianSmoothing2D(kernel_size=3)
jpeg_filter = JPEGFilter(75)

defense = nn.Sequential(
    jpeg_filter,
    bits_squeezing,
    median_filter,
)
blur_image=defense(i1)
blur_image.shape

a1=np.asarray(a)
#a2=torch.as_tensor(a1)

blur_image = cv2.medianBlur(a1,15)
gg=np.reshape(blur_image,(3,112,96))
bi=torch.as_tensor(blur_image)
bi=bi.permute(2,0,1)
print(bi.shape)
bi=toPIL(bi)

a1=np.asarray(a)
#a2=torch.as_tensor(a1)

blur_image = cv2.GausianBlur(a1,(7,7))
gg=np.reshape(blur_image,(3,112,96))
bi=torch.as_tensor(blur_image)
bi=bi.permute(2,0,1)
print(bi.shape)
bi=toPIL(bi)

plt.imshow(bi)
```