RUHR
UNIVERSITÄT
BOCHUM

RUB

RUHR-UNIVERSITÄT BOCHUM

Evaluating defense mechanisms against universal adversarial perturbations for convolutional neural networks

Pablo Schmücker

Master or Bachelor Thesis – March 31, 2024.
Chair for Information Security.

1st Supervisor: Prof. Dr. Ghassan Karame
2nd Supervisor: M. Sc. Pascal Zimmer

InfSec

HG

Abstract

Adversarial attacks pose a major threat to the security and reliability of machine learning models. Particularly dangerous are universal adversarial perturbations (UAPs), which are computed once and are applicable to almost every input across numerous models. In the past, various defense strategies have been developed for conventional adversarial perturbations, however, their performance has never been evaluated on UAPs. For this reason, we analyse four well-known defenses for their performance against UAPs. We find out that most of them provide a solid protection factor against UAPs, although the performance is highly volatile depending on the dataset. For this reason, we develop based on the perturbation rectifying network an optimal defense strategy that protects against both conventional and universal adversarial perturbations. It achieves an exceptionally high performance reducing the foolrate from nearly 100% to under 20% for almost every dataset and attack evaluated. It has proven to be particularly reliable across all evaluated attacks and offers an almost identical clean accuracy in comparison to the base model. The findings of this thesis can be used to assess which defenses provide protection against UAPs and to protect own machine learning models against adversarial attacks whether they are universal or not.

Contents

1 Introduction

Until a few years ago, deep learning and artificial intelligence were just a small area of research within computer science. The beginnings of machine learning date back to the 1950s. At that time, it was a purely theoretical field of research, developing algorithms and architectures for machine learning. The computers available at the time were nowhere near as powerful enough to implement these algorithms. This changed in the early 2000s with advances in computing power and data availability. It was now possible to train advanced machine learning models which became deeper and deeper in their architecture, coining the term 'deep learning'. In the following years, the availability of large amounts of data and the development of increasingly powerful graphics processing units (GPUs) led to breakthroughs in image recognition, speech processing and time series analysis. The progress in recent years has been so rapid that the trend towards artificial intelligence has reached the masses. AI is being used in almost every aspect of life: checking emails for spam, keeping cars on track or recognizing faces on ID cards. Therefore AI is also increasingly being used in critical applications, making the security and reliability of AI models extremely important.

When developing and training algorithms, performance is often the only relevant metric. Architectures and algorithms are designed in such a way that they fulfill their task as error-free as possible. Until a few years ago, a central issue, the security, was left out of the equation. In 2013, the paper "Intriguing properties of neural networks" [30] was published. It was one of the first paper addressing the issue of security by providing a new type of attack against AI. The researchers showed that it is possible to fool an AI model using so-called adversarial perturbations, thereby compromising the reliability of its predictions. Adversarial perturbations are deliberate changes to the input data that are barely noticeable to a human, but influence the AI so massively that it produces incorrect predictions. In their paper, Szegedy et al. focused on the generation of adversarial perturbations for images. They were able to show that the altered images can fool advanced convolutional neural networks (CNNs), posing a serious threat. During this work, we will refer to them as conventional adversarial perturbations (CAPs). It was shown, that CAPs can influence a high number of safety-critical applications such as the traffic sign recognition in autonomous cars or the facial recognition in passport controls [35, 31]. Without appropriate defense measures, this can lead to catastrophic consequences, as shown in Figure 1.1 for autonomous driving. It shows three different scenarios in which adversarial attacks influence the behavior of the corresponding vehicle. Not only is it

possible to manipulate traffic sign recognition or the lane assistant, it is also possible to add or remove objects or entire persons from the vehicle's field of vision. This gives the attacker almost complete control of the vehicle.

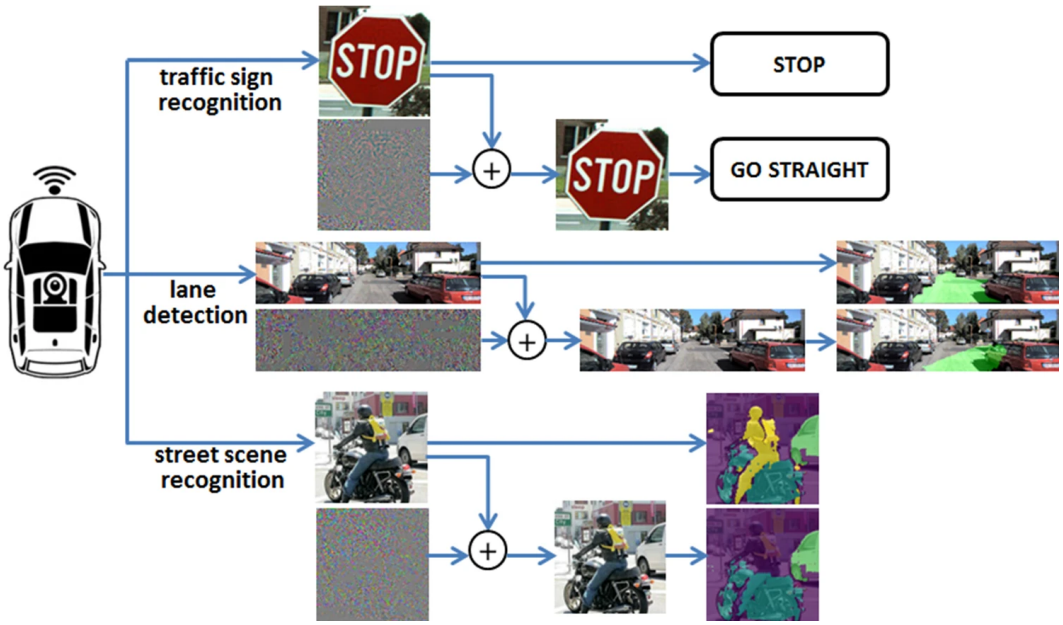

Figure 1.1: Different attack vectors for autonomous driving[43]

The use of CAPs is most common in images, however they can also be applied to other input data, such as text or audio [25, 7]. In this work we will only discuss adversarial examples in the form of images, since most of the research is conducted on this type of perturbations.

Based on the research of Szegedy et al., numerous methods and algorithms were developed in the following years to make CAPs even more effective and faster to compute [4]. All of the algorithms developed had one important thing in common: the perturbation had to be computed individually for each input, which had a negative impact on the computational power and flexibility of the attack. Therefore, the aim was to find perturbations that are transferable both at model and input level. Moosavi-Dezfooli et al. made the crucial breakthrough in 2017 with their work on universal adversarial perturbations (UAPs) [22]. Unlike CAPs, which are tailored to a specific model and input, universal adversarial perturbations can be applied to a wide range of models and inputs. UAPs are as well so minimal that they are barely perceptible to humans. Because of their high transferability and imperceptibility, UAPs are a dangerous attack vector for machine learning models.

It is therefore of great importance to make machine learning models more resilient to adversarial attacks, especially against UAPs. For this reason we will evaluate different defense measures regarding their effectiveness on UAPs and compare their performance. We use these findings to implement a new defense strategy which achieves the best possible defense performance against both CAPs and UAPs.

1.1 Approach

Although the existence of UAPs has been known for years, the research has mainly focused on CAPs. Therefore, there are only a small number of papers dealing with protection against UAPs. Most protective measures have been designed and evaluated exclusively for CAPs, leaving their effectiveness against UAPs unknown. As a consequence, it is not unlikely that some of these defenses are vulnerable to UAPs, posing a potential security risk to the models in which they are deployed. For this reason, we want to analyze the effectiveness of the defenses against UAPs so that models are protected against both CAPs and UAPs in the future. For this purpose, we implement four different defenses on various models and datasets. The evaluation is performed on UAPs to collect the data for our research question and on PGD-10 perturbations to ensure the effectiveness of the defenses on CAPs. We will then compare the results with the performance of defenses that were explicitly developed for UAPs. In order to make this as representative as possible, we use two different algorithms to generate the UAPs: Deepfool for better comparability with other papers and Stochastic Gradient Aggregation (SGA), as this algorithm has been shown to generate the most powerful UAPs. The aim of this work is to determine whether defenses developed for CAPs are also robust against UAPs. Based on the results, we can also determine which defense strategies offer the highest protection against UAPs and what performance can be expected. In the best case scenario, we find a way to protect machine learning models from both UAPs and CAPs

1.2 Related Work

So far, only a few papers have been published that systematically evaluate the performance of different CAPs/UAPs. Among them is the publication by Jing Wu et al. who evaluated different adversarial attacks on clean models in the year 2021 [37]. Another relevant paper provides a good overview of the performance of different UAPs [34]. There, UAPs are evaluated extensively using different thread models. Despite the recent publication date, the state of the art SGA algorithm was not considered in the evaluation by Juanjuan Weng et al.. In addition, the performance was only evaluated on clean models that have no integrated defense. There are numerous surveys on the topic of adversarial defenses in the published literature [41, 9, 8]. They

usually describe how the individual defenses work and summarize the performance values of the original papers. Rarely are a separate evaluation of various defenses is conducted, like in the paper by Gaurang Sriramanan et al. [28]. There they evaluate the protection factor of different defenses against adversarial attacks. However, only CAPs are used, so the protection factor against UAPs remains unknown. Due to the small number of UAP-specific defenses, we are not aware of any paper that systematically evaluates them. To our knowledge, no CAP specific defenses have been tested on UAPs to the extent that we will do in this paper.

1.3 Structure of the thesis

This thesis is divided into five separate chapters. The introductory chapter provides an overview of the research topic, its significance and the objectives of the thesis. The background chapter provides the necessary theoretical foundation and context for understanding the research topic. It includes the key concepts of machine learning, a technical introduction to the attacks and defenses and an overview of existing approaches. In the methodology section, we explain the experimental design, introduce the evaluation metrics and describe the implementation of both attacks and defenses. The results of our experiments are discussed in detail in the evaluation chapter. There, we analyse our results and compare them with each other. In the final chapter, we summarise the main findings of the evaluation, discuss their relevance, and outline the contributions of our work. We also suggest possible directions for future research.

2 Background

This chapter introduces the basics of machine learning and discusses all relevant aspects that are necessary for understanding the following sections of this thesis. It begins with an overview of machine learning and explains its basic concepts and functionalities. We will then discuss the algorithms used in modern applications, including neural networks and convolutional neural networks. Following this, we will cover the learning process and optimization algorithms. After that, we will give an overview of adversarial perturbations and explain the calculation process. At the end we will thematise the defenses that we will evaluate in this thesis.

2.1 Deep Neural Networks

At the beginning of this work, we would like to explain some basic terms and concepts and categorize them in the growing field of artificial intelligence. Artificial intelligence (AI) is a field in computer science where the imitation of human cognitive abilities by machines is researched. AI serves as a generic term (especially in the media) for numerous approaches that are based on both pre-programmed processes and machine learning algorithms. The term machine learning, which is often used synonymously with AI, is a sub-area of AI that focuses on the development of algorithms which enable a computer to learn. Impressive progress has been made in this area in particular over the last few decades. A particularly powerful class of machine learning algorithms are artificial neural network (NN). The basic idea behind them is to artificially model the human brain with its neurons and synapses. Based on the biological model, NNs have so-called nodes and weights which are the equivalent of neurons and synapses. An NN is often represented as a graph, an example of which can be seen in Figure 2.2. The architecture of an NN is built up in layers. Each layer contains a fixed number of nodes that are connected to the nodes of the next layer. A distinction is made between the input layer, the hidden layer and the output layer. The input layer is the interface between the input data and the neural network. Most of the learning process takes place in the hidden layer. This is where the NN extracts the patterns and features of the input data and uses them to learn. The name is due to the fact that the calculations within the hidden layer are not directly accessible. The output layer converts the extracted features into an output and provides the corresponding prediction. If an NN contains many hidden layers, it

is called a deep neural network (DNN). The sequence of many hidden layers makes it possible to develop very powerful NNs, as the learning process takes place in these layers and thus allows complex features to be learnt. This architecture gave the term deep learning its origin, which describes the use of DNNs for machine learning. Deep learning is therefore a subcategory of machine learning.

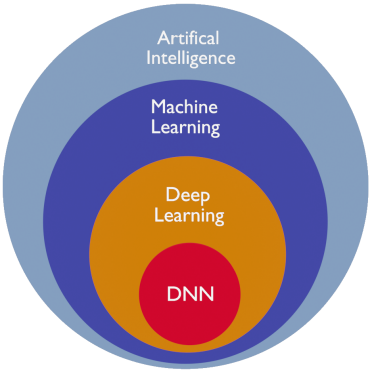

Figure 2.1: Overview of the subcategories of artificial intelligence

The simplest architecture of a DNN is a multilayer perceptron (MLP) which is represented in Figure 2.2. It can be used for numerous tasks including image recognition, regression and speech processing. Although MLPs are very flexible, they quickly reach their capacity limits for specialized tasks such as image recognition. This is due to the shape of the architecture. For an image with dimensions of 200 × 200 pixels, a total of 40,000 nodes are required in the input layer to process the image pixel by pixel. If the first hidden layer is also 40,000 nodes in size, this results in 1,600,000,000 connection edges that are required to connect the first two layers! (assuming a fully connected network) For this reason, MLPs are not suitable for very large input data. We would therefore like to present a more advanced architecture that specializes in image processing.

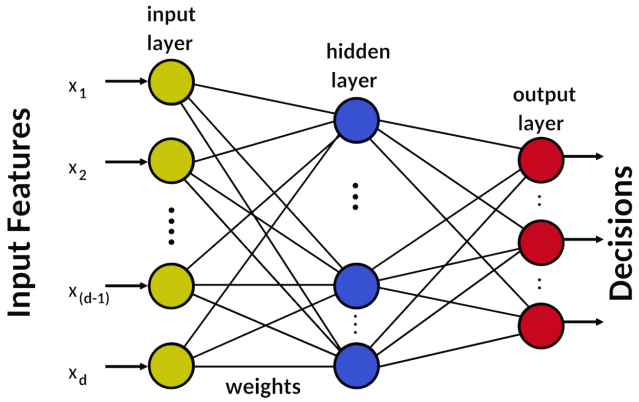

Figure 2.2: Structure of an artificial neural network consisting of three layers

2.2 Convolutional Neural Networks

Convolutional neural networks also belong to the DNNs. The first part of the name reveals the special nature of this architecture. Convolution describes a mathematical operation in which two functions are overlaid to create a new function. This is an essential component of the so-called convolutional layer. The convolutional layer has the task of extracting patterns from small areas of the image. It consists of one or more filters that move across the image and perform a convolution operation. A filter is basically a matrix that is placed over a section of the image and is combined with the image values. Depending on which values the matrix contains, other features are extracted from this image area (e.g. vertical lines or corners). The values of the matrix are adjusted during the training process so that exactly those features are extracted that are necessary for the classification of the images. The convolution between the filter and the image results in a feature map that represents the extracted features of the image. Usually, a convolutional layer applies several filters simultaneously to the input, so that different feature maps and thus different features are extracted.

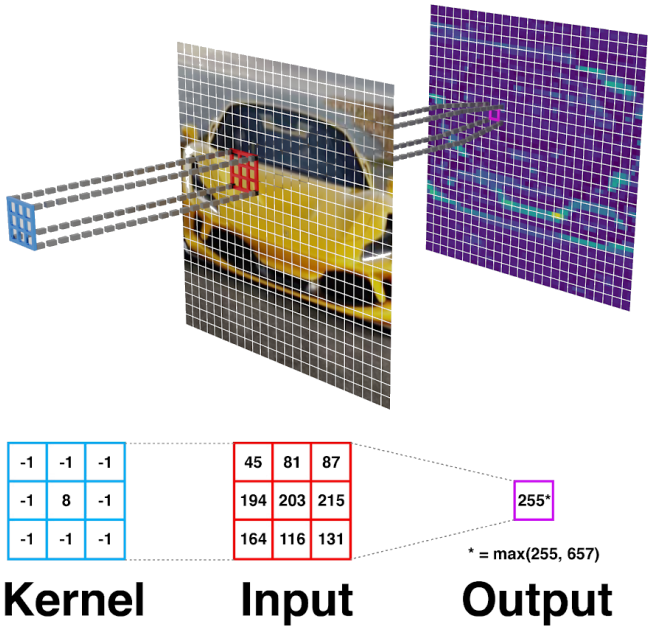

Figure 2.3: Generation of a feature map by the convolution between input image and filter

Figure 2.4 shows some examples of feature maps that were generated in the first convolutional layer of the Resnet50 model. It is obvious that the feature maps were generated by different filters.

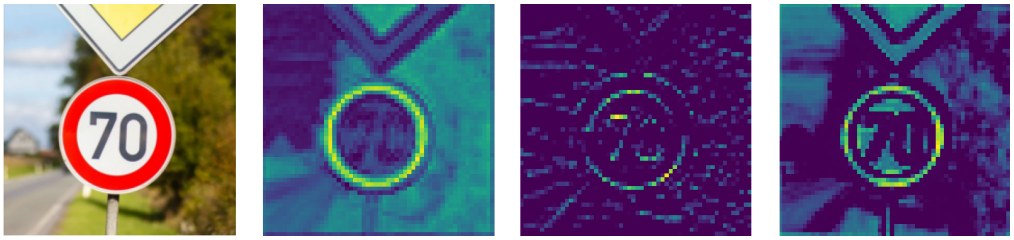

Figure 2.4: Feature maps extracted from the Res1 block of a Resnet50 trained on Imagenet. The image on the left is the original input image, the other three are feature maps generated with different filters

The feature maps, which are the output of a convolutional layer, are typically processed further by a pooling layer. Within the pooling layer, the dimension (i.e. the size) of the feature maps is reduced in order to reduce the computing load and improve the generalization capability of the network. There are various approaches to reduce the dimensions of feature maps. One of the most common is max-pooling, in which the feature map is divided into small image areas and only the maximum value of each area is used. In this way, the most distinctive features are retained and the dimensionality of the feature map is reduced by a factor of 2-4. A simplified illustration of how max pooling works is shown in the Figure 2.5 below.

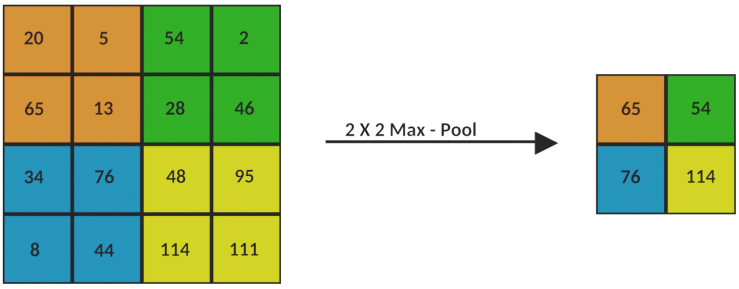

Figure 2.5: Visual representation of the dimension reduction through Max Pooling

The classic architecture of a CNN contains several convolutional and pooling layers that are arranged alternately. As the data progresses through the network to the output layer, the dimensionality of the feature maps decreases, with the extracted features becoming more and more distinctive. The last pooling layer is followed by a classic neural network like the one presented above. Here, the feature maps from the last pooling layer are converted into a vector and used as input for the NN. In this step, the feature maps are only a few pixels in size, so the total size of the input layer consists of a few 100 to 1000 nodes and not 40,000 as in the example above. This dimensional reduction is one of the reasons why CNNs are used for image recognition. The NN processes the input and then returns a prediction. The CNN architecture

described above is shown graphically in Figure 2.6. The input image is processed here from left to right. It should be noted that the functionality of the CNN is presented in a simplified form. In reality, additional layers and modules are used to make the feature extraction as reliable and precise as possible. However, this level of detail is beyond of this work and is not further relevant for the basic understanding of CNNs.

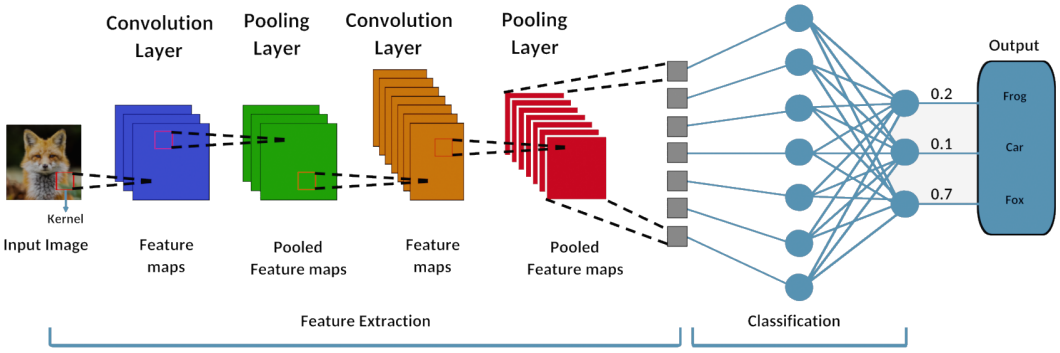

Figure 2.6: Exemplary representation of a CNN architecture [27]

2.3 Learning process

In the field of machine learning, there are three major categories which describe the learning type of a model: supervised learning, unsupervised learning and reinforcement learning. Since DNNs are mainly assigned to the category of supervised learning, we will take a closer look at this class. Supervised learning describes the training with data for which the corresponding output value (labels) is already known. This could be, for example, images and their corresponding class or numerical data (e.g. in property analysis) and the corresponding target value (price). In this way, a model can learn patterns and correlations between the input data and the labels. To do so, the internal parameters (the weights of the model) are adjusted during training. The adjustment of the parameters can be understood as a mathematical optimisation problem. The output of the DNN is the actual value and the label is the target value. Using a loss function, the deviation between the actual and target values is calculated. One common loss function used in multiclass classification tasks is the cross-entropy loss [1].

$$\mathcal{L}\left(y_j, \hat{y}_j\right) = \quad -\sum_{j=1}^{K} y_j \log\left(\hat{y}_j\right) \tag{2.1}$$

The loss is calculated by iterating through each class and multiplying the current label y_j with the logarithm of the prediction $\hat{y}_j$, which is represented as a probability

[1]. By using the logarithm, the function returns a higher loss value for predictions that are confidently wrong. Based on the output of the Loss function, the parameters of the DNN are adjusted by an optimization function, which makes the model learn. This process is repeated several times and leads to an increasingly adaptation of the model to the training data. The aim of the training is to minimise the value of the loss function (loss value) so that the error between the output of the DNN and the label is minimal. The whole process can be illustrated as a movement on a multi-dimensional landscape. The loss value in dependence to the model parameters results in a multidimensional landscape in which the global minimum is searched for. The movement within the landscape is achieved by adjusting the model parameters. The adjustment of the parameters is determined by gradients that indicate the direction of the steepest increase within a given function, in this case the loss function. They are calculated by taking the partial derivatives of the loss function with respect to each parameter.

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \frac{\partial \mathcal{L}}{\partial \hat{y}_j} \cdot \frac{\partial \hat{y}_j}{\partial x_j} \cdot \frac{\partial x_j}{\partial w_{ij}} \tag{2.2}$$

2.2: Chain rule to calculate the derivatives of the loss function $\mathcal{L}$ with respect to the model parameters w_{ij}. The input of each neuron is represented by x_j and the corresponding output by $\hat{y}_j$ [2]

The gradients point in the direction in which the loss value increases most rapidly. Therefore, the parameters are updated in the direction of the negative gradient in order to minimise the loss value. The aim is to reach the global minimum of the functional landscape.

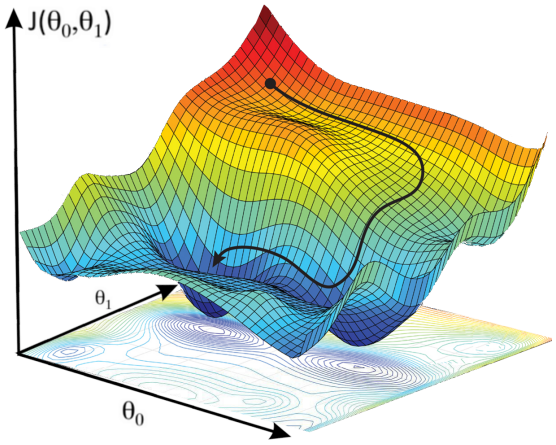

Figure 2.7: Gradient descent in a 3-Dimensional landscape [12]

This process is called gradient descent and is carried out iteratively, whereby the error size of the model is continuously reduced. By minimising the loss value, the model continuously adapts to the training data, although it is essential that the model retains a certain generalisation capability and is not adapted too much to the training data (overfitting). If this were the case, the performance of the model on new data would drop drastically. The opposite of overfitting is underfitting which means that the model cannot fully adapt to the training data due to an insufficient number of parameters. In this case, the performance would also decrease significantly. The challenge is therefore to construct a model that is complex enough to learn the input data, but with parameters that can be regulated to avoid overfitting. The parameters that are responsible for regulating the model are called hyperparameters. One of the most important hyperparameter is the learning rate, which indicates how much the model weights are adjusted during the training process. It can be regarded as the step size with which the gradient descent is performed. A learning rate that is too high can lead to oscillations or even to the divergence of the training, as the gradient descent shoots beyond the global minimum. If the learning rate is too low, it leads to slow or stagnating learning. Therefore, it is common practice to dynamically adjust the learning rate during training. The effect of the learning rate on the gradient descent is sketched in Figure 2.8. Since the training success of a DNN depends largely on the choice of hyperparameters, a appropriate selection is essential.

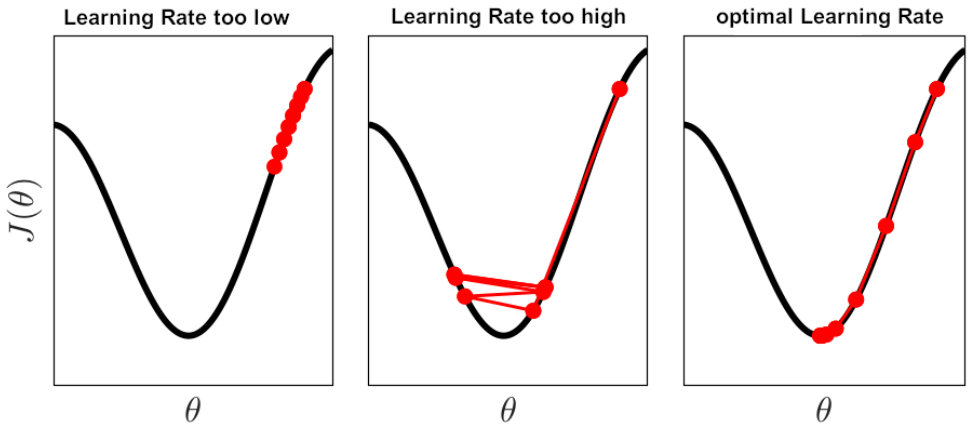

Figure 2.8: Impact of the learning rate on the convergence of the gradient descent [32]

2.4 Adversarial Perturbation

Adversarial perturbations are small, often barely perceptible changes to the input image that have a significant influence on the output of the model. The perturbations are constructed in such a way that they can hardly be recognised by humans or are only perceived as a weak noise. Although the perturbations appear random in their structure, they are calculated precisely for each pixel so that they exploit the weaknesses of a model. The calculation of (untargeted) adversarial perturbations can be described as the following optimisation problem [33]:

$$\begin{aligned} x^{\text{adv}} &= x + \delta \\ \delta &= \arg\min_{r} \{\|r\|_p : f(x+\delta) \neq f(x)\} \end{aligned} \tag{2.3}$$

Whereby x is the input, and $f(x)$ is the prediction of the DNN given the input x. There are numerous algorithms that can generate adversarial perturbations based on this optimisation problem [4]. Most of them use the gradients or the signs of the gradients to construct the perturbation (assuming a white-box setting). We distinguish between targeted and untargeted adversarial perturbations. Given a Dataset $\mathcal{D}$ and a target class $\mathcal{T}$, they are defined as follows:

$$\text{Targeted: } f(x+\delta) = \mathcal{T}\ \forall x \in \mathcal{X}, \mathcal{X} \subset \mathcal{D}, |\mathcal{X}| = 1 \tag{2.4}$$

$$\text{Untargeted: } f(x+\delta) \neq f(x)\ \forall x \in \mathcal{X}, \mathcal{X} \subset \mathcal{D}, |\mathcal{X}| = 1 \tag{2.5}$$

For targeted perturbations, the aim of is to influence the model in such a way that a specific class or a specific prediction $\mathcal{T}$ is returned. Whereas, the goal of an untargeted perturbation is to generate an incorrect prediction, regardless of the predicted class (label flip). In general, untargeted perturbations are more robust because they are not aiming at a specific target class.

Figure 2.9 compares original images with the perturbed ones. The predictions clearly indicate that a perturbation is embedded and successfully fools the DNN. To visualise the perturbations, they are amplified with a factor of $\times 100$.

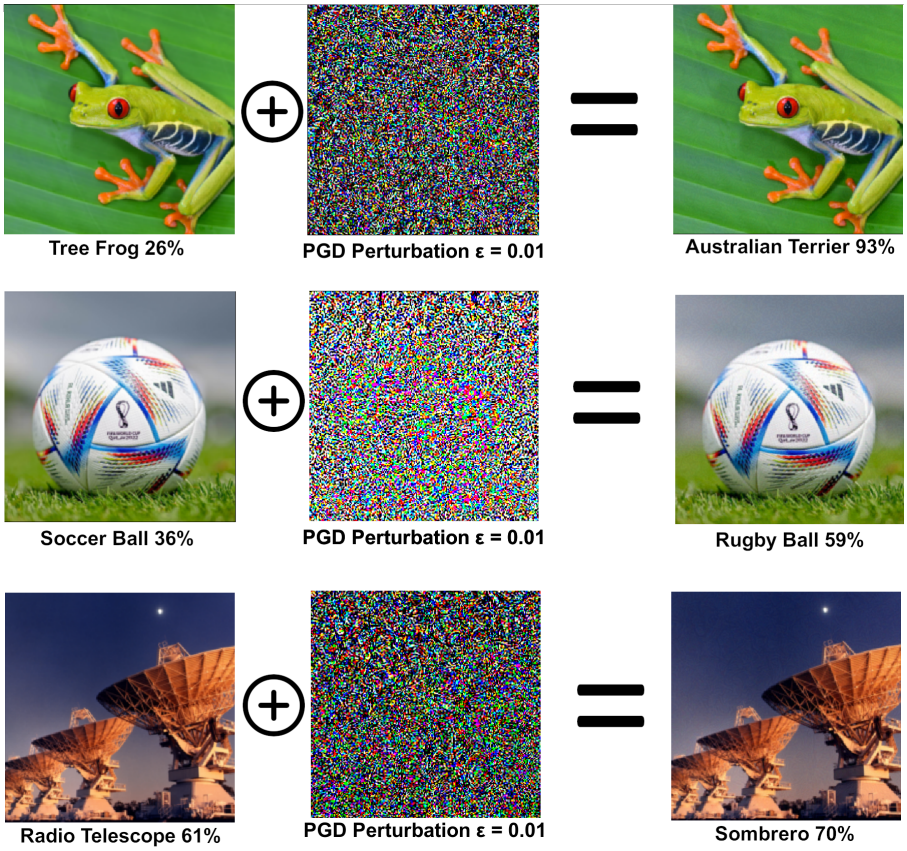

Figure 2.9: The original images (left) combined with the adversarial perturbation result in an adversarial input (right) that was consistently misclassified. The classification was performed by a Resnet50, using PGD-10 perturbations

To understand exactly how perturbations work, it is necessary to take a look at the input space. The input space represents all possible inputs that a DNN can process. In the context of images, this means that all RGB images with the dimension h×w (height×width) lie in an n = 3×h×w dimensional space. Each pixel value is assigned its own dimension, which creates the high-dimensional space. The input space for images is therefore the space of all possible combinations of pixel values that an image can represent. During training, a DNN learns to segment the input space and to assign a class to each area. Depending on the area in which an input image lies, the DNN classifies it correspondingly. The border between two classification areas is called the decision boundary. To achieve misclassification through adversarial perturbations, the image must be pushed over a decision boundary (figure 2.10). Each pixel change in the image causes the image to move within the input space. Adversarial perturbations change precisely those pixels that are necessary to move the input image across the decision boundary. The perturbation can therefore be

seen as a vector that moves the image within the input space. As each image has its own position, the perturbation must be calculated individually for each input. In most cases, the transfer of perturbations is not effective and does not result in a misclassification.

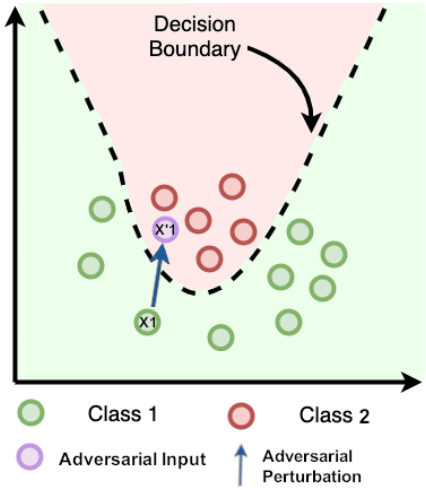

Figure 2.10: Shifting the input $X1$ across the decision boundary through an adversarial perturbation

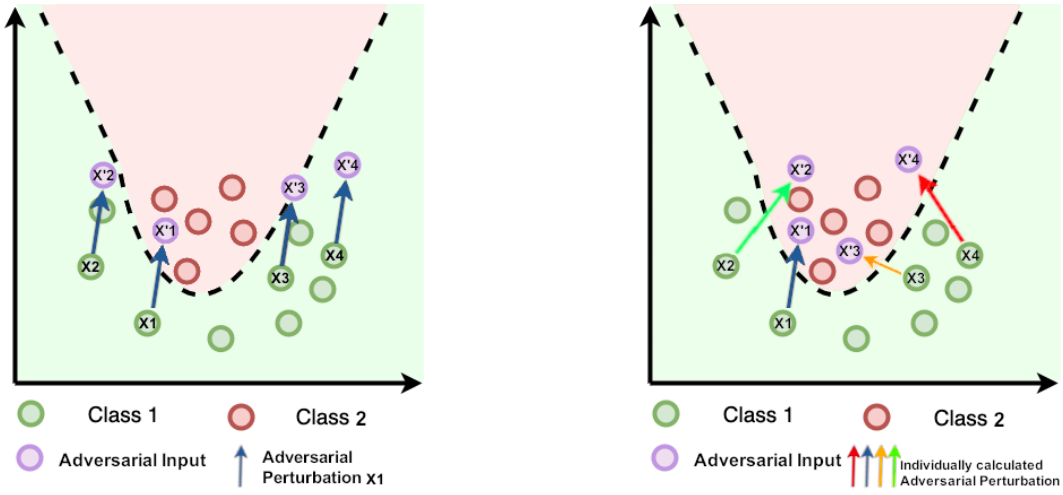

Figure 2.11: Left: Adversarial perturbation of input $X1$ applied to other inputs. A shift across the decision boundary is only successful for X1. Right: Adversarial perturbations calculated individually for each input.

2.4.1 Universal Adversarial Perturbations

The distinctive feature of universal adversarial perturbations (UAP) is that they can be applied to many different inputs. They work after the same principle as CAPs with the difference that they shift almost every input outside their own classification range. To achieve this, adversarial perturbations are usually calculated for entire batches (multiple input data) or individual perturbations are added together. The exact procedure depends on the algorithm used, which we will discuss later. Figure 2.12 sketches the aggregation of perturbations. The individual calculated vectors of $X1$, $X2$, $X3$ and $X4$ are aggregated and form a new vector. This vector is universal and shifts each of the four inputs outside its own classification area. In addition, UAPs also generalise well across various models. Moosavi and co. showed in their paper that UAPs generated on one specific model are quite effective on other models too [22]. Therefore, there is a universal transferability both on the input data and on the models themselves. UAPs can also be targeted or untargeted. However, they are mostly generated untargeted for reasons of robustness.

$$\text{Untargeted UAP: } f(x+\delta) \neq f(x) \forall x \in \mathcal{X}, \mathcal{X} \subseteq \mathcal{D}, 1 < |\mathcal{X}| \leq |\mathcal{D}| \tag{2.6}$$

Figure 2.13 shows some UAPs that were generated on different models. In contrast to conventional adversarial perturbations, it is noticeable that they contain a pattern that is influenced by the generation algorithm, the training data and the target model. As they do not have a noise-like structure, defenses that focus on noise removal may not be as effective. We will investigate this as well in our work.

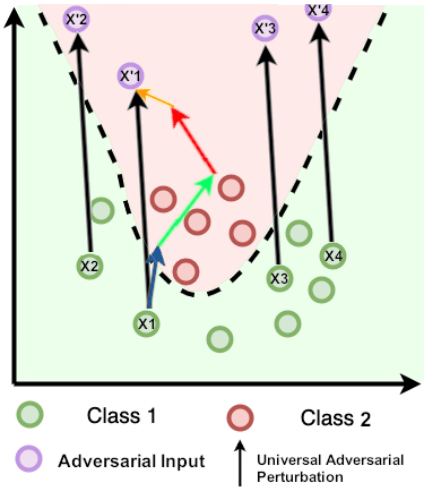

Figure 2.12: The addition of the individual perturbations results in a universal vector - the UAP

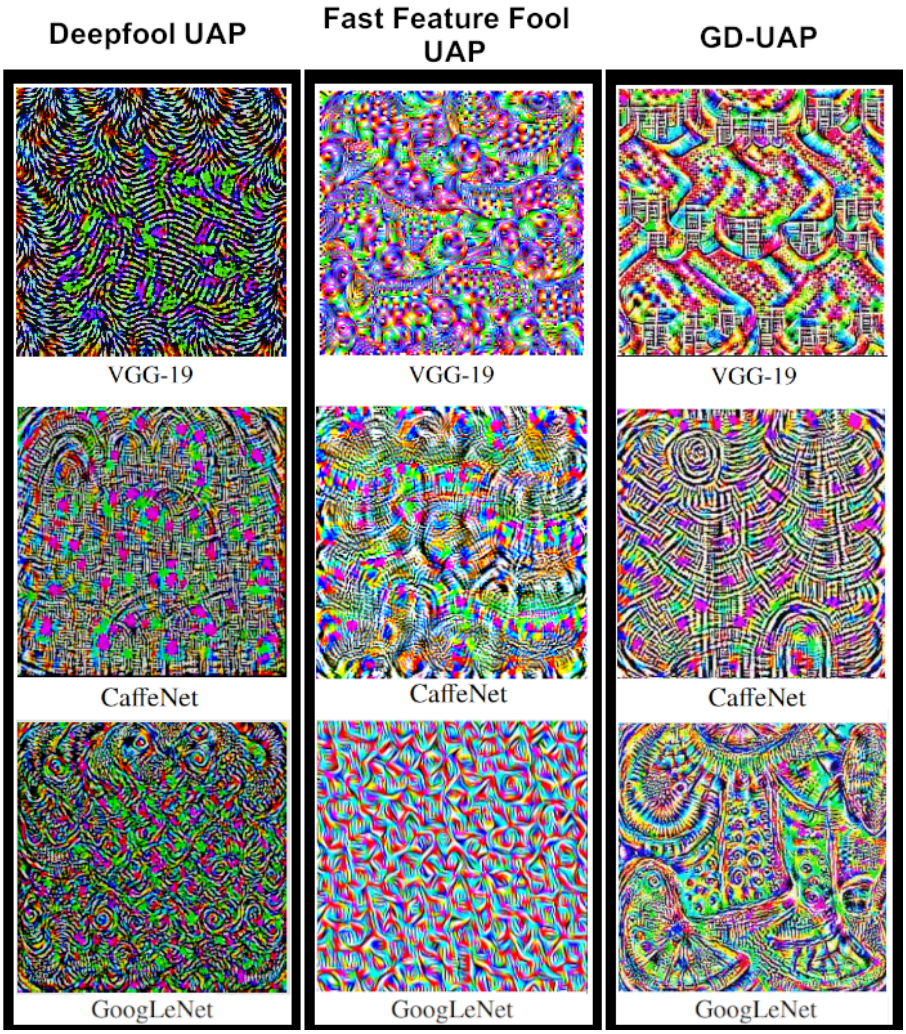

Figure 2.13: Overview of different UAPs generated by various algorithms on different models

PGD10

Projected Gradient Descent (PGD) is one of the most powerful algorithms for generating CAPs [19][26]. Developed by Madry et al., the iterative algorithm searches for an optimal perturbation where the values are limited by the L_p norm. PGD can be implemented both targeted and untargeted, although we only use the untargeted implementation in this work. In contrast to other methods such as the I-FGSM (Iterative Fast Gradient Sign Method), PGD projects the current perturbations onto the valid L_p space after each iteration (see L_p norm). This forces the optimization algorithm to search for a functioning perturbation that is within the

L_p range. This procedure results in particularly effective perturbations, as the optimization is performed in each iteration depending on the L_p norm. PGD calculates the gradients of the input image in each iteration and adjusts the pixel values accordingly. The number behind PGD indicates the number of iterations that PGD runs through. We use PGD-10 in this work, so each perturbation is generated by 10 iterations.

Algorithm 1: PGD generation loop for a adversarial step size α and N PGD steps. [36]

$\delta = 0$ // *or randomly initialized*
for $j = 1 \dots N$ **do**
  $\delta = \delta + \alpha \cdot (\nabla_\delta \ell(f_\theta(x_i + \delta), y_i))$
  $\delta = \max(\min(\delta, \epsilon), -\epsilon)$
end for

Deepfool

In many cases, algorithms developed for the computation of CAPs can be modified to also generate UAPs. This was demonstrated by Moosavi et al. in their work on UAPs [22]. They used the Deepfool [21] algorithm as the basis for generating the first UAPs. Deepfool is an iterative optimization algorithm that was introduced in 2015. It calculates approximately the orthogonal vector in the multidimensional space between the input x and the next hyperplane of the decision boundary to find the minimum perturbation of x. The intuition behind this is based on the fact that the orthogonal vector represents the shortest distance between the input x and the next hyperplane. This vector is used as a perturbation and shifts the original input x across the decision boundary. The perturbation of x is minimal (according to L_2 norm), since the shortest distance implies the smallest change.

Algorithm 2: Inner loop Deepfool UAP generation. [22]

for each datapoint $x_i \in X$ **do**
  if $\hat{k}(x_i + v) = \hat{k}(x_i)$ **then**
    Compute the *minimal* perturbation that sends $x_i + v$ to the decision boundary: $\Delta v_i \leftarrow \arg\min_r \|r\|_2$ s.t. $\hat{k}(x_i + v + r) \neq \hat{k}(x_i)$.
    Update the perturbation: $v \leftarrow \mathcal{P}_{p,\xi}(v + \Delta v_i)$.
  end if
end for

In each iteration, the algorithm behaves greedy, so it cannot be guaranteed that the optimal perturbation will be found. However, the observations of Moosavi et al. show that in practice Deepfool strongly approximates the minimum perturbation. This is also consistent with our observation, where the values of the perturbations we generated were negligibly small. The calculation of UAPs according to Moosavi et al. is basically an iterative execution of the deepfool algorithm on images of various classes (see algorithm 2). The aim is to calculate the minimum perturbation for each image and to combine this with the perturbations of the other images. This is achieved by adding the (orthogonal) vectors. In order to achieve the best possible generalization, the algorithm runs through several epochs whereby the number can be adjusted and have a direct impact on the quality of the UAPs.

SGA

The Stochastic Gradient aggregation algorithm (SGA) is currently the most powerful UAP generation algorithm. Xuanna Liu et al. demonstrated that they achieve a higher fooling rate using SGA than any other generation method published so far [17]. SGA solves two problems normally encountered in the calculation of UAPs: Gradient instability and quantization error. The term gradient instability describes the phenomenon that during the optimization process gradients can behave irregularly and fluctuate strongly due to factors such as disappearing or exploding gradients, which leads to challenges in adapting the adversarial perturbation. In addition many generation algorithms use the signs of the gradients to calculate the perturbations. Using the signs multiple times during the optimization process results in an increasing quantization error, which reduces the performance of the adversarial perturbations.

The authors solved these two challenges by further developing the SPGD algorithm (to see in algorithm 3). The input data is processed in batches ($\boldsymbol{x}^{\mathrm{LB}}$) and for each batch a set of gradients g^{Aggs} is calculated and is then used to adjust δ, which is the UAP. However, the crucial difference lies in the calculation of the gradients. Random elements are taken from each batch and stored in a so-called minibatch ($\boldsymbol{x}^{\mathrm{LB}}$). This is repeated several times until a fixed number of minibatches have been generated. Then the batch delta (δ^{inner}) is calculated for each minibatch and temporarily stored. Afterwards g^{Aggs} is calculated by averaging all the gradients of δ^{inner}. By averaging the gradients, SGA reduces the previous mentioned gradient instability. The quantization error is minimized by the one-time adjustment of δ by g^{Aggs}. SGA runs through several epochs and terminates after the iteration of the last epoch returning the δ as the generated UAP.

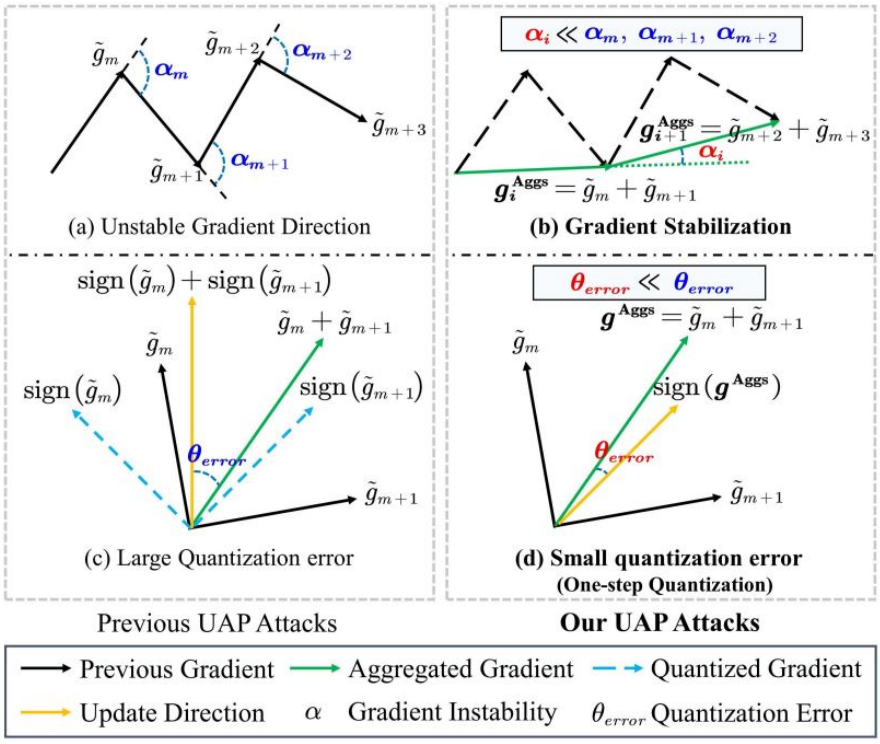

Figure 2.14: Illustration of gradient instability (a) and quantization error (c). SGA solves this issues by adding and averaging the gradients(b) and a one-step quantization(d) [17]

The underlying idea behind the use of minibatches is to achieve the lowest possible gradient instability by averaging the gradients. In the past, large batches were used for this purpose, but it turned out that large batches tend to get stuck in local minima and therefore do not generalize very well [14, 13, 20]. The random sampling of the minibatches addresses the problem of gradient instability, while the small size reduces the probability of converging in a local minimum. The combination of low gradient instability, high generalization and minimal quantization error leads to very powerful UAPs. Both the minibatch size and the number of minibatches per batch are hyperparameters investigated by Xuanna Liu et al. in their paper. With the optimal minibatch parameters, which we specify in the methodology, it is possible to achieve a fooling rate of well over 90% for all common CNN models.

Algorithm 3: The SGA attack algorithm [17]

input : A surrogate model f, loss function $\mathcal{L}$
input : The training image set $\boldsymbol{X}$, large-batch $\boldsymbol{x}^{\mathrm{LB}}$, small-batch $\boldsymbol{x}^{\mathrm{SB}}$
input : Maximum perturbation magnitude ϵ, number of epochs T, step size α
output: A universal adversarial perturbation δ

Initialize $\delta = 0$;
for $t = 0$ *to* $T - 1$ **do**
  for $\boldsymbol{x}^{\mathrm{LB}} \in \boldsymbol{X}$ **do**
    $\delta_0^{\mathrm{inner}} = \delta$;
    $g^{\mathrm{Aggs}} = 0$;
    for $m = 0$ *to* $M - 1$ **do**
      Random select $\boldsymbol{x}_m^{\mathrm{SB}} \in \boldsymbol{x}^{\mathrm{LB}}$;
      $\tilde{g}_m = \frac{1}{|\boldsymbol{x}^{\mathrm{SB}}|} \nabla_\delta \mathcal{L}\left(\boldsymbol{x}_m^{\mathrm{SB}} + \delta_m^{\mathrm{inner}}\right)$;
      $\delta_{m+1}^{\mathrm{inner}} = \mathrm{Clip}_\delta^\epsilon\left(\delta_m^{\mathrm{inner}} + \alpha \cdot \mathrm{sign}\left(\tilde{g}_m\right)\right)$;
      $g^{\mathrm{Aggs}} \leftarrow g^{\mathrm{Aggs}} + \tilde{g}_m$;
    end
    $\delta \leftarrow \mathrm{Clip}_\delta^\epsilon\left(\delta + \alpha \cdot \mathrm{sign}\left(g^{\mathrm{Aggs}}\right)\right)$;
  end
end
return δ.

2.4.2 L_p-Norm

To keep the perturbation as imperceptible as possible, the difference between the original input and the perturbed input must be minimal. In practice, the L_p norm is used to measure this. This mathematical function is defined as follows [33]:

$$\|\mathbf{d}\|_p = (|d|_1^p + |d|_2^p + \ldots |d|_n^p)^{\frac{1}{p}} \tag{2.7}$$

Depending on which value is selected for p {0,1,2...∞}, the difference is calculated using different properties. For example if choosing $p = 1$, this is known as the L_1 norm. This calculates the sum of all absolute vector values, in an image this is the sum of all pixel values. If you subtract the L_1 norm of the two images, you get the sum of the pixel differences. However, this value is not particularly informative, which is why the L_1 norm is only used in very special application areas [29]. The L_2 norm (also known as the Euclidean norm), which calculates the distance of a vector in space, is much more common. By imagine the input images as two points in multidimensional space, the L_2 norm is the distance between the two images. At pixel level, this is the average difference of all pixel values. Together with the L_∞ norm, which measures the maximum change of a single pixel, the L_2 norm is the most used one. In the case of adversarial perturbations, the L_p norm is used to limit the perturbation

and therefore to limit the change in the image. The hyperparameter ϵ specifies the maximum value for the L_p norm. With universal adversarial perturbations, it is common to use the L_∞ norm, which is why all ϵ specifications in this work refer to the L_∞ norm.

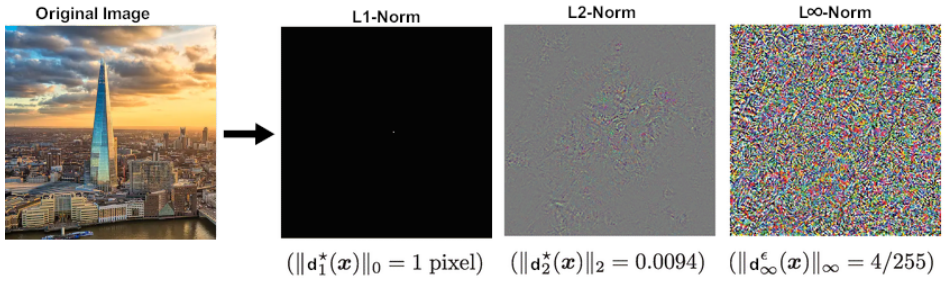

Figure 2.15: Visualization of the adversarial perturbations depending on the different L_p norms

2.5 Defenses

Over the years, numerous approaches have been developed to increase the protection against adversarial perturbations. The spectrum ranges from preprocessing measures to advanced architectures that are specifically designed to detect perturbations. All of these protective measures can be assigned to one of three categories. For this work, we have chosen at least one defense from each category to make the result as representative as possible.

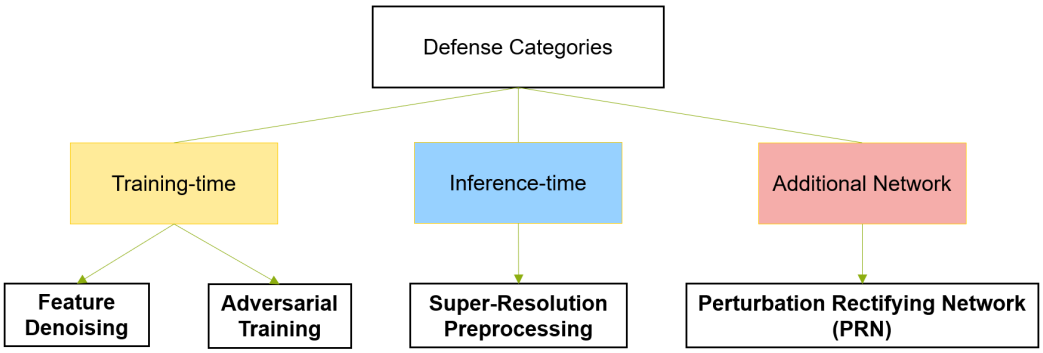

Figure 2.16: Overview of the defense categories for adversarial attacks. we have selected at least one defense from each category

The training-time category includes all defenses that aim to increase the robustness of a model against adversarial attacks during the training process. This includes

changes to the model architecture or to the training itself. Inference-time, on the other hand, includes the defenses that are active during the execution of the model. These could be for example pre- or post-processing measures. In the third category, an additional DNN is trained that recognizes or eliminates adversarial perturbations. The DNN can be integrated into the target model or into the preprocessing pipeline. A combination of defenses is not mutually exclusive. However, it is important to mention that the combination of individual defenses does not always make sense, especially if individual defenses have vulnerabilities. These can be exploited by an attacker, making the entire construct insecure.

2.5.1 Adversarial Training

This is the most intuitive way to protect models against adversarial attacks. Adversarial training is a protective measure that is conducted during model training and helps the model to better deal with adversarial inputs by adapting to perturbed inputs during the training process. For this purpose, adversarial perturbations are continuously calculated during training and integrated into the training data. Paired with the correct label of the input image, the model processes the perturbed input and learns to classify the image correctly. This approach was first introduced by Christian Szegedy et al. [30] and has been continuously developed since then [6]. Over the years, numerous variations and optimizations have been proposed for adversarial training, but the basic principle has remained the same. Adversarial Training serves as the foundation for other defenses, such as feature denoising.

2.5.2 Feature Denoising

As already described in the chapter on CNNs, feature maps extract distinctive features from the input image. The CNN learns to distinguish important image areas from unimportant ones. However, this differentiation is influenced by adversarial perturbations. Cihang Xie et al. were able to show that perturbed inputs lead to noisy feature maps [38]. Therefore areas that are irrelevant for the classification of the image are activated by the perturbation. In addition, the activation of important feature areas is reduced in some cases. From this it can be concluded that the adversarial perturbation of an image significantly influences the feature extraction of feature maps. To minimize the influence of the perturbations, Cihang Xie et al. introduced a new architecture, which is trained to eliminate the noise of the feature maps[38]. To achieve this, they use several denoising blocks that are integrated into an existing CNN architecture. In the example of a Resnet architecture, each Residual block is followed by a denoising block. Similar to the super-denoising procedure [23], the basic idea is to eliminate image noise. While super-denoising preprocessing

is used exclusively for input images, the denoising block operates at feature map level.

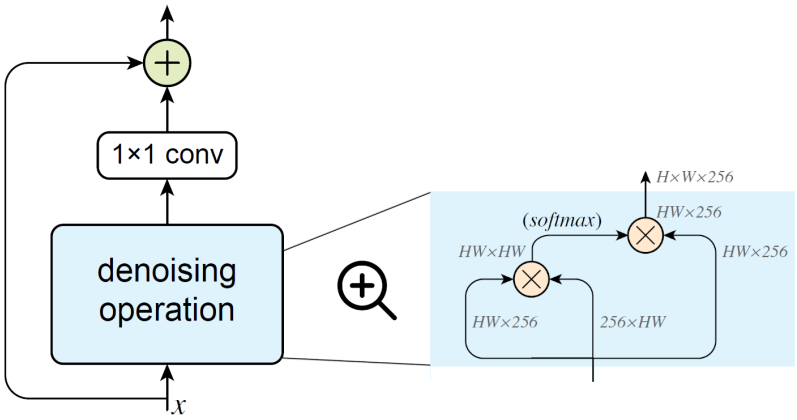

Figure 2.17: Detailed view of a denosing block [38]

The denoising block first processes the feature maps using a denoising operation (non-local means) to reduce the noise. The denoised feature map is then passed through a convolutional layer and then combined with the original feature map using a residual connection. The convolutional layer and the residual connection are used for feature combination and signal retention. Since the denoised feature maps are visibly different from the original ones (figure 2.18), it is necessary to train the entire model from scratch so that it learns to classify these feature maps correctly. The training is done in the form of adversarial training.

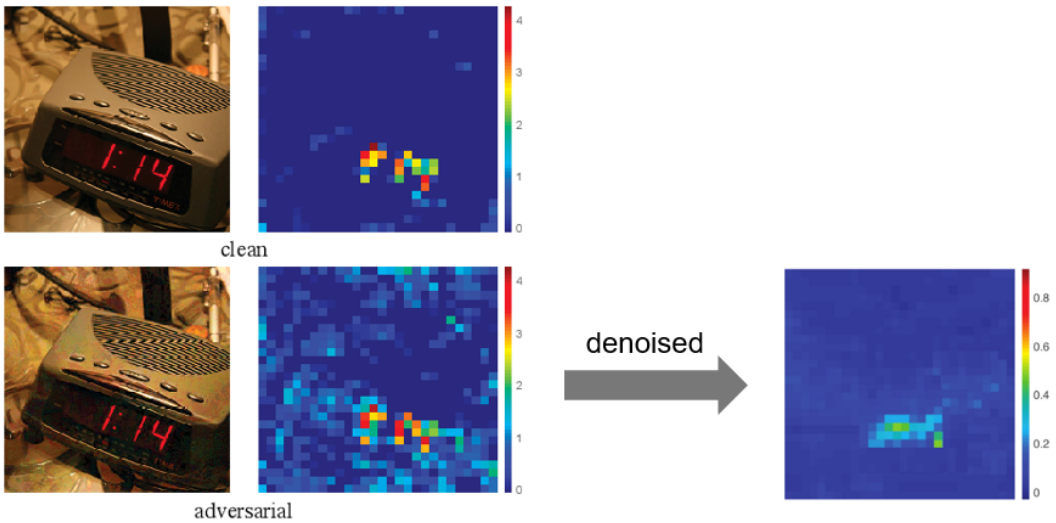

Figure 2.18: Comparison between a regular and an adversarial feature map. The corresponding denoised feature map is on the right side [38]

2.5.3 Image Super-Resolution Preprocessing

An interesting and effective method to remove adversarial perturbations by image preprocessing was presented by Aamir mustafa et al. [23]. The basic idea is to process the input image before it is classified by the DNN in such a way that most of the perturbations are removed, making them ineffective. Since CAPs are often embedded in the image in the form of noise, the preprocessing technique focuses on noise removal in the input image. Because this defense method is neither model-dependent nor in need of training, it can be used in any environment and is therefore highly flexible. This approach is not new. In the past, numerous image preprocessing techniques have been published to protect against CAPs. These include JPEG compression [18], feature squeezing [39] and pixel deflection [24], although Image Super Resolution is the most effective of these, as shown by Aamir Mustafa et al. It is important to note that they only evaluated on CAPs. Universal adversarial perturbations were not considered in the evaluation. This is why we will analyze the performance of Super Resolution Preprocessing as a protective measure against UAPs. The preprocessing is performed in 2 stages. At the beginning, the noise in the image is reduced using wavelet denoising. For that, the image is broken down into different frequency ranges. Particularly high-frequency parts that contain noise are removed or smoothed. The low-frequency image areas are retained, as these generally contain the basic image structures. A particularly important parameter for wavelet denoising is Sigma, which specifies the threshold value at which frequency the areas are smoothed. The higher the sigma, the more aggressive is the noise suppression. However, a high sigma leads to a decrease in image quality, so the aim is to find a balance between noise reduction and the preservation of important image structures.

The central processing step takes place after wavelet denoising. It is based on the assumption that high-dimensional data (in this case images) is not randomly distributed in high-dimensional space. In addition it assumes that there is a certain similarity between natural images and that they can therefore be represented in a low-dimensional space (Manifold assumption) [42]. The similarity of natural images can be explained by feature correlation. Images from a natural origin usually have many features in common, including edges, textures and colours. The more structures and features they share, the closer two images are in high-dimensional space. Assuming that naturally generated images share many common structures, it can be concluded that they lie in a cluster within the high-dimensional space and can therefore be represented by lower dimensions. The spatial proximity of natural images explains the generalization ability of DNNs, which are trained with a relatively small dataset but still perform well on unknown inputs. The DNN learns the spatial structure of the natural inputs and assigns them to different classification areas. If an image is perturbed with an adversarial perturbation, it moves outside the natural cluster with a high probability of being misclassified by the DNN. In the second pre-processing step, this displacement is reversed so that the image returns to the original classification area of the DNN.

This process can be formally described as follows:

$$\begin{gathered} \mathcal{N} \text{ Set of natural images} \\ \forall x \in \mathcal{N} : (x + \delta) \notin \mathcal{N} \\ \forall x \in \mathcal{N} : f(x + \delta) \in \mathcal{N} \end{gathered} \tag{2.8}$$

2.8: shifting the adversarial inputs back into the set of natural images, by using a super resolution network f

The backshift of the image is not trivial and can only be approximated in most cases, as it is not possible to calculate the inverse of the perturbation without the unaltered original image. Therefore, a super-resolution network is used in the second step of the preprocessing procedure. Super-resolution networks are CNNs that are used to increase the resolution of an image. Depending on the architecture, an image can be enlarged by a multiple without any noticeable loss of quality. In other words, the super-resolution network learns how to map low-dimensional inputs to high-dimensional outputs. As the network was trained with natural images, it can be assumed that the high-dimensional outputs are located in the cluster of natural images. This property is used to move the perturbed image back into the natural cluster. So, during preprocessing, the image is scaled by the super-resolution network. This process removes a significant amount of adversarial perturbation as unnatural noise is lost during the scaling process. The output is an enlarged image that is largely cleared of unnatural noise. The image is then reduced to its original size using a simple interpolation process so that it can be used as input for the target DNN. Reducing the size only changes the dimensionality of the image, but not its position in space. It can therefore be assumed that the shift, away from the perturbation towards the natural images, is maintained. Aamir Mustafa et al. showed in their paper that the shift is large enough to neutralise most adversarial perturbations.

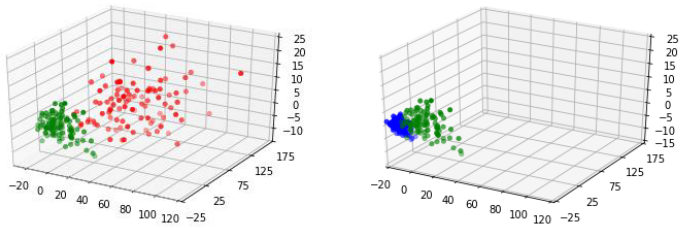

Figure 2.19: 3-D representation of the multidimensional space. The adversarial inputs (red) lie outside the natural cluster (green). After the shift through the super resolution network, the adversarial inputs are moved back into the natural cluster (blue) [23]

To illustrate the denoising process, we added random noise to an image and then denoised it using the described preprocessing technique (figure 2.20). It can be clearly seen that the unnatural noise is well removed. This is essential for the neutralisation of adversarial perturbations, as the misclassification is caused by these fine nuances. In contrast to conventional adversarial perturbations, UAPs are not characterised by a noise-like structure. In most cases, UAPs have a certain pattern that is more or less structured depending on the target model and the data set. High-frequency noise can only be found in small areas, if at all. This raises the question of whether noise removal is also effective for UAPs.

Figure 2.20: Denoising performance of the super resolution preprocessing on artificial noise. The original image on the left, the artificially noised image in the middle and the denoised image on the right

2.5.4 Perturbation Rectifying Network

All the defenses presented so far have been developed and evaluated for CAPs. There are only a handful of publications that deal with defenses against UAPs. A promising method was developed in 2018 by Naveed Akhtar et al. [3]. They introduced the concept of perturbation rectifying networks (PRNs). A PRN is a CNN that is placed in front of the input layer of the target DNN to neutralise the UAPs in the input. The filtered image is then passed to the actual DNN, which is no longer influenced by the UAP. Figure 2.21 shows the structure of a PRN architecture. On the left-hand side are the input images that are to be classified by the DNN. They pass through the PRN and are rendered harmless if they contain a UAP. The output is then passed as input for the DNN. A decisive advantage of the PRN is its modularity. In contrast to other defense measures, the PRN is not invasive as it only serves as an input filter for the DNN. It therefore does not change the DNN and can be replaced without any problems. In addition, the PRN can be trained on any dataset and UAP classes so that a high degree of flexibility is also possible here.

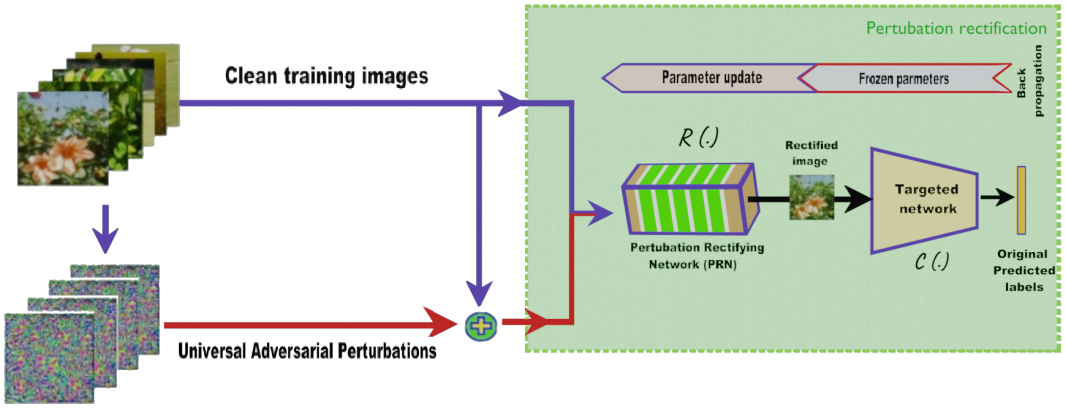

Figure 2.21: Overview of the PRN architecture and the training process [3]

The PRN consists essentially of 3 main components. The input layer is a convolutional layer, followed by 5 residual network blocks that form the core of the PRN. A residual network block (Resnet block) is a compact CNN that is particularly suitable for training deep architectures, as it prevents training problems such as vanishing gradients. The output layer of the PRN consists of 2 convolutional layers. It should be noted that both the convolutional layers and the Resnet blocks are configured in such a way that the input image retains the same dimensionality after being processed by the PRN. This means that the output can be used smoothly as input for the following DNN. To train the PRN, it is necessary to have access to the target DNN in order to calculate the gradients of the DNN's output all the way to the PRN's input layer using backpropagation. Only the weights of the PRN are adjusted, the parameters of the DNN remain frozen in order to not change the model. During training, approx. 50% of the inputs are perturbed with UAPs so that the PRN learns to deal with both normal and adversarial inputs. The training data is processed in batches and saved as a rectified batch, with each rectified batch having a corresponding clean batch containing the unchanged input images without UAPs. The two batches are then classified by the target DNN and the output is saved. The aim of the training is to minimise the following cost function:

$$\mathcal{J}\left(\boldsymbol{\theta}_p, \mathbf{b}_p\right) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(\ell_i^*, \ell_i\right) \tag{2.9}$$

ℓ_i^* and ℓ_i are the outputs of the DNN. ℓ_i^* stands for the prediction of the rectified batch and ℓ_i for the prediction of the clean batch. For all training batches, the loss function is calculated using the two outputs ℓ_i^* and ℓ_i. The loss value indicates how accurate the reconstruction of the rectified batch is compared to the clean batch. Based on this, the gradients for the parameters $\boldsymbol{\theta}_p$ and $\mathbf{b}_p$ are calculated. This process teaches the PRN to process the input images in such a way that the output is as similar as possible to the original unaltered images.

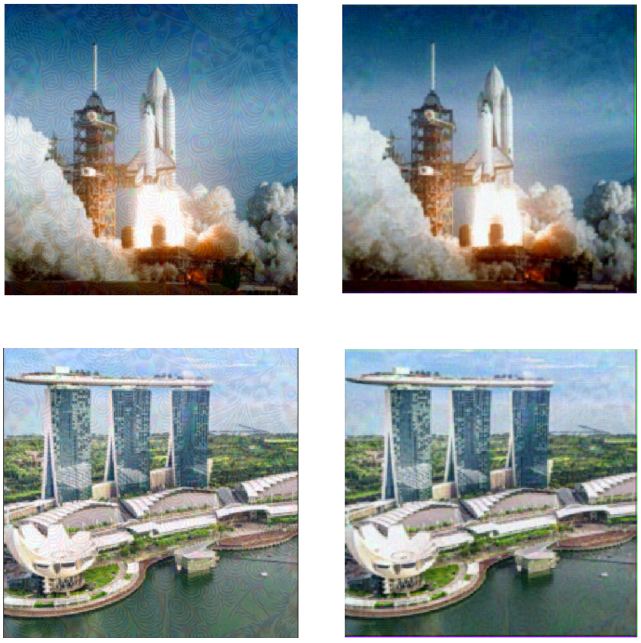

Figure 2.22: Comparison between images perturbed with SGA UAPs (top) and the output of the PRN, the rectified images (bottom). To see the differences, we recommend to view the images in color and enlarged

3 Methodology

In this chapter, we will describe how we implement the Defenses, which Models and Dataset we selected and how the UAPs are calculated. It provides an overview of our general approach and the evaluation metrics that we will use for the evaluation in chapter 4.

3.1 Overview

We evaluate all defense measures on three different models, each on a different dataset. As mentioned above, SGA UAPs are proven to be the most effective UAPs, so we chose these for evaluation. For better comparability to other papers, we also evaluate with Deepfool UAPs, as these have been used in the most evaluations in the past. Since the generation of SGA-UAPs and Deepfool-UAPs differs fundamentally (see above), a difference in the performance of the defenses cannot be ruled out and is therefore another point of our evaluation. If the implementation of a defense requires training, it is performed using PGD-10 perturbations. We opted for PGD-10 because it offers a good ratio between performance and computational effort and is one of the most effective CAPs. All models and algorithms are implemented in Pytorch.

3.2 Models and Datasets

To evaluate the defense methods, we will use three different Resnet models that have been trained on various data sets. In this way, we ensure that influencing factors such as the model architecture or the training data do not bias the evaluation. We chose Imagenet, CIFAR-100 and CIFAR-10 as datasets [11, 15]. Imagenet is one of the most comprehensive and popular datasets for image recognition tasks. It contains 1.2 million images grouped into 1000 categories. The dimension of the images is not uniform, however in practice they are scaled to 224×224 pixels. The evaluation on Imagenet serves as a benchmark and comparison value with other scientific papers. CIFAR-100 and CIFAR-10 are also widely used datasets for image classification, each consisting of a collection of 60,000 colour images splitted into 10 and 100 classes respectively. Compared to Imagenet, the images have a significantly smaller dimension of 32×32 pixels.

All three datasets contain a training set and an evaluation set. The generation of the UAPs and the training of the defenses is performed on the training data set. All evaluations and performance measurements are conducted on the complete evaluation data set.

We use Resnet50 to classify the Imagenet data. It is a powerful CNN that can be pre-trained and loaded into PyTorch.

For datasets with low image dimensions such as CIFAR-10/100, it has been shown that wider architectures outperform deep ones [40]. Especially well suited is the wide resnet architecture that we use to evaluate CIFAR10/100. We train a wide-resnet 34-10 for CIFAR-10 and a wide-resnet 70-16 for CIFAR-100. We train both models for 200 epochs with a learning rate of 0.1 and a decay of 90% after every 60th epoch. This way we achieve a clean accuracy of 89.86% for CIFAR-10 and 65.09% for CIFAR-100. These models are the basis of our implementation. We call them "Clean Models" in the following.

3.3 Evaluation Metrics

In order to evaluate the performance of defenses against (universal) adversarial perturbations, the following three metrics are generally used in the literature:

Clean Accuracy

Measures the prediction accuracy of a model on the normal, non -perturbed dataset. The higher the clean accuracy, the better the performance of the model on normal inputs. A good defense should not negatively influence this metric, otherwise the model performance will decrease.

$$\frac{1}{n} * \sum_{i=1}^{n} f\left(x_i\right) = label_i \tag{3.1}$$

Adversarial Accuracy

Measures the prediction accuracy of a model on a perturbed dataset. If the adversarial accuracy is low, many images are classified incorrectly. With an optimal defense, the Adversarial Accuracy is equal to the clean accuracy. This metric is often referred to as robust accuracy

$$\frac{1}{n} * \sum_{i=1}^{n} f\left(x_i^{adv}\right) = label_i \tag{3.2}$$

Foolrate

Measures the percentage of label flips on a perturbed dataset. The better the perturbations are neutralized by the defense, the lower this value is.

$$\frac{1}{n} * \sum_{i=1}^{n} f\left(x_i^{adv}\right) \neq f\left(x_i\right) \tag{3.3}$$

We use all three metrics during the evaluation process (denoted as CA, AA and FR), in order to achieve a reliable and meaningful result. This allows us to determine how well a defense neutralizes (universal) adversarial perturbations and how the performance of the model is influenced by the defense.

3.4 UAP Generation

For each dataset, we generate SGA and Deepfool UAPs. Based on the original paper, we chose the hyperparameters to generate the UAPs. Xuannan Liu et al. have investigated the difference in performance of various hyperparameters to generate SGA UAPs [3]. Based on this, we decided to use the parameters listed in 3.1. Moosavi et al. have also specified their parameters in their paper on Deepfool UAPs [22]. We have adjusted these slightly, allowing the algorithm to converge much faster with the same performance. A peculiarity of the Deepfool algorithm is the parameter End foolrate, which specifies the desired fool rate of the UAP. If this is reached, the algorithm terminates immediately. Therefore, unlike SGA, Deepfool can terminate in less than 20 epochs. With UAPs, it is common to use the L∞ norm as the measured value for the image change. Therefore, all Epsilon parameters are to be interpreted as L_∞ values.

Parameter	SGA	Deepfool
Epsilon	10/255	15/255
Learnrate	0.1	0.7
Epochs	20	20
Batch size	250	-
Minibatch size	10	-
End Foolrate	-	0.8

Table 3.1: Overview of the parameters used to generate the UAPs

Each UAP was generated with a total of 10,000 different images from the respective training dataset. If we would generate the UAPs on the images of the evaluation dataset, they would be specially adapted to them and would falsify the evaluation. To proof this, we generate some SGA UAPs using the evaluation dataset and compare the performance in table 3.2. It can be seen that the UAPs generated on the evaluation images have a higher performance, which is a result of the adaptation to the data.

Dataset	SGA on train data	SGA on eval data
Average FR	93.39%	96.6%
Average AC	6.23%	3.11%

Table 3.2: Comparasion between UAP performances generated on the training or evaluation data. Experiment conducted on Imagenet, calculating 10 different UAPs and averaging their performance.

For Imagenet, we selected 10 random images from each class. For CIFAR 10/100, we selected 1000 / 100 images from each class, so that 10,000 images were also used for the generation. Since the CIFAR datasets are considerably smaller than Imagenet, it can be assumed that the UAPs could also be generated with fewer images. For reasons of consistency, we decided to use the same number of images as generation data for all datasets. As the generation of UAPs is a stochastic process (choice of images, composition of the minibatch, etc.), we only selected UAPs for the evaluation that have a certain performance. For this purpose, we calculated the average foolrate of all SGA/Deepfool UAPs and defined a minimum foolrate on this basis.

Dataset	SGA	Deepfool
CIFAR-10	$> 80\%$	$> 80\%$
CIFAR-100	$> 90\%$	$> 80\%$
Imagenet	$> 90\%$	$> 80\%$

Table 3.3: Overview of the minimal foolrate for each dataset. UAPs with a lower foolrate were discarded.

The foolrate is consistent through all three datasets except for the SGA UAPs with CIFAR-10. Without adjusting the generation parameters, which would falsify the evaluation, the SGA UAPs on CIFAR-10 are not as effective.The reason for this is unclear. We limited the similarity between the UAPs to create the evaluation set as divers as possible. SGA UAPs in particular are relatively similar to each other (Figure 4.1). Many similar UAPs in the evaluation dataset can influence the results. The similarity between two UAPs can be calculated using the scalar product.

The scalar product between two vectors $\vec{a}$ and $\vec{b}$ (in this case images) can be defined as follows:

$$\vec{a} \cdot \vec{b} = \sum_{i=1}^{n} a_i \cdot b_i \tag{3.4}$$

The maximum scalar product results between two identical UAPs and is between 187-192 for SGA and between 197-234 for Deepfool. We systematically calculated the scalar product between all UAPs and discarded those with a scalar product >10% of the maximal value. By ensuring a high level of diversity, a small amount of UAPs is sufficient to obtain reliable results. For each dataset, we generated a total of 400 UAPs of each sort. Of these, 320 are for training purposes and 80 for the evaluation (80/20 split). During the evaluation, we assigned one of the 80 UAPs to each image. The assignment was random. Depending on the constellation, the result of the evaluation varies minimally. Therefore, to be on the safe side, we evaluated each defense 5 times and calculated the average of the metrics. However, the differences between the individual runs were negligible (0.1-0.5%), so one run is sufficient for future evaluations.

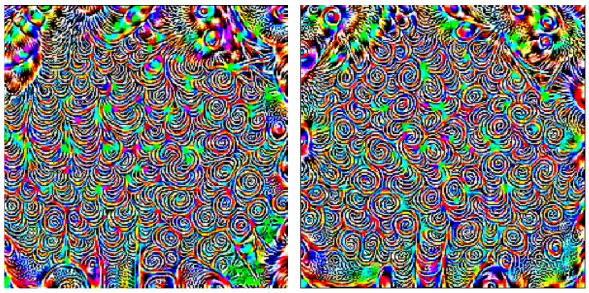

Figure 3.1: Two different SGA UAPs with a high similarity score

3.5 Defense Implementation

All defenses except for the Perturbation Rectifying Network were implemented in such a way that they achieve the highest possible performance against CAPs. Based on the results of the original paper and our own experiments, we have found optimal parameters to maximize the performance of the defenses against CAPs for the previously mentioned models. We deliberately avoided adapting the defenses to UAPs in advance (e.g. adversarial training with UAPs) in order not to falsify the evaluation.

Adversarial Training

We evaluated adversarial training with models from the Robust Bench platform [10]. Robust Bench is a benchmarking initiative that aims to measure the robustness

of adversarial trained models. It provides access to these models and lists their performance, which has been evaluated in a standardized way. For each dataset, we filtered by the architectures mentioned above and selected the model with the highest performance. By choosing the same architecture, the consistency of our evaluation is ensured. We first tested the selected models on CAPs to ensure the performance indicated by the paper. The evaluation on UAPs was performed on the evaluation dataset according to the procedure described above.

Super-Denoising Preprocessing

Super-denoising preprocessing requires no training and no adaptation of the model. Therefore, we were able to use the clean models and insert the preprocessing step into the data pipeline before the model. As in the original paper, we use wavelet denoising to reduce the noise of the input. The only relevant parameter, Sigma, controls the amount of denoising. We successively evaluated all values between 0.03 and 0.08 and found 0.04 to be the optimal value against CAPs. This is consistent with the results of the original paper. For the super resolution network, we use the enhanced deep super-resolution network (EDSR) proposed in the paper [16] which is pre-trained on Imagenet data. We have implemented a total of three EDSR models, each scaling to a factor of $\times 2$, $\times 3$ and $\times 4$. We achieved the best performance with the $\times 2$ EDSR and used it therefore for the evaluation.

Feature Denoising

For reasons of computing capacity, we implemented feature denoising exclusively on CIFAR-10 and CIFAR-100. The training for Imagenet is simply too time-consuming to be possible within the time frame of this thesis. Feature denoising is implemented using a modified model architecture, which we call "Denoise Model" below. The Denoise model consists of the same architecture as the Clean model but also contains some Denosing Blocks. We have inserted these after each regular block. The Denoise model is trained with adversarial training from scratch. To do this, we use the same parameters as for training the clean model. For each clean batch, we calculate an adversarial batch which contains the PGD-10 perturbed images. In each training iteration, we calculate the respective loss values of the clean and adversarial batches. These are then added together and averaged so that the adjustment of the weights takes into account the loss value of both batches.

Perturbation Rectifying Network

We constructed the Perturbation Rectifying Network using the same architecture as proposed in the original paper. The PRN is the only defense in this paper that was developed specifically for UAPs, so the training was also performed with UAPs. In

order to train a PRN, sufficient training data and UAPs must be available. In the example of Imagenet, we have chosen 200 random images from each class. With 1000 classes, this results in 200,000 images. We have calculated the UAPs in advance, which speeds up the training process considerably. We use cross-entropy loss as loss function and use the ADAM optimizer with a learning rate of 0.001. The learning rate is reduced by 10% after every 1000th iteration. We train for 5 epochs with a batch size of 50. In our experiments, more epochs did not lead to any significant increase in performance. We train several PRNs with the same parameters, but change the number of training UAPs. In this way, we want to determine how many different UAPs are required for training an PRN in order to ensure good performance. We train the PRNs on homogeneous UAPs. In other words, exclusively on Deepfool or exclusively on SGA. However, the evaluation is carried out on both UAP types. This allows us to evaluate how well a trained PRN can generalize to other UAP types. Naveed Akhtar et al. have evaluated the PRN exclusively on Deepfool UAPs. Therefore, our evaluation provides new insights about the performance of PRNs against newer and more powerful UAPs. Since all previous defenses were exclusively designed for and evaluated on CAPs, we also want to train and evaluate the PRN with CAPs in our experiments. The procedure is not that different from the one described above, except that CAPs are used instead of UAPs during the training.

3.6 Thread Model

This section deals with the interplay between the target model and the attacker. The threat model describes which goals the attacker is pursuing, which points of contact he has with the model and how he implements the attack to achieve his goal. To describe a threat model as precisely as possible Giovanni Apruzzese et al. have defined the criteria: Goal, Knowledge, Capabilities and Strategy[5]. We use these to determine the thread model for our own experiments.

Goal: The goal of the attacker is to (negatively) influence the predictions of a model. To do this, he calculates (universal) adversarial perturbations, which are integrated into the input and therefore become adversarial inputs. They can hardly be distinguished from the original input, but are classified differently. Untargeted perturbations lead to an arbitrary misclassification of the input, whereas targeted perturbations force a specific class. UAPs are normally calculated untargeted, which is also the case in this work.

Goal

The goal of the attacker is to (negatively) influence the predictions of a model. To do this, he calculates (universal) adversarial perturbations, which are integrated into

the input and therefore become adversarial inputs. They can hardly be distinguished from the original input, but are classified differently. Untargeted perturbations lead to an arbitrary misclassification of the input, whereas targeted perturbations force a specific class. UAPs are normally calculated untargeted, which is also the case in this work.

Knowledge

Depending on what knowledge the attacker has about the target model, the CAP/UAPs are calculated in different ways. Usually, a distinction is made between white box and black box. These concepts describe the access and knowledge that an attacker has about a model. In white-box attacks, the attacker has complete access to the target model and therefore knows all the details about the architecture, weights and other parameters. This is the most powerful attacker and is often used to test the robustness of models. In a black box setting, the attacker has no information about the target model and can only learn about its functionality through interaction. A distinction is made between limited black box and score-based black box. In both variants, the attacker can interact with the model (pass input, receive output), but they differ in the amount of information contained in the output. In Limited Black Box setting the attacker only has access to the final value of the output (e.g. the class of the input), whereas in Score Based Black Box the attacker has access to the entire prediction values (e.g. confidence value for each class). In this paper, I will only use the white-box setting, as it is the most powerful way to assess the robustness of models and their defenses.

Capabilities

The capabilities of an attacker are strongly related to the knowledge. In the case of UAP generation, it is not necessary to change the parameters of the model or its training data. It is sufficient to have a reading access to the model and its training data. In addition, the high transferability of UAPs allows them to be trained on another model that has a similar architecture and was trained with similar data.

Strategy

Using the gradients, the attacker solves the optimization problem from described in Formula (2.3). As already mentioned in the chapter on adversarial perturbations, there are various approaches (one-step, iterative, etc.). Our UAPs are generated with the gradients using iterative algorithms.

4 Evaluation

In this chapter we will present and analyze the results of our experiments. In addition, we will examine the protective behavior of defenses depending on their parameters and configurations. We will answer our research question using the data collected as well as design and test the best possible defense against both CAPs and UAPs. Our evaluation is conducted dataset wise. We will discuss the performance of the individual defenses within a dataset and then look at the differences between the individual datasets. There are numerous differences between Imagenet and CIFAR10/100 in particular, which we will then try to explain. We also conducted our evaluation Defense wise and illustrated it graphically, making it easier to compare the them with each other. The graphics can be found in the appendix.

4.1 CIFAR-10

	Attack		None	Adv. Training	Feature Denoising	Super Resolution	PRN UAP trained	PRN CAP trained
CIFAR-10	Clean	CA	89.86	**91.47**	88.91	81.55	85.23	84.74
	PGD-10	AA	6.54	65.3	53.83	8.48	14.06	**80.04**
		FR	100	39.54	52.53	97.97	92.48	**17.91**
	SGA	AA	15.08	**90.26**	88.02	13.53	83.8	78.6
		FR	84.90	**2.89**	3.05	86.31	12.52	19.14
	Deepfool	AA	19.05	**89.71**	87.5	15.00	82.72	74.34
		FR	80.83	**3.72**	4.13	85.22	13.64	23.71

Table 4.1: Evaluation results for CIFAR-10

We start the evaluation with the smallest dataset and work our way up to the largest one. The first look at the evaluation table 4.1 reveals that the use of no defense leads to fatal performance drops, regardless of whether CAPs or UAPs are used. This confirms the high effectiveness of adversarial attacks. It is striking that the adversarial accuracy is particularly high in relation to the 100% foolrate for the PGD attack. This is a phenomenon that can be observed in both CIFAR datasets. At first glance, this seems contradictory, as the adversarial accuracy behaves antiproportionally to the foolrate, but the explanation is relatively simple. Since the foolrate measures the

number of label flips, 100% means that the PGD-10 perturbation successfully changes the output class of each image. For a dataset with only a few classes, the probability is relatively high, that images that were initially misclassified by the DNN will be mapped to the correct class by the label flip. This is the reason for the adversarial accuracy of 6.54 %. In general, it can be said that any defense apart from the super resolution preprocessing reduces the effectiveness of CIFAR-10 UAPs. Especially adversarial training shows a particularly good performance. It not only increases the base accuracy of the model, it also provides an exceptionally high protection factor against UAPs, reducing the foolrate to under 4%.

The Feature Denoising shows similar results, although the training is much more complex and time consuming. As expected, the perturbation rectifying network offers a high protection factor against UAPs and delivers a high clean accuracy of 85.23%. Therefore, it can be concluded that the performance of a PRN does not depend on the size of the input images and that even small input images can be reconstructed effectively. However, the performance against CAPs is not optimal and in a very low range, reducing the foolrate from 100% to only 92.48%.

To improve this, we try to train the PRN with CAPs instead of UAPs. To our knowledge, we are the first ones to do this. By doing so, we discovered a previously unknown property of the PRN, which is its outstanding performance against CAPs when properly trained. With our training, we achieve an adversarial accuracy of 80% for PGD-10 perturbations and 78% / 74% for UAPs. In fact, the CAP-trained PRN protects by far better against CAPs than adversarial training. This is a very strong performance for CIFAR-10, however the true potential of the PRN becomes more apparent with the more complex datasets, so we will address this topic in a later part of the evaluation.

The worst of all CIFAR-10 defenses was the Super resolution preprocessing. Although it offers a minimal protection against CAPs, it seems to increase the negative effect of UAPs leading to a lower performance than the clean model with UAPs. The protection factor of the super resolution preprocessing depends on the selected *sigma* value. The values in the table refer to $sigma = 0.04$. The protection factor tends to increase with a higher *sigma*, while the clean accuracy decreases. This can be explained by the fact that a strong denoising of the image will eliminate a large part of the perturbation, but important image elements that the DNN needs to make a reliable prediction will also be lost during the process. Therefore, the challenge is to find a suitable *sigma* value, that ensures a high protection performance on the one hand and does not affect the clean accuracy too much on the other. We therefore adjusted the hyperparameter *sigma* during our experiments, but observed that no significant improvement could be achieved. Figure 4.1 shows the tradeoff between clean accuracy and adversarial accuracy. Although a high *sigma* leads to better adversarial accuracy, it reduces the clean accuracy so much that it is not a useful defense in practice. The poor performance of the Super resolution preprocessing can be observed in both CIFAR datasets, so we suspect that the Super

resolution network and the denoising function cannot handle the small CIFAR image size.

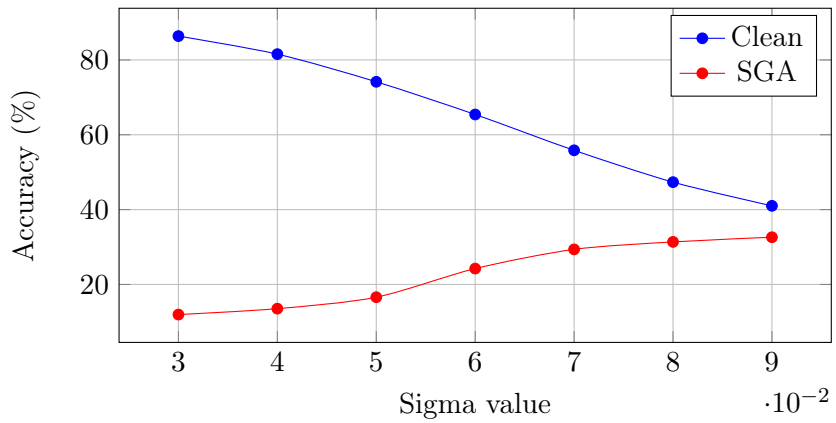

Figure 4.1: Super resolution preprocessing: accuracy tradeoff clean and adversarial. Performed on CIFAR-10.

4.2 CIFAR-100

	Attack		None	Adv. Training	Feature Denoising	Super Resolution	PRN UAP trained	PRN CAP trained
CIFAR-100	Clean	CA	65.09	**69.15**	67.39	55.02	60.64	59.51
	PGD-10	AA	10.46	43.86	30.31	11.91	13.77	**54.44**
		FR	100	61.93	78.99	98.05	96.59	**38.09**
	SGA	AA	7.09	**68.25**	66.18	18.44	58.23	45.86
		FR	92.32	**6.49**	9.98	79.76	29.49	48.14
	Deepfool	AA	15.2	**67.61**	65.34	28.24	55.19	44.42
		FR	83.2	**8.83**	12.68	68.65	35.26	50.02

Table 4.2: Evaluation results for CIFAR-100

The evaluation of CIFAR-100 shows similar results to CIFAR-10. The clean accuracy of 65.09% is lower in direct comparison, which is due to the higher complexity of the data set. Therefore, the accuracy of the defenses should not be compared in absolute terms but relative to the clean accuracy of the base model. For CIFAR-100, adversarial training also provides a good performance. Although the protection against CAPs is only mediocre, it is very high for UAPs, outperforming with 68% and 67% the clean accuracy of the base model. Adversarial training is therefore particularly suitable for CIFAR datasets.

Feature denoising offers a similar protection factor as adversarial training, but does not outperform it on any metric. Since we could also observe this result with CIFAR-10, we can conclude that the extra complexity during training is not worth it compared to adversarial training.

As already pointed out, the Super Resolution Preprocessing has especially great difficulties with the CIFAR data sets. For CIFAR-100, the protection factor against UAPs is with a foolrate of 18 % and 28% very low but higher than against CAPs (12%) . Compared to CIFAR-10, the performance is slightly better, but still far from being optimal. Therefore, the preprocessing is not suitable as a reliable protective measure.

In contrast, the next defense in our evaluation, the PRN, is very reliable. Even for this data set with a relatively high complexity and a low image dimension, the PRN achieves a good performance. Although the UAP-trained PRN is not as effective as adversarial training, it still provides an high level of protection against UAPs. For PRNs that have been trained with CAPs, the protection factor against UAPs is reduced, but it increases massively the protection against CAPs. However, the protection against UAPs is not optimal, which is why we have gone one step further and optimised the training even more. To do this, we trained the PRN with both CAPs and UAPs and evaluated it in an additional experiment. The results are represented in figure 4.2.

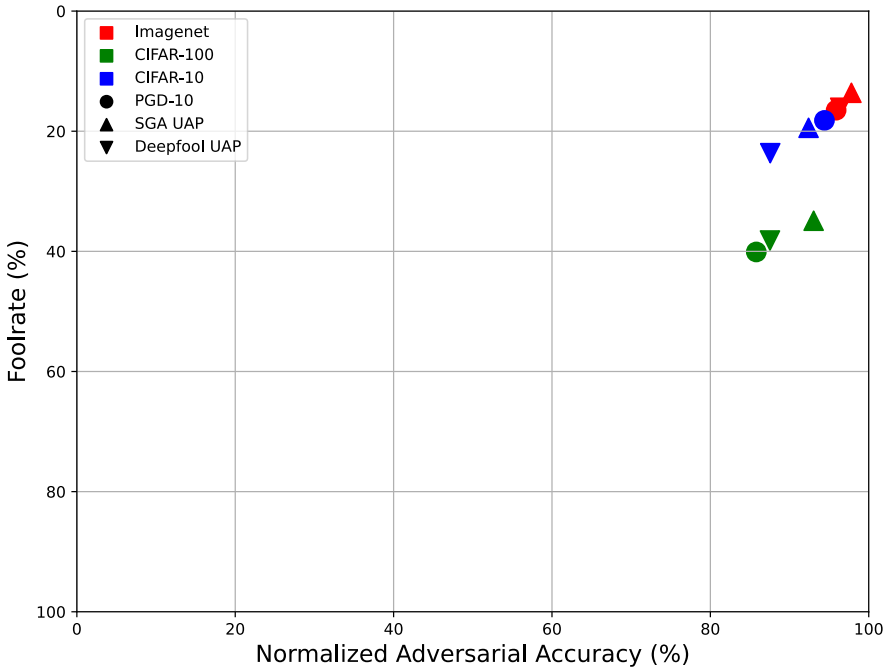

Figure 4.2: Defense performance of the perturbation rectifiying network trained with both CAPs and UAPs

The shapes within the coordinate system stand for the different attacks and the colours for the respective datasets. The X-axis is the adversarial accuracy relative to the clean accuracy. A value of 100 therefore indicates that the adversarial accuracy is just as high as the clean accuracy. We have chosen this metric in order to set the performance of the defenses in relation to the clean accuracy. This is because the datasets have different clean accuracies, which would bias the direct comparison, as CIFAR-100, for example, is relatively low and CIFAR-10 relatively high. In order to visualize the performance of the PRN properly, it is therefore necessary to set the values relative to the clean accuracy. Since the PRN itself already has a very high clean accuracy, this is a meaningful evaluation metric. The Y-axis represents the foolrate as we know it. The further a data point is in the upper right corner, the better the defense performs, as the foolrate is minimized and the accuracy is maximized. It is clear to see that our specially trained PRN performs extraordinarily well against all evaluated attacks on every single dataset. The PRN outperforms every evaluated defense, in terms of overall security against both CAPs and UAPs. For better comparability with the other defenses, we have created a separate graphic for each defense, which can be found in the appendix. A direct comparison of the absolute values for the CIFAR-100 PRNs, trained with the different methods is shown in table 4.3. The numbers clearly show that our training method with CAPs and UAPs results in the best overall performance of all PRN models. This finding is particularly important in regard to our research question about which defense strategy offers the best protection against both UAPs and CAPs. In addition, the clean accuracy is only minimally influenced by the PRN, so that the performance of the target model is barely affected when processing regular inputs.

	SGA trained PRN	CAP trained PRN	SGA and CAP trained PRN
Clean	60.64%	59.51%	60.21%
PDG-10	13.77%	54.44%	51.66%
SGA	58.23%	45.86%	56.02%
Deepfool	55.19%	44.42%	52.7%

Table 4.3: Accuracy comparison between PRNs using different training methods. Evaluation performed on CIFAR-100

4.3 Imagenet

	Attack		None	Adv. Training	Super Resolution	PRN UAP trained	PRN CAP trained
Imagenet	Clean	CA	80.34	63.86	69.73	**78.67**	77.73
	PGD-10	AA	5.04	22.28	60.29	59.19	**75.48**
		FR	98.68	83.37	39.34	37.33	**16.06**
	SGA	AA	6.26	62.85	46.47	**77.30**	71.43
		FR	93.31	9.29	49.79	**12.6**	19.16
	Deepfool	AA	16.93	62.4	70.07	**76.84**	70.94
		FR	81.39	10.82	22.32	**13.54**	22.06

Table 4.4: Evaluation results for Imagenet

As expected, PRNs offer the best protection against UAPs. The UAP trained PRN achieves a 77.3% accuracy on SGA perturbations and a 76.84% accuracy on Deepfool. By putting this in relation to the clean accuracy of the base model (80.34%), it is obvious that this value is very high. For this and all other Dataset evaluations, we chose a PRN that was trained with 20 different SGA UAPs. Our experiments have shown that the performance of a PRN does not depend on the number of UAPs it has been trained with. Therefore, it is possible to train the entire PRN with just a single UAP. This high generalization capability is another feature that, to our knowledge, was not previously known. Interestingly, it does not matter either with which perturbations the PRN is trained. The performance is similar in all our experiments as the table 4.5 shows. This suggests that the PRN's reconstruction process is particularly robust and flexible to different attacks. PRNs are therefore much more adaptable and flexible than previously thought.

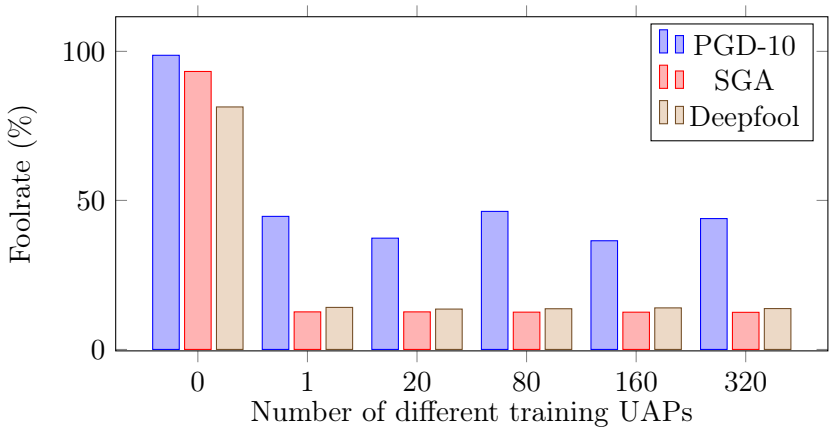

Figure 4.3: PRN perfomance depending on number of training UAPs. Evaluation performed on Imagenet with SGA UAPs

	SGA trained PRN	Deepfool trained PRN
Clean	78.67%	79.27 %
PDG-10	59.19%	54.12 %
SGA	77.30%	76.04 %
Deepfool	76.84%	76.72 %

Table 4.5: Accuracy comparison between PRNs trained on different UAPs classes. For each class 20 UAPs were used for the training

The protection factor of PRNs is so high that it overshadows all other defenses evaluated on Imagenet. Despite this, we want to take a brief look at the other ones. Adversarial Training offers good protection against UAPs, but the clean accuracy (63.86%) is low, which reduces the overall performance of the model. This form of defense is therefore only suitable for the CIFAR Datasets. Super Resolution Preprocessing offers a fairly high protection factor against both UAPs and CAPs. However, the basic performance of the model suffers and the clean accuracy (69.73%) is 10% lower than the clean accuracy of the base model. It is noticeable that the preprocessing is more effective against deepfool UAPs (22.32% foolrate) than against SGA UAPs (49.79% foolrate). We hypothesize that this is related to the structure of the individual UAPs. SGA UAPs only contain a few noise elements and are characterized by their distinctive pattern (see figure 3.1). It is difficult to remove this using preprocessing techniques, as the patterns are considered part of the image structure and not perceived as noise. Deepfool UAPs contain far fewer colors and patterns, which makes it much easier to remove them from the image.

4.4 Summary

The measured values across all datasets confirm, that UAPs are nearly as effective and powerful as CAPs. Especially SGA UAPs are significantly more effective than the Deepfool variant. The foolrate of SGA, is for example 93.31% where the foolrate of Deepfool is only 81.39% on the unprotected Imagenet model. CAPs in the form of PGD-10 perturbations have a slightly higher performance than UAPs, achieving a foolrate of 98.68% on Imagenet and 100% on both CIFAR-100 and CIFAR-10. However, this is not surprising and can be explained by the fact that CAPs, unlike UAPs, are optimized for one specific input. The comparison between the Imagenet and the CIFAR datasets clearly shows that the performance of a defense strategy is highly dependent on the input data and the architecture used. Despite the partly different results within the different datasets, a general trend can be observed. Adversarial training with CAPs offers a solid protection against UAPs, but the performance varies greatly from dataset to dataset. Feature denoising achieves similar results as adversarial training, but it is significantly more complex to train due to the increased model complexity. The absolute values in the evaluation tables 4.1 and 4.2 show that

feature denoising performs slightly worse than adversarial training. So the additional complexity during the training is not worth it. Super Resolution Preprocessing is particularly effective for larger input images and for UAPs that have a noise-like structure. The Perturbation Rectifying Network offers a very high level of protection across all data sets, regardless of the UAP class with which it was trained. Due to its high generalization capability, it is possible to train the PRN with only a single UAP. Training the PRN with CAPs is highly effective and provides an high level of protection against CAPs. The combination of UAP and CAP training, performs even better and results in the best defense technique to protect against both UAPs and CAPs. It can therefore be concluded from our evaluation that CAP defenses certainly offer a protective factor against UAPs. However, this is highly variable and depends heavily on the UAPs and the dataset used. The defense with the best performance on both CAPs and UAPs is the PRN. With our special training the performance of the PRN can be maximized.

5 Concluding Remarks

Following the detailed evaluation, it is now time to summarise our findings and place them in a broader context. As already mentioned in the introduction, the danger posed by adversarial perturbations should not be underestimated. UAPs in particular pose a major threat due to their universal transferability to both inputs and models. It is therefore essential to protect safety-critical AI applications against them. Despite their considerable danger, they have been left out of the development and evaluation of defensive measures in the past. This represents a potential security risk for the models in which the defences are implemented in. For this reason, we have devoted this thesis to the question of whether the defenses offer a protection factor against UAPs. Our work is intended to make a contribution to the security of machine learning models making them more resistant to attacks.

During our experiments, we evaluated four different Defenses on three Datasets to determine whether the defense methods developed CAPs are also robust against UAPs. We selected at least one defense from each defense category and tested it on various metrics to obtain meaningful results. Through our extensive evaluation, we are confident that defenses designed for CAPs are also largely effective against UAPs. We have found out that the protection factor against Deepfool is slightly higher than SGA and that the performance of the defenses varies from dataset to dataset. We can also conclude that the denoising preprocessing is not optimal to neutralize UAPs, as UAPs contain only few noise elements and have a pattern like structure. In addition, we have discovered the high generalization capability of PRNs and their outstanding performance against CAPs. Our research question went even further, to find the optimal defense strategy against both CAPs and UAPs. Through our newly discovered properties of the PRN, we have succeeded in implementing a novel training procedure using both UAPs and CAPs for training. The performance comes really close to an optimal defense. We have therefore fulfilled all the objectives of this thesis and will now take a look at further possible areas of research.

5.1 Future work

Although we have performed the evaluation very extensively in this work, some aspects remain open that we would like to investigate in the future. Firstly, we would like to evaluate more attacks on the defenses. There are numerous ways

to generate UAPs, each of them unique in their method of generation [8]. While writing this thesis, we came up with our own ideas to generate UAPs even more effectively and efficiently. We would like to implement them in future. Furthermore, an evaluation of real world UAPs would be interesting, as most works (including this one) only evaluate the effectiveness of UAPs under laboratory conditions. Ultimately, we would like to use our findings from the PRN to further develop the defense and achieve an even better protection factor. There are still some parameters that are very promising and that we have not yet studied in our experiments. Our long-term goal is to further advance the protection of AI models, and there are plenty of opportunities to do so.

5.2 Final thoughts

At the time of writing, Sora, the text-to-video model from OpenAI, has been launched. It is able to generate videos in a quality and level of detail that was unthinkable a few years ago. This is just one example of how fast the development of AI is progressing. However, with every advance, we also face new questions and challenges, particularly in terms of the safety, ethics and reliability of AI. These should play an essential role in the development process. While models such as Sora undoubtedly make impressive progresses, we must also ensure that appropriate safety measures and ethical guidelines are implemented. The responsibility lies not only in developing innovative technologies, but also in ensuring that they operate in a responsible and secure manner.

A Acronyms

CAP conventional adversarial perturbation

CNN convolutional neural network

DNN deep neural network

MLP multilayer perceptron

NN neural network

PRN perturbation rectifying network

UAP universal adversarial perturbation

B Appendix

Note: The shapes within the coordinate system stand for the different attacks and the colours for the respective datasets. The X-axis is the adversarial accuracy relative to the clean accuracy. A value of 100 therefore indicates that the adversarial accuracy is just as high as the clean accuracy. The Y-axis represents the foolrate as we know it. The further a data point is in the upper right corner, the better the defense performs, as the foolrate is minimized and the accuracy is maximized.

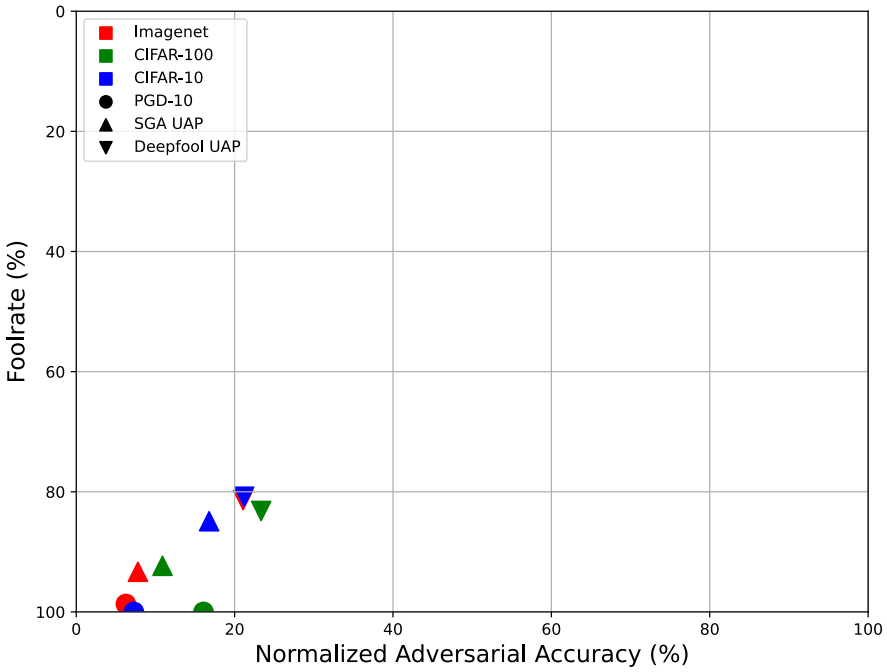

Figure B.1: Attack performance on the different datasets without any defense

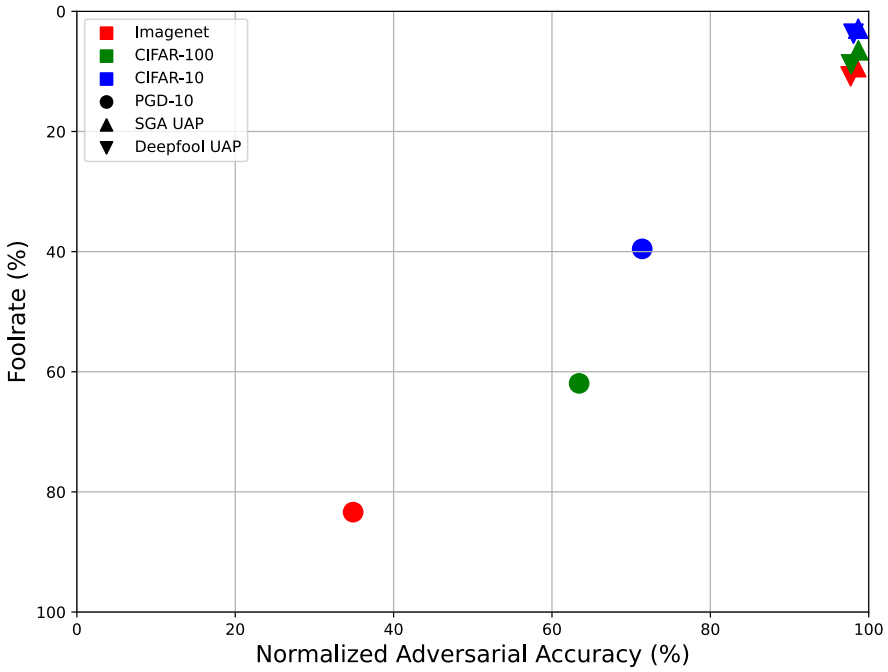

Figure B.2: Defense performance of the adversarial training

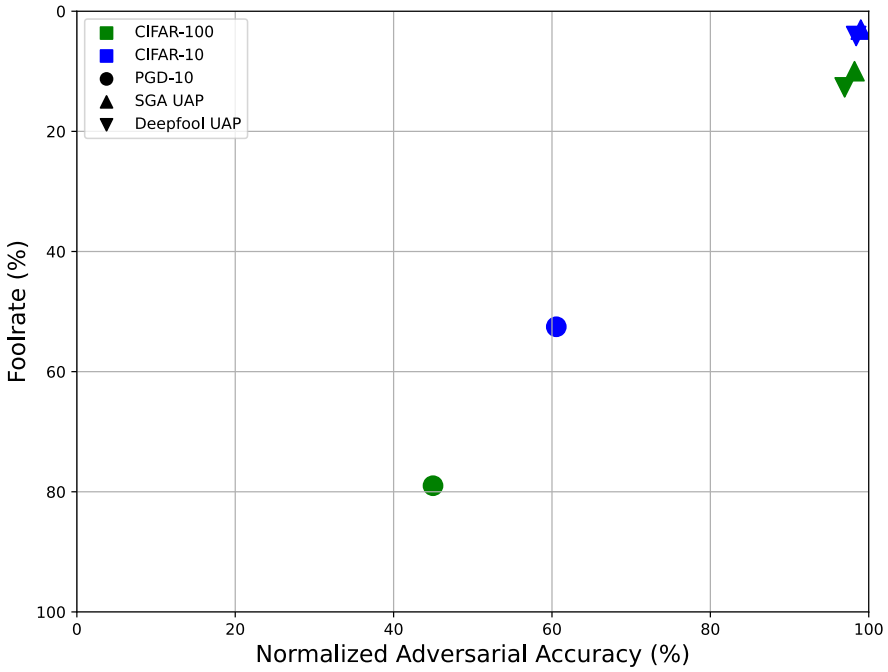

Figure B.3: Defense performance of the feature denoising

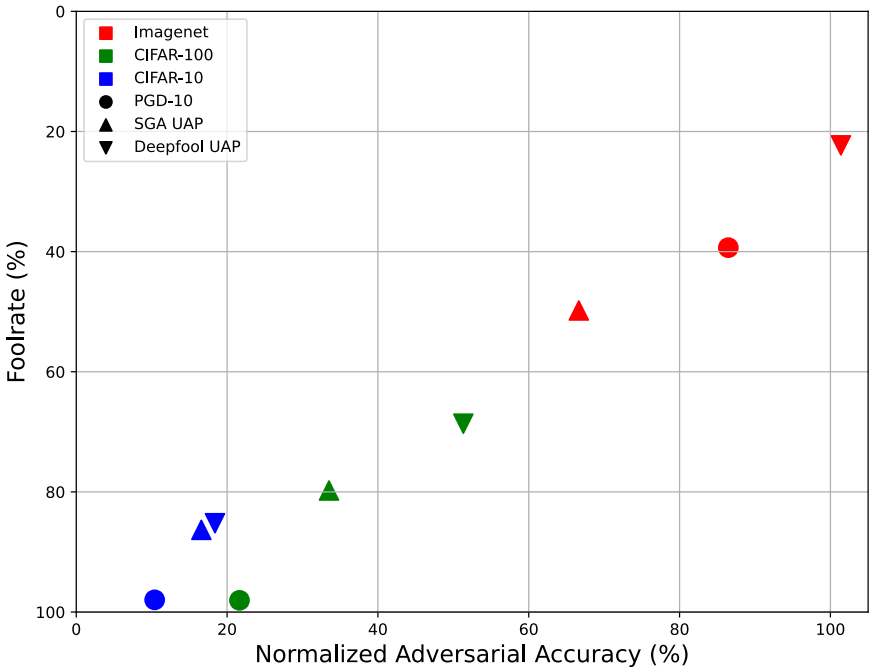

Figure B.4: Defense performance of the image super resolution preprocessing

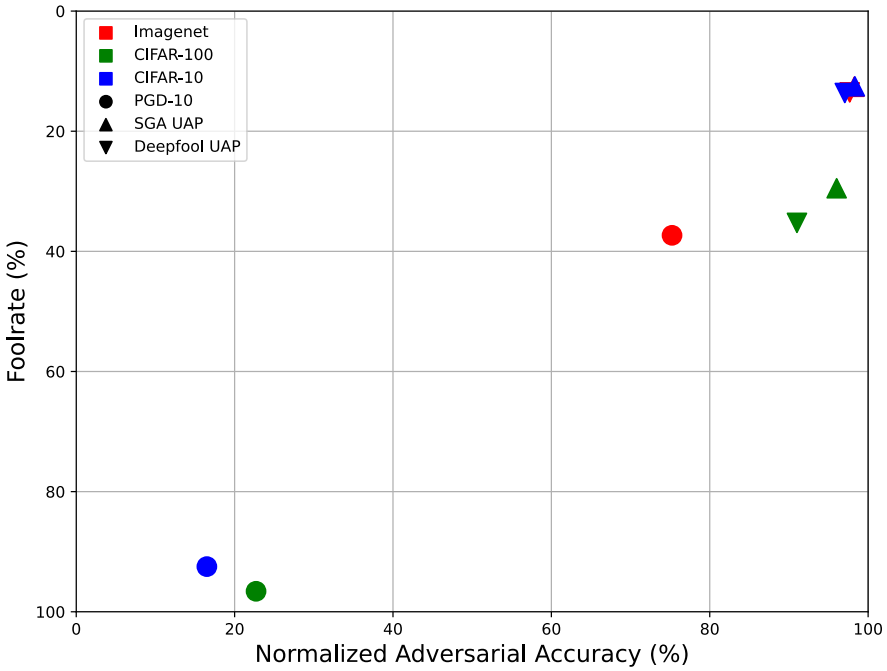

Figure B.5: Defense performance of the perturbation rectifiying network trained with UAPs

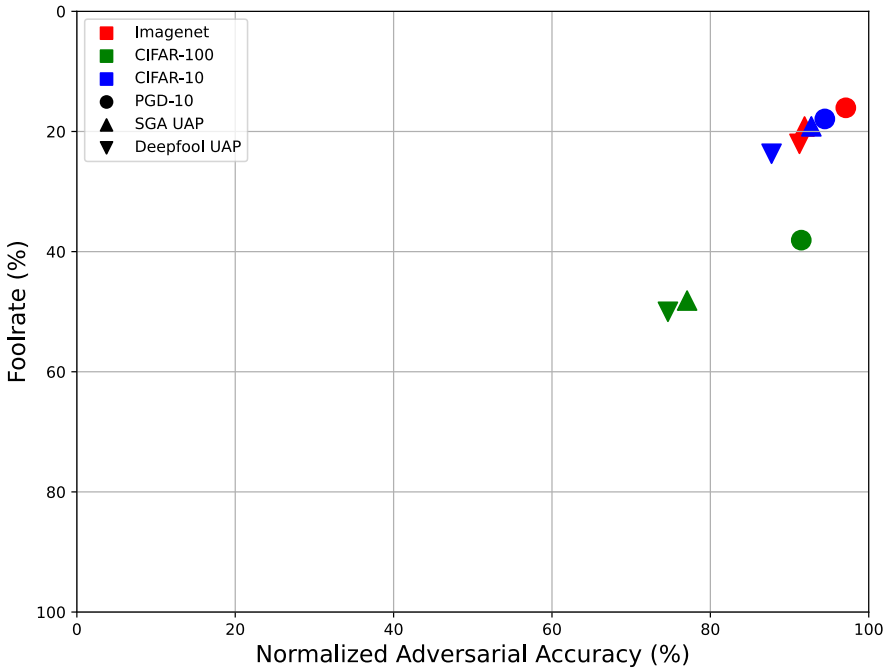

Figure B.6: Defense performance of the perturbation rectifiying network trained with CAPs

List of Figures

List of Tables

List of Listings

Bibliography

[1] Wikipedia.org, 2024.

[2] Wikipedia.org, 2024.

[3] Naveed Akhtar, Jian Liu, and Ajmal Mian. Defense against universal adversarial perturbations, 2018.

[4] Naveed Akhtar, Ajmal Mian, Navid Kardan, and Mubarak Shah. Advances in adversarial attacks and defenses in computer vision: A survey. *IEEE Access*, 9:155161–155196, 2021.

[5] G. Apruzzese, H. S. Anderson, S. Dambra, D. Freeman, F. Pierazzi, and K. Roundy. "real attackers donx27;t compute gradients": Bridging the gap between adversarial ml research and practice. In *2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, pages 339–364, Los Alamitos, CA, USA, feb 2023. IEEE Computer Society.

[6] Tao Bai, Jinqi Luo, Jun Zhao, Bihan Wen, and Qian Wang. Recent advances in adversarial training for adversarial robustness, 2021.

[7] Melika Behjati, Seyed-Mohsen Moosavi-Dezfooli, Mahdieh Soleymani Baghshah, and Pascal Frossard. Universal adversarial attacks on text classifiers. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7345–7349, 2019.

[8] Ashutosh Chaubey, Nikhil Agrawal, Kavya Barnwal, Keerat K. Guliani, and Pramod Mehta. Universal adversarial perturbations: A survey, 2020.

[9] Joana C. Costa, Tiago Roxo, Hugo Proença, and Pedro R. M. Inácio. How deep learning sees the world: A survey on adversarial attacks defenses, 2023.

[10] Francesco Croce, Maksym Andriushchenko, Vikash Sehwag, Edoardo Debenedetti, Nicolas Flammarion, Mung Chiang, Prateek Mittal, and Matthias Hein. Robustbench: a standardized adversarial robustness benchmark, 2021.

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[12] Mohammed Hassan. Medium.com, 2023.

[13] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. 05 2017.

[14] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima, 2017.

[15] Alex Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012.

[16] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution, 2017.

[17] Xuannan Liu, Yaoyao Zhong, Yuhang Zhang, Lixiong Qin, and Weihong Deng. Enhancing generalization of universal adversarial perturbation through gradient aggregation, 2023.

[18] Zihao Liu, Qi Liu, Tao Liu, Nuo Xu, Xue Lin, Yanzhi Wang, and Wujie Wen. Feature distillation: Dnn-oriented jpeg compression against adversarial examples, 2019.

[19] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks, 2019.

[20] Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks, 2018.

[21] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. Deepfool: A simple and accurate method to fool deep neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2574–2582. IEEE Computer Society, 2016.

[22] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 86–94, 2017.

[23] Aamir Mustafa, Salman H. Khan, Munawar Hayat, Jianbing Shen, and Ling Shao. Image super-resolution as a defense against adversarial attacks. *IEEE Transactions on Image Processing*, 29:1711–1724, 2020.

[24] Aaditya Prakash, Nick Moran, Solomon Garber, Antonella DiLillo, and James Storer. Deflecting adversarial attacks with pixel deflection, 2018.

[25] Yao Qin, Nicholas Carlini, Garrison Cottrell, Ian Goodfellow, and Colin Raffel. Imperceptible, robust, and targeted adversarial examples for automatic speech recognition. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97

of *Proceedings of Machine Learning Research*, pages 5231–5240. PMLR, 09–15 Jun 2019.

[26] Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S. Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free!, 2019.

[27] Saily Shah. Analytics vidhya, 2022.

[28] Gaurang Sriramanan, Sravanti Addepalli, Arya Baburaj, and R. Venkatesh Babu. Guided adversarial attack for evaluating and enhancing adversarial defenses, 2020.

[29] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, October 2019.

[30] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv:1312.6199*, 2014.

[31] Fatemeh Vakhshiteh, Ahmad Nickabadi, and Raghavendra Ramachandra. Adversarial attacks against face recognition: A comprehensive study. *IEEE Access*, 9:92735–92756, 2021.

[32] Viswa. Medium.com, 2023.

[33] Katy Warr. Strengthening deep neural networks, 2019.

[34] Juanjuan Weng, Zhiming Luo, Dazhen Lin, and Shaozi Li. Comparative evaluation of recent universal adversarial perturbations in image classification, 2023.

[35] Fabian Woitschek and Georg Schneider. Physical adversarial attacks on deep neural networks for traffic sign recognition: A feasibility study. In *2021 IEEE Intelligent Vehicles Symposium (IV)*, pages 481–487, 2021.

[36] Eric Wong, Leslie Rice, and J. Zico Kolter. Fast is better than free: Revisiting adversarial training, 2020.

[37] Jing Wu, Mingyi Zhou, Ce Zhu, Yipeng Liu, Mehrtash Harandi, and Li Li. Performance evaluation of adversarial attacks: Discrepancies and solutions, 2021.

[38] Cihang Xie, Yuxin Wu, Laurens van der Maaten, Alan Yuille, and Kaiming He. Feature denoising for improving adversarial robustness, 2019.

[39] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. In *Proceedings 2018 Network and Distributed System Security Symposium*, NDSS 2018. Internet Society, 2018.

[40] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks, 2017.

[41] Chaoning Zhang, Philipp Benz, Chenguo Lin, Adil Karjauv, Jing Wu, and In So Kweon. A survey on universal adversarial attack. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, IJCAI-2021. International Joint Conferences on Artificial Intelligence Organization, August 2021.

[42] X. Zhu and A. B. Goldberg. Introduction to semi-supervised learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 3(1):1–130, 2009.

[43] Yinghui Zhu and Yuzhen Jiang. Imperceptible adversarial attacks against traffic scene recognition. 25(20):13069–13077, oct 2021.