Lars Kåre Syversen

Neural Network Robustness Against Semantic Adversarial Attacks

Master's thesis in Informatics, Artificial Intelligence
Supervisor: Jingyue Li
Co-supervisor: Mathias Lundteigen Mohus
June 2021

Lars Kåre Syversen

Neural Network Robustness Against Semantic Adversarial Attacks

Master's thesis in Informatics, Artificial Intelligence
Supervisor: Jingyue Li
Co-supervisor: Mathias Lundteigen Mohus
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

For the past few years, deep learning has seen an explosive growth in popularity both in academic literature and industrial settings. The invention of the Convolutional Neural Network has taken image recognition tasks to the next level in terms of accuracy and versatility and become an integral part of many computer systems in the present day.

Recently however, it has been revealed that these networks, while accurate, are not as robust as once thought. In fact, an unnoticeably small, worst-case perturbation can be applied to most images, resulting in high-confidence misclassification. This is known as an adversarial attack. There are many different kinds of adversarial attacks, but one subgroup in particular stands out from the rest, namely semantic adversarial attacks. A defining characteristic of these types of attacks is large, unrestricted perturbations that makes defending against them more difficult. This is because a lot of defenses relies on an upper bound on the perturbation, and semantic attacks cause perturbations larger than this.

In this thesis, we show that adversarial training, a popular adversarial defence method, can be used to defend against multiple different unrestricted adversarial attacks simultaneously. We train three different robust models against individual attacks and one model against all three of them at once. The results show that the model trained against all three attacks perform equally well against each individual attack as their single-attack trained counterparts. Moreover, the adversarial training does not impact the standard accuracy of any of the robust models.

Additionally, we develop an adversarial toolbox specifically designed to generate and defend against semantic adversarial attacks. This toolbox is built using PyTorch and allows users with little prior knowledge of adversarial attacks to improve model robustness towards them.

Finally, we provide a literature review on the topic of semantic adversarial attacks, which has never been conducted previously. This review will assist people in attaining a deeper understanding of these types of attacks, hopefully leading to more research on the topic.

Sammendrag

I de siste årene har dyp læring hatt en eksplosiv vekst i popularitet, både i akademiske og industrielle applikasjoner. Introduksjonen av konvolusjonære nevrale nettverk har tatt bildegjenkjenningsoppgaver til et nytt nivå når det gjelder både nøyaktighet og allsidighet, og har blitt en integrert del av mangfoldige datasystemer i dag.

Nylig har det imidlertid blitt avslørt at disse nettverkene, selv om de er svært nøyaktige, ikke er så robuste som vi en gang trodde. Faktisk kan en nærmest usynlig endring i et bilde resultere i feilklassifisering med høy sikkerhet. Dette er kjent som et adversarialt angrep. Det finnes mange forskjellige typer adversariale angrep, men en spesifikk undergruppe av disse skiller seg ut fra resten, nemlig semantiske adversariale angrep. Et vanlig trekk ved disse typer angrep er store, ubegrensede endringer som gjør det vanskeligere for et nevralt nettverk å forsvare seg mot dem. Dette er fordi mange forsvarsmetoder er avhengige av en øvre grense på endringen av bildet, og semantiske angrep overgår disse grensene.

I denne oppgaven viser vi at adverarial trening, en populær forsvarsmetode mot adversariale angrep, kan brukes til å forsvare mot flere forskjellige semantiske adverariale angrep samtidig. Vi trener tre forskjellige robuste modeller mot individuelle angrep og en enkelt modell mot alle tre samtidig. Resultatene våre viser at modellen trent mot alle de tre angrepene oppnår like bra robusthet mot hvert enkelt angrep som modellene trent spesifikt mot hver av de. Dessuten viser resultatene at denne treningen heller ikke påvirker nøyaktigheten av vanlige bilder i noen av de robuste modellene.

I tillegg presenterer vi en toolbox som er spesielt utviklet for å generere og forsvare mot semantiske adversariale angrep. Denne toolboxen er designet for PyTorch og tillater brukere med få forkunnskaper om adversariale angrep å forbedre robustheten i nevrale nettverk overfor dem.

Til slutt presenterer vi en litteraturstudie om semantiske adversariale angrep, noe som aldri har blitt gjennomført tidligere. Denne studien vil hjelpe til med å oppnå en dypere forståelse av denne typen angrep, og forhåpentligvis føre til mer forskning om emnet.

Preface

This thesis is part of a 2 year long masters program at the Norwegian University of Science and Technology. I would like to thank my supervisor Jingyue Li for his continuous guidance and help with this project throughout the past 2 semesters. I also want to thank Mathias L. Mohus for his assistance and input in making this project come to fruition. Lastly, I would like to thank my friends and family who have supported me throughout my 5 years of studies at NTNU.

Table of Contents

List of Tables

List of Figures

Terms and Abbreviations

AI	=	Artificial Intelligence
ANN	=	Artificial Neural Network
ART	=	Adversarial Robustness Toolbox
BIM	=	Basic Iterative Method
CPU	=	Central Processing Unit
CNN	=	Convolutional Neural Network
FGSM	=	Fast Gradient Sign Method
GAN	=	Generative Adversarial Network
GPU	=	Graphical Processing Unit
HSV	=	Hue, Saturation, Value
ML	=	Machine Learning
MSE	=	Mean Squared Error
NTNU	=	The Norwegian University of Science and Technology
PGD	=	Projected Gradient Descent
ReLU	=	Rectified Linear Unit. Neural activation function.
RGB	=	Red, Green, Blue
RT	=	Rotations and translations

Chapter 1

Introduction

This chapter provides a short introduction to the topic of adversarial attacks and the defences against them. The general goal of the thesis is stated, and three different research questions are defined. These research questions are the focal point of our research and will guide the structure of the remaining parts of this thesis.

1.1 Research Motivation

In recent years artificial intelligence has become more accessible, more useful, and more advanced. The applications of AI in the current day are many and in the future they may be deeply involved in our everyday life, affecting transportation, healthcare, and manufacturing, as well as many other areas of importance. One of the greatest breakthroughs in neural network research came with the invention of the Convolutional Neural Network. Convolutional Neural Networks uses a special operation known as convolution to create feature mappings of the input and has been shown to be exceptional at processing images and videos, making them the go-to solution for tasks relating to either of these formats.

In 2015, it was discovered that CNNs could be consistently misled by applying a small but intentional worst-case perturbation to an image, resulting in high-confidence misclassification [7]. Images that successfully misled classifiers in this manner were called adversarial examples. Since then, extensive research has gone into the field, and several new attack methods to generate these adversarial examples have been identified [12, 15, 26]. As AI grows more and more widespread, the ability to consistently fool neural networks is a problematic one. Moreover, most of these attacks use mathematical bounds on the perturbations, ensuring that the differences between the original images and the adversarial examples are minimal. As a result, the adversarial examples are often indistinguishable from their original counterparts by the human eye.

Fortunately, the same bounds that ensure the quality of adversarial examples can be exploited to defend against them. Among the most popular defence methods, many are based on reducing image detail in some form, effectively washing away any highly precise perturbations from the image itself [23, 33, 10]. Since we can usually guarantee that the perturbation is within a specific measurement, these defences are very efficient and use few resources. Of course, with the introduction of such defence methods, other approaches to generating adversarial examples started being evaluated. One of these subgroups of adversarial attacks is known as semantic adversarial attacks.

Whereas conventional adversarial attacks focus on ways to perturb an image with as few visual artifacts as possible, semantic adversarial attacks does not adhere to these restrictions, but instead generate large, sweeping, and unbounded perturbations. As long as an adversarial example does not look doctored or fake, the extensiveness of the perturbation is of no importance. Semantic adversarial attacks, therefore, perturbs the semantic contents of the image while avoiding any changes that would make it obvious that the image classification has changed. Often this means perturbing image features, resulting in uniform changes to all parts of the image affected by the tweaked feature[63, 57, 34] .

The advantage of generating adversarial images this way is that the perturbations are large enough to avoid many defence methods that would otherwise be very effective. Since semantic attacks are unrestricted, the variations among them are wide, leading to adversarial examples that are unique and interesting in appearance [57, 34, 60, 24]. The same property makes defending against all or most of them difficult.

Although the unrestricted nature of semantic adversarial attacks makes them harder to defend against, there are still methods that have proved efficient, of which adversarial training is considered the most effective. Adversarial training, as the name suggests, involves training classifiers on datasets consisting of both normal and adversarial images. The added adversarial examples in the dataset trains the classifier to recognize adversarial images, providing an added layer of robustness. This method is also a common defence against restricted adversarial examples, although the drawback of having to generate enough adversarial images for training makes them less viable compared to other methods.

Semantic adversarial attacks are still a very new field, and while toolboxes made for generating adversarial examples do exist [27, 55, 37, 18, 29, 44], they do not have enough support for semantic attacks yet. The benefits of having a toolbox for these types of attacks could potentially be very significant, both in the industry, to improve robustness of existing ML models, and in research, as a tool to explore the interactions between different types of defensive strategies.

1.2 Research Goals

The main goal of this thesis is to improve model robustness against multiple different unrestricted perturbations by creating a toolbox specifically designed for generating and defending against semantic adversarial attacks. Both defences and attacks should be interchangeable and fluid, allowing for experimentation on semantic adversarial attacks that were not previously possible. Attacks should be simple to run, but simultaneously offer enough customization to be usable in both industry and research settings.

1.3 Research Questions

RQ1: What is the current landscape of semantic adversarial examples? Given the fast developments within the field, finding the current state-of-the-art attacks and defence methods is crucial in order to contribute to it. Semantic adversarial attacks tend to vary widely, so grouping similar attacks may help with organizing the current landscape and maybe even allow for new discoveries in defending against them. Because of this, a systematic literature review of the field will be conducted in order to map the current state-of-the-art attacks and the similarities between them.

RQ2: Can we make a specialized toolbox for generating and defending against semantic adversarial attacks? Several toolboxes specializing in adversarial attacks already exist and provides a large variety of attacks and defence methods for both research and industrial purposes. Unfortunately, these toolboxes generally focus on conventional, restricted attacks, and provides little to no support for the unrestricted, semantic ones. With the growing interest in semantic attacks, the need for an adversarial toolbox focusing on semantic attacks is high. The goal here is to implement state-of-the-art attacks identified in the literature review into a toolbox along with different defence methods, allowing for easy experimentation and improvements to classifier robustness. The toolbox should be easy to use and generate different semantic adversarial examples using as little time and resources as possible.

RQ3: Is it possible to effectively defend against multiple semantic adversarial attacks using adversarial training, and what are the drawbacks to using this approach? Since the birth of adversarial image generation, more and more attacks have been researched and developed. Defending against one type of attack is not helpful if the model remains susceptible to all the other ones [47, 49]. Finding a decent all-round defence against adversarial attacks is a difficult, but important task. It is also important that such a defence does not severely impact its performance on its intended task [26, 51, 50].

1.4 Contributions

The different contributions provided by this thesis are as follows:

RQ1: A systemic literature review of a rapidly evolving field of study, namely semantic adversarial attacks. No such literature review has previously been conducted.

RQ2: A new Python-based toolbox for generating and defending against several different semantic adversarial attacks.

RQ3: Showing that adversarial training for multiple semantic attacks is more effective than for conventional adversarial attacks and with fewer drawbacks. Previous studies on conventional adversarial attacks has shown that adversarial training for different perturbations negatively impact each others performance [50], as well as standard accuracy [49]. Later in this thesis, the adversarial training for multiple attacks will be referred to as *integrated adversarial training*.

1.5 Outline

The rest of this thesis is organized as follows:

Background provides quick summaries of the different scientific fields relevant to the thesis.

Related Work showcases a select few papers that relate to the contents of the thesis in some way.

Method describes the setup and implementation details of the different research contributions, such as the semantic adversarial toolbox (RQ2), or the hyperparameters used in the integrated adversarial training (RQ3).

Results provides the actual results of the research, such as the test accuracy of classifiers trained on multiple adversarial attacks compared to other adversarially trained networks.

Discussion discusses the obtained results in relation to previous works, as well as academic and industrial impact.

Conclusion wraps up the thesis, providing final thoughts on the different research questions and identifying future work.

Chapter 2

Background

This chapter provides introductions to the different scientific topics related to this thesis. A general introduction to neural networks and their structure is given. Adversarial attacks and defences against them are covered as well in order to give the reader basic knowledge on what they are and how they work. We assume that the reader has prior knowledge regarding computer science concepts and at least some understanding of machine learning.

2.1 Neural Networks

2.1.1 Neuron

The most basic piece of a neural network is the Neuron. On the most basic level, a neuron takes an input and performs a simple mathematical operation on it before outputting the result. Because of its simplicity, both the input and output is typically a numerical value. One neuron will have one or several inputs, and each input also includes one weight, which is also a numerical value. From Figure 2.1, you can see how a neuron performs its function. The neuron takes each input and multiplies it by its corresponding weight. All the results from these operations are summed, which then becomes the output.

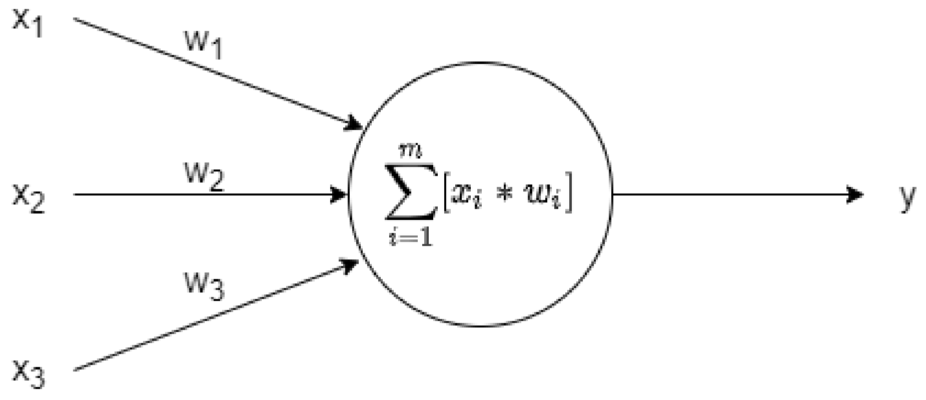

Figure 2.1: Illustration of the inner workings of a single neuron.

Depending on the inputs and their corresponding weights, a single neuron may mimic the behavior of a simple function, such as acting as a logical gate. The main strength of the neuron however lies in their numbers. Whereas a single neuron is not capable of much, a large amount of them working together are capable of accomplishing highly complex tasks. When several neurons are connected together in a large network, we call it a neural network.

2.1.2 Layers and Architectures

When several neurons are connected in a network, they are connected in such a way that the output of one neuron will feed into the input of another neuron. As mentioned in section 2.1.1, a single neuron may have several inputs, meaning that the output of a neuron may feed into several other ones. This creates a layered structure where each neuron in a layer generates values that are fed into the neurons of the next layer. A neural network may consist of many layers, all of which will get their input from the previous layer and output their results to the next. The exceptions to this rule are the input layer and the output layer. In the input layer, neurons do not get their input from the previous layer, but rather the inputs to the network itself. This serves as the gateway for data to pass through the network. The output layer is the last layer of the whole network, so their output does not feed into any next layer.

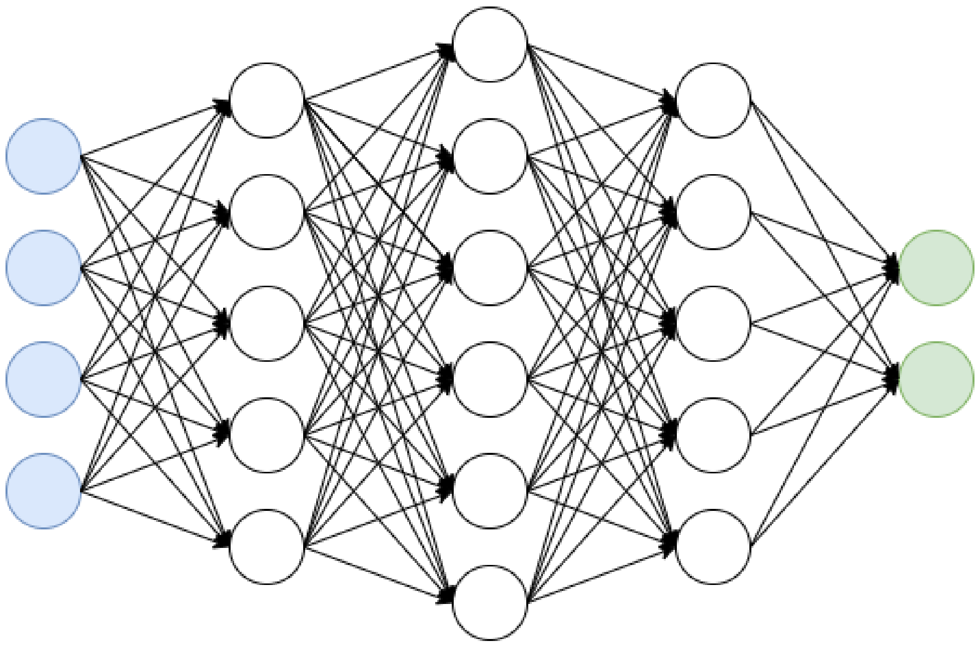

Figure 2.2: Illustration of a fully connected neural network with three hidden layers. The blue neurons are in the input layer and the green neurons are in the output layer.

A common way of organizing neural networks is by structuring them in three parts, this being the input layer, the output layer, and the hidden layers. The last one refers to every layer in between the input and output layers and is generally described this way since there is no need for any more detailed description from an outsider perspective. When every neuron in a layer is connected to every neuron in the next layer, it's called a fully connected neural network. An example of such a network is illustrated in Figure 2.2. Of course, not every neural network needs to be fully connected. Many different variations of neural networks exists, though the principle of connected layers of neurons remains the same. The number of layers in a network and the number of neurons in each of them may vary depending on the task to be done. We refer to these parameters as the network architecture.

2.1.3 Activation Functions

When thoughts are formed in your head, the brain transmits information from neuron to neuron through synapses. It is well known that artificial neural networks are based on the neural structure of the brain. A biological neuron does not always transmit information forwards when they get an input. Rather, they have an electrical threshold that dictates whether the information should be transmitted or not. When the threshold is reached, we say that the neuron 'fires' or activates.

This same neuron activation strategy is present in artificial neurons as well. In order to mimic the behavior of a neuron firing, artificial neurons use a mathematical function called an activation function. This function takes the result of the neuron's mathematical operation as input and uses this to decide the final output of the neuron.

As mentioned, biological neurons have a binary activation function, they either fire when their threshold is reached, or they don't. This used to also be the case for the perceptron, which was the predecessor to neural networks. It was later found out that these binary activation functions did not perform as well as expected, so a new approach of using continuous values was adopted instead. This means that artificial neurons today do not simply operate on a fire or no fire basis, but rather always activates to some variable extent depending on the activation function.

Today, there are many variations of activation functions in use. One of them is the Sigmoid function, described in equation 2.1. Sigmoid constrains the range of activation values to between 0 and 1 and is often used in the middle layers of a neural network.

$$S(x) = \frac{1}{1+e^{-x}} \tag{2.1}$$

Another very common activation function is the Rectified Linear Unit or ReLU for short. ReLU outputs more linear values but always excludes negative ones, as shown in equation 2.2. Its linear nature makes it easy to compute and it is commonly used in hidden layers.

$$R(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.2}$$

2.1.4 Weights and Biases

In section 2.1.1, it was mentioned how a neuron computes its output based on inputs and weights. Essentially, inputs multiplied by weights are summed together to produce a result. In this scenario, the only controllable elements are the weights, which we can modify to produce a desired output. Consider the Sigmoid activation function described in equation 2.1. Here, x is the input multiplied by a weight w. We can detach w from x in this equation by modifying it slightly, as shown in equation 2.3. Now, x only represents the actual input to the neuron.

$$S(x) = \frac{1}{1+e^{-(w\cdot x)}} \tag{2.3}$$

From this equation, we can see that the value of w only changes the steepness of the function, while the intercept remains at 0.5. This may lead to problems fitting data to the function, leading to poor performance. In order to fix this, we add another term to the output calculation, the bias. The bias is simply a number added to the output equation and is, like weights, controllable by the network. The full equation for calculating neuron output is described in equation 2.4.

$$S(\sum_{i=1}^{m}[x_i * w_i] + b) \tag{2.4}$$

Here m is the number of inputs, b is the bias and S is the activation function.

2.1.5 Forward Pass

Though several types of neural networks exist, the most famous and most common type is the feed-forward network. In a feed-forward neural network, layers of neurons can be arranged in a manner of start to finish where each layer output feeds forwards with no cycles present in the network. Figure 2.2 shows one such network.

These types of networks operate on two different algorithms, the forward pass, and backpropagation. In the forward pass, the network takes an input and passes it along to the hidden layers. Each neuron in these layers produce their own activation output and pass these along to the next layer. This continues until the data has been passed all the way to the output layer. The activations of the output neurons become the final output of the whole network. For classification tasks, the number of output neurons usually equals the number of different possible classes, where each neuron's activation denotes the likelihood of their corresponding class.

2.1.6 Back-propagation

In order for a neural network to produce meaningful results, the weights and biases of the network have to be accurately defined. For small tasks, each of these parameters could be set manually by calculating the outputs and finding appropriate values. While the smallest neural networks could contain no more than 10 separate parameters, large ones could contain upwards of several millions. In order to properly fine-tune this many variables, an automatic method of finding parameter values is a crucial necessity. Backpropagation is such a method.

Backpropagation is the step following the forward pass, when the final output of some sample has already been calculated. In order for backpropagation to be viable, we need to know the true classification of the sample, which we call the label. Based on the network output and the label, we can calculate the error of the network and use this information to adjust the parameter values accordingly. The error is calculated using a loss function, which can vary depending on the task. Common loss functions include Mean Squared Error (Equation 2.5) and Cross-Entropy (Equation 2.6).

$$MSE(y,\hat{y}) = \frac{1}{n}\sum_{i=1}^{n}[y_i - \hat{y}_i]^2 \tag{2.5}$$

$$CE(y,\hat{y}) = -\hat{y}[y] + log(\sum_{j} exp(\hat{y}[j])) \tag{2.6}$$

In these equations, $\hat{y}$ is the predicted output and y is the actual output.

2.1.7 Gradient Descent

Gradient Descent is the most used backpropagation algorithm to date. Consider a differential function denoting the output loss of a neural network. We can graph this function by the network parameters, and the initial parameters will be a point somewhere on the graph. Somewhere on this graph there will be a global minimum loss where the parameters of the network are optimized just right to produce the best possible result. From the initial point on the graph, we can find the gradient and update the parameters in a way that effectively moves them closer to the global minimum. This is an iterative process, wherein each iteration we progressively move closer and closer to the minimum by modifying the parameters using gradients. Eventually, after a number of iterations, the parameters are optimized for the loss function. Figure 2.3 shows an illustration of gradient descent.

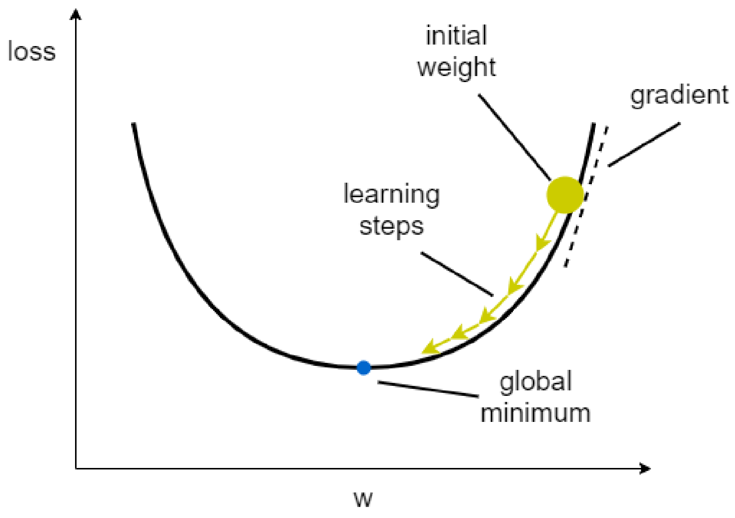

Figure 2.3: Illustration of Gradient Descent converging on a global minimum loss.

At the beginning of the gradient descent algorithm, it is safe to assume that the initial parameters of the network are far from the global minimum. Because of this, in the first few iterations we can take larger steps towards the optimum. As we get closer we take smaller steps as we don't want to risk stepping too far and accidentally increasing the loss. The step size is determined by the steepness of the gradient and a separate variable called the learning rate. Variables that are used to control the learning process are called hyperparameters. These are not found during the training phase but defined in advance.

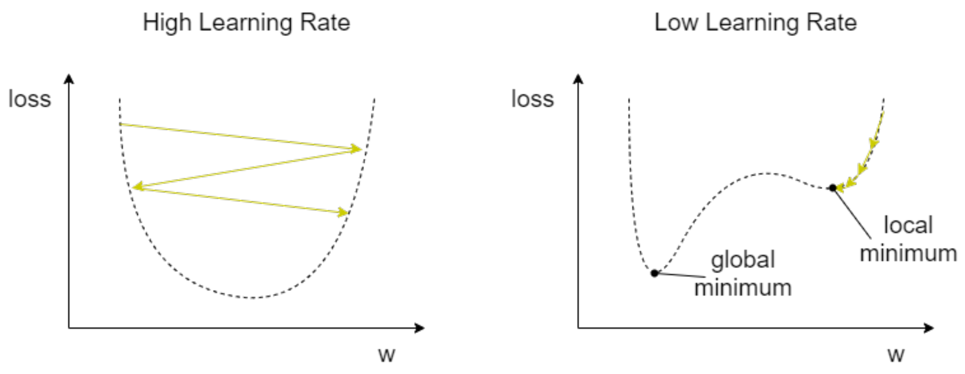

Figure 2.4: Effects of extreme learning rates.

The learning rate is a numeric hyperparameter that controls the step size through a simple multiplication. It is very important to use an appropriate learning rate as too large values can stop the network from converging. Low learning rates on the other hand can cause the gradient to get stuck in a local minimum rather than a global one, as demonstrated in figure 2.4. It is common practice to use what is known as a learning rate scheduler to dynamically adjust the learning rate during gradient descent. Using a scheduler can mitigate the risk of learning rate related issues.

2.1.8 Generalization and Overfitting

During the training phase, a neural network will converge on whatever parameters grants the best accuracy for the training data. Its actual performance however is measured on unseen data, as this is the data it will process in a real setting. An ANNs performance on real, unseen data is referred to as its generalization. This can be seen as the network's ability to identify and learn the underlying concepts of different classes. In other words, the network is trained to correctly classify training data, but we actually want it to correctly classify unseen data.

As a side effect of this approach, there is a risk of the model not learning enough or even learning too much. If a model is allowed to train on the same data for too long, it may learn the images themselves and not the concepts of what they contain. This is known as overfitting, and it leads to bad model generalization and poor testing performance. If the model doesn't train long enough however, it will not learn enough from the training data to make good predictions. We call this phenomenon underfitting. Ideally, the training phase should last long enough to avoid underfitting, but not long enough to cause overfitting. Avoiding overfitting is not always easy, and what amount of training is appropriate must be judged on a case-by-case basis.

2.1.9 Convolutional Neural Networks

A Convolutional Neural Network (or CNN), is a neural network variant that specializes in image-related tasks. On a very high level, it is known that CNNs extract image features and makes predictions based on these. This is done using special mathematical operations known as convolutions. A CNN is typically built using a mixture of convolution layers, pooling layers, and fully connected layers.

Convolution Layer

A convolution takes an input matrix and applies a sliding window operation to it using a filter to generate an output which we call a feature map. The input can be an image or the feature map of another convolution, while the filter is usually a smaller matrix. Using Figure 2.5 as an example, the operation starts from the top left, where each cell in the 2x2 filter matrix is multiplied by their counterparts in the top-left 2x2 square in the image matrix. The example is very simplified using only ones and zeroes, so the only cell that produces a non-zero result is the very top left one, where 1 is multiplied by 1. The results from each cell are summed together, which in this case would be 1. This result goes in the top-left cell in the resulting feature map. The 'window', which is the position in the image where the filter is being applied, then shifts one column to the right, where the same operation as before is applied. Comparing the filter to the top middle 2x2 square of the input, we can see that two cells are now producing non-zero results, so the top-middle cell in the feature map is 2. We keep applying this filter to every possible 2x2 square in the input, generating the final feature map present in the figure.

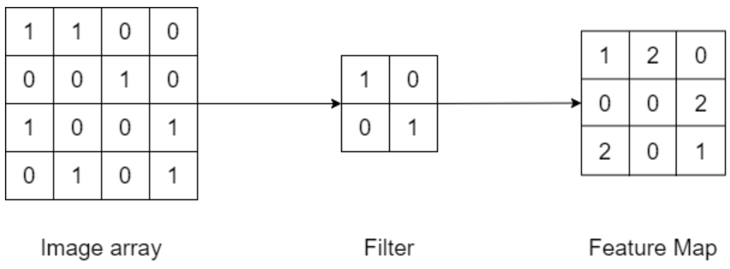

Figure 2.5: 2x2 convolution finding diagonal lines in an image.

Looking at the example convolution in Figure 2.5, the filter contains a diagonal line from top left to bottom right. This generates a feature map where every cell containing 2 represents parts of the input that contains diagonals. You could say that diagonal lines are the features we're filtering for and the output is a mapping of those features in the input.

One important property of convolutions is that the feature map has smaller dimensions than the input. Using a filter size of 2x2, the output dimensions will only be reduced by 1, but larger filter sizes will generate smaller output dimensions. If this is not acceptable, we can use input padding to control the output dimension size. In the example, adding one column and one row of zeroes will produce a feature map of size 4x4, which is the same as the input. Another hyperparameter we can add to a convolution is stride. In figure 2.5 we use a stride of 1. This means that the 'window' only moves one row or column between each slide. Using a stride of 2, the filter would jump directly from the top left to the top right 2x2 squares in the input matrix. Larger values of stride is another way of reducing the feature map dimension size.

Pooling Layer

A pooling layer condenses the information of feature maps by reducing their dimension sizes while keeping the most important feature information. There are two main methods of performing a pooling operation, both of which utilize a similar sliding window mechanic to convolution layers. In Figure 2.6, a max-pooling operation reduces a 4x4 matrix to a 2x2 matrix. Each window is highlighted by color, along with its output. The output of each window is simply the largest cell value within them. Another approach uses the average value of each window rather than the max. This is aptly named ’Average Pooling’.

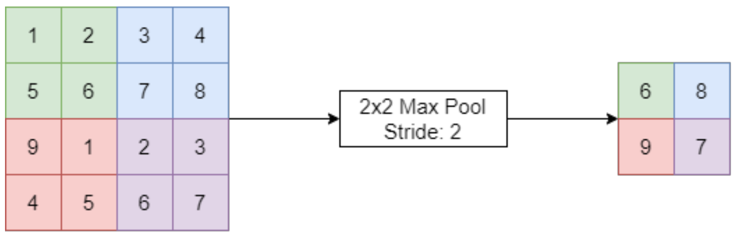

Figure 2.6: Example of a max pooling operation.

Architecture

A CNN uses a mixture of convolution layers and pooling layers to generate feature maps. The structure of these layers is similar to the structure of fully connected networks, where the output for each layer is passed on to the next one. For feature maps, this structure gradually produces more and more high-level features deeper into the network. As described in section 2.1.9, a filter can be used to find diagonal lines in an image. By changing the filter, one can also find horizontal or vertical lines. The resulting feature maps can then be used in the next layer to find bends or corners. These feature maps can be further used to find even higher-level features. As the feature maps start representing more complex features, their dimension size becomes smaller and smaller. The filter of each convolution is not set manually, as this is not a feasible strategy. Instead, the filters are determined using gradient descent, much like the ordinary parameters of a neural network.

Neither convolution layers nor pooling layers are capable of actually predicting the class of an image, they simply encode them into a list of high-level features. In order to produce a proper prediction, they use a sequence of fully connected layers that take the feature maps and outputs a classification. The full architecture of an example CNN is showcased in Figure 2.7.

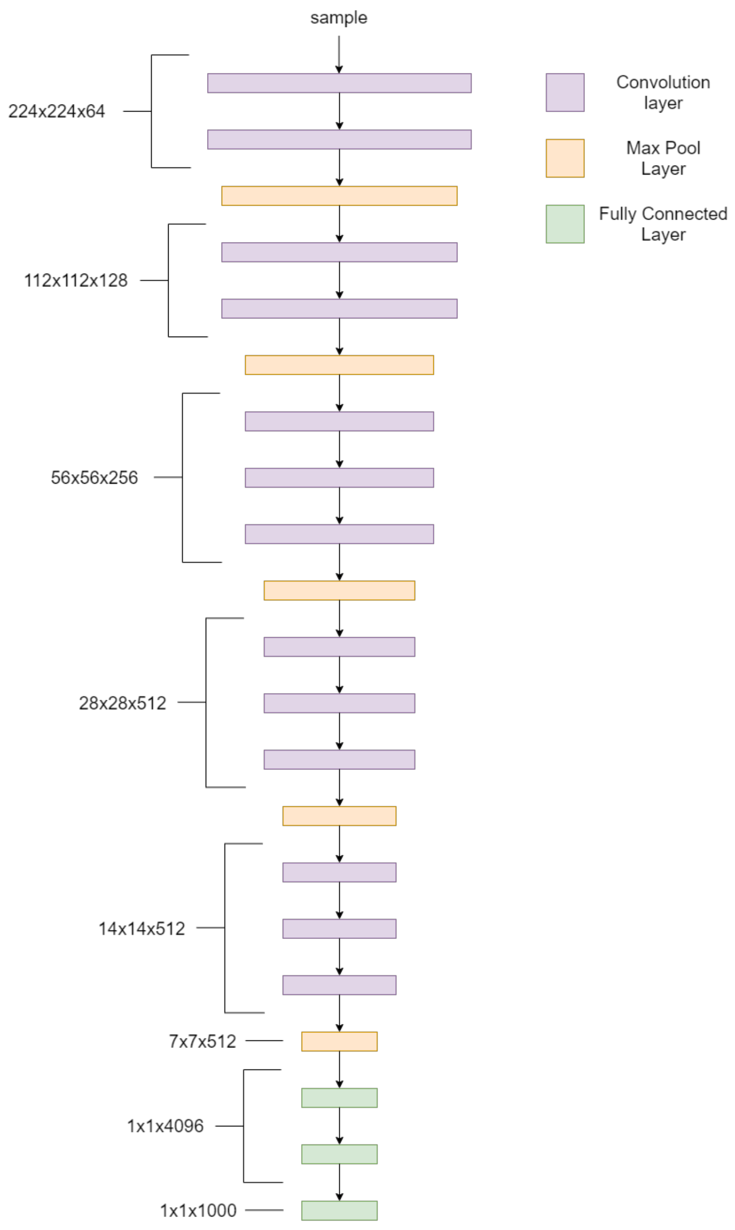

Figure 2.7: A typical CNN architecture. This specific architecture is called VGG16.

2.1.10 Residual Neural Networks

The number of layers in a neural network is referred to as its depth. In recent years, neural networks have become deeper and deeper in order to accommodate more complex tasks. It seems intuitive that deeper networks result in better performance, given that it has more weights and biases to fine-tune. The reality, however, is very different. It seems that, past a certain threshold, deeper networks perform worse than their shallower counterparts [11]. What's more, this increase in classification error is not caused by overfitting, as the error does get progressively better even in the deepest of networks. Rather, it seems that the error is consistently higher when using deep neural networks. This is known as the degradation problem [11].

Given a deep neural network, it should be possible to make it at least as good as its shallow counterpart by utilizing identity functions. Put simply, if a deep neural network produces results worse than their shallow counterpart, there must be some neurons in that network that negatively impacts the final results. If these unnecessary neurons are able to output their exact input, their impact on the final result vanishes completely. The problem with this approach is that deep neural networks are incapable of learning the identity function on their own. Residual neural networks [11] are the solution to this problem.

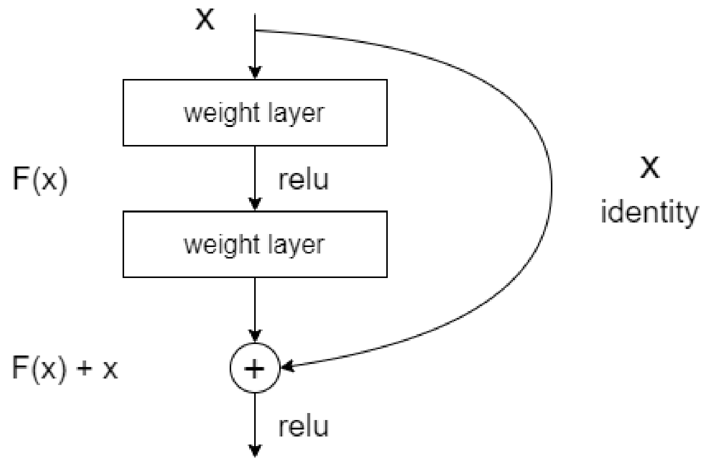

Figure 2.8: The building block of a residual neural network. The shortcut allows the block to output the identity if necessary.

In a plain neural network, the output of one neuron feeds directly into the next layer. Since a neuron cannot learn the identity function, all nodes in this type of network will produce an output that is at least somewhat different from the input. Given a true output function h(x) and an input x, we can calculate this difference as:

$r(x) = h(x) - x.$

This is the residual of the neuron. The neuron itself is of course trying to learn the true output function. In case the neuron is unnecessary, the ideal output becomes the identity function, which is a problem. Rearranging this formula, we get:

$h(x) = r(x) + x.$

From this new formula, we can get the true output by learning the residual function and summing this with the input. The advantage here is that the residual function is easier to learn than the true output function. This also means that if the ideal output is the identity, the neuron simply needs to output 0, which when summed with x will give the correct result. This is the core idea of a residual neural network.

To achieve this, a shortcut is implemented called the identity shortcut, as seen in Figure 2.8. Unlike plain neural networks, where the output only feeds into the next layer, we also send the output a few layers ahead. Using this approach, input x goes through both the normal route and the shortcut. The output from the normal route is then summed with the shortcut to get the true output.

2.2 Image Representation

In the world of computers, an image file is just a grid of values where each grid cell represents a pixel. Imagine a gray-scale image 20 pixels wide and 30 pixels tall. The computer representation of this image would be a matrix with 30 rows and 20 columns. Each cell would contain the color value of the corresponding pixel in the image, with the value 0 representing black, 255 representing white, and everything in between representing a shade of gray. This way, the entire image could be encoded in the grid.

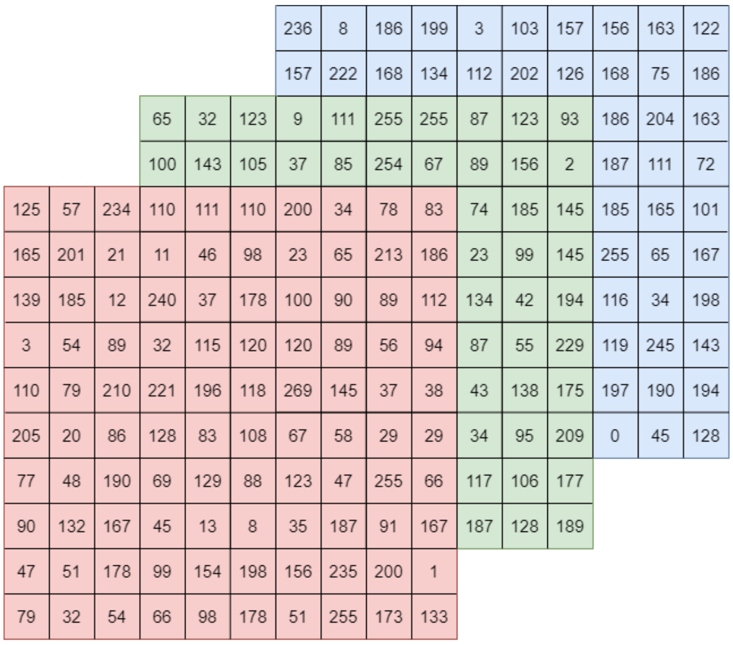

Figure 2.9: Image file of size 10x10 encoded as a 3D matrix.

But most images today are not gray-scale, they are fully colored. Computers define colors using the RGB color model. In this model, all colors are defined as a mix of red, green, and blue components. Representing all three components in a single grid cell would make the resulting value too complex. The solution to this problem is making three different grids, each representing a different color. One grid contains only the red value of each pixel, another the green values, and a third one containing the blue values. This structure is illustrated in Figure 2.9. The three different grids are called *color channels*. A fully colored image therefore is represented by a matrix of size $[width]x[height]x[color\_channels]$.

2.2.1 Color Representation

For many years RGB has been the go-to solution for encoding colors in a computer-readable way. The RGB model encompasses more than 16 million different colors, all of which are made up of some combination of red, green, and blue. Figure 2.10 shows a visualization of the color space in the shape of a cube. The model itself is additive, meaning the final color is an additive sum of its components. Although RGB has seen widespread adoption in the digital world, there are many alternate color models available for use. Popular alternatives to representing colors include HSV, HSL, CIELAB, and CIELUV.

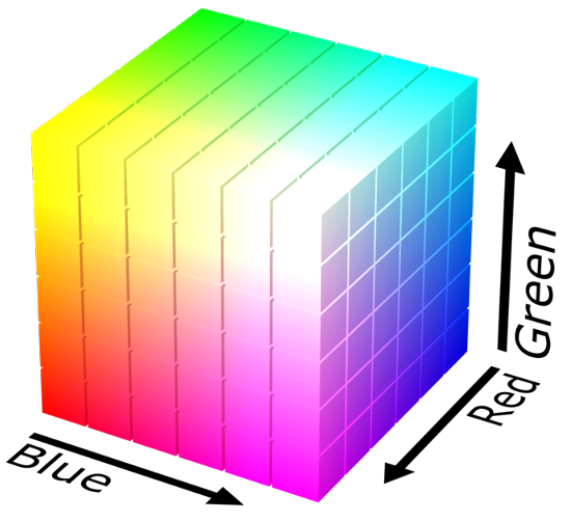

Figure 2.10: RGB color space represented as a cube.

HSV

In the HSV model [1], a color is defined as a combination of hue, colorfulness and brightness. The name is an abbreviation for hue, saturation and value. Much like the RGB model, three components are combined to create a final color. Figure 2.11 visualizes the color space of HSV. The hue, saturation, and brightness components combining to make a specific color may be more intuitive to humans than a sum of red, green, and blue values, but as a result, the model is incapable of producing a true black and white color. A black color in the HSV model is simply any hue value devoid of brightness, whereas white is any hue with very large brightness.

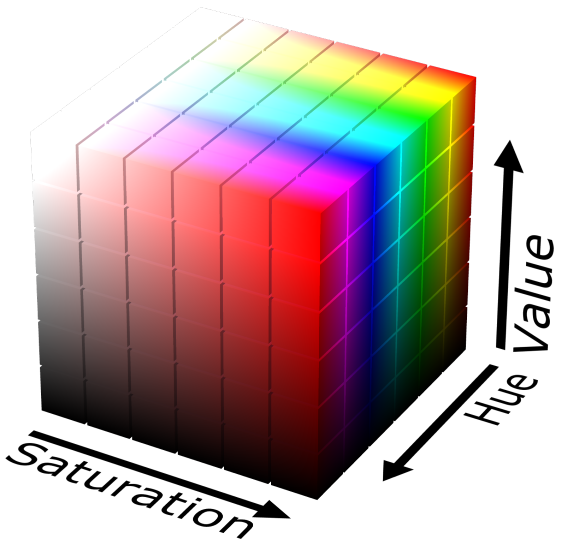

Figure 2.11: HSV color space represented as a cube.

LAB

CIELAB (or LAB) [4] is a color model built on a brightness channel (L) and two different color channels (A and B). A visualization of the color space can be seen in Figure 2.12. The A channel is a color spectrum of green to red and the B channel is a spectrum of blue and yellow. LAB is designed to be perceptually linear in color and to approximate human vision. This means perceptual differences in color are uniform across the range of different component values. Despite this, LAB is known to have issues with different shades of blue.

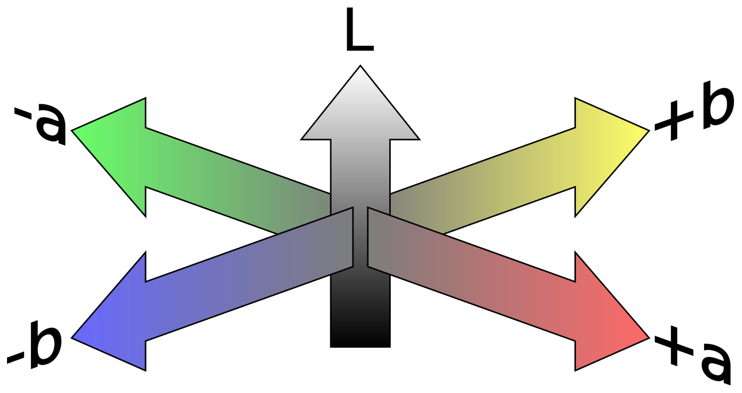

Figure 2.12: Lab color space represented in a 3D coordinate system.

2.3 Adversarial Attacks

In machine learning, we define an *adversarial example* as some input that has been deliberately perturbed in a way that causes a model to incorrectly label said input. Using image classification as an example, an adversarial example could be an image of a dog that has been purposefully designed to get misclassified as a cat by a specific network model, such as the example shown in Figure 2.13. Such examples showcase the vulnerability of neural networks and open up the possibility for targeted attacks on such models.

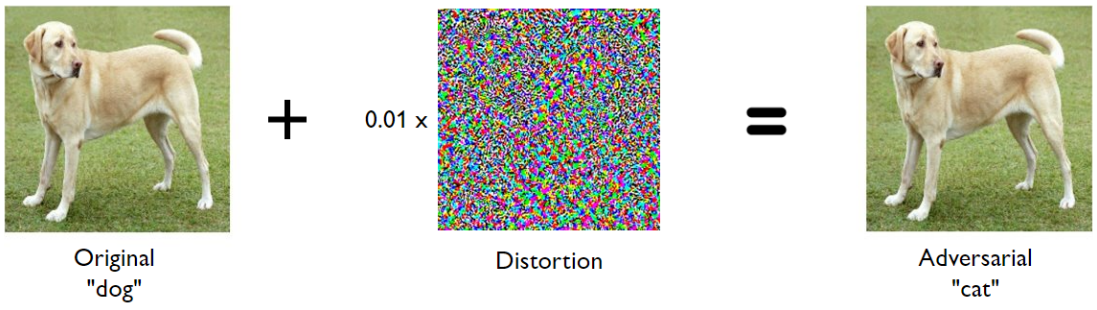

Figure 2.13: An adversarial example tricking a neural network to misclassify a dog. A small distortion is added to the original image to make the perturbation unnoticeable. The specific attack used in this example is called FGSM [7] (see section 2.3.5).

A targeted neural network attack that aims to trick the system using adversarial images is called an *adversarial attack*. Such attacks are not only limited to the digital world but can in fact be transferred to physical world scenarios, as shown by Kurakin et al. [12], when they printed adversarial images on paper and fed those images to a neural network classifier through a standard smartphone camera. Eykholt et al. [22] also manufactured a real-world stop sign that was misinterpreted as a speed limit or added lane sign in 100% of testing conditions. Adversarial attacks exist on several AI systems, but for this thesis we will mainly focus on image classification problems using neural networks.

2.3.1 Whitebox and Blackbox attacks

Like most computer attacks, the more knowledge you have about a system, the easier it is to exploit. This is no different for attacks on neural networks. Having access to a network's parameters, gradient, or training set allows for a variety of different attacks to be used. Attacks that exploit and require knowledge about a target model's inner workings are called *whitebox attacks*. Exploiting gradient information yields impressive success rates, but may be less applicable in a real-life scenario. After all, it is unusual for neural networks to provide such information to outsider clients.

On the opposite end, we call attacks applied with no target model knowledge *blackbox attacks*. Such attacks may involve some type of heuristic search method to find adversarial images or even take advantage of their transferability, as described in section 2.3.2. Perhaps the biggest advantage of black-box attacks is how they can be applied to virtually any neural network, as access to the output classification is the only information needed.

2.3.2 Transferability

Part of why adversarial attacks are so effective against deep neural networks is the property of transferability that these attacks seem to have. As long as the classification task remains the same, two entirely different models may be weak against the same adversarial image, regardless of architecture and training set [14]. This not only means that an attack on one network may affect another different network, but also allows attackers to apply whitebox attacks to blackbox models by creating their own target model replica. Any adversarial image generated for the replica model also has a chance of working against the original target.

The reason as to why adversarial attacks have this property is not fully understood yet, but there's suspicion that there is some intrinsic vulnerability to AI models causing this to happen [36]. Regardless, the prevalence of such a property provides an additional challenge for neural networks to overcome.

2.3.3 ℓ_p-norm metrics

An adversarial image is not a threat to any network if they are easily identifiable. The clearly identifiable contents of an adversarial image to the human eye coupled with its intentional misclassification is what separates them from other, non-nefariously misclassified images. It is therefore important for any such image to look as innocent as possible. In order to ensure the visual quality of adversarial examples, some sort of metric is needed to measure it. For this purpose, we commonly use the metrics ℓ_0, ℓ_2 and ℓ_∞.

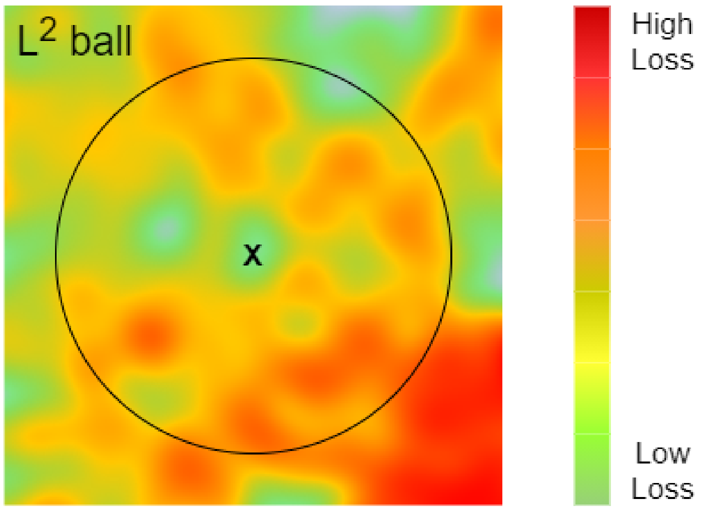

Figure 2.14: Illustration of a ℓ_2 ball encircling a sample x.

Most adversarial attacks restrict their perturbations to being within some bound of one or more of these ℓ_p metrics. By defining these bounds, we can ensure that an attack will create adversarial images of at least acceptable quality. Not all adversarial images use the same bounds or metrics. Attacks can have some unrestricted ℓ_p metrics while others are restricted. Bounding attacks by different types of metrics cause different types of perturbations.

A common term associated with ℓ_p metrics is ℓ_p balls. ℓ_p balls are a visualization of all perturbations within a ℓ_p boundary. An example of this can be seen in Figure 2.14, where the circle (or ball) shows the specific ℓ_2 boundary for sample x.

ℓ_0 metric

ℓ_0 gauges the number of pixels that have been perturbed in an adversarial image. How significant the perturbations for these pixels are is not part of the measurement. The only focus is on the count of pixels that have had their values change.

$$\ell_0 = |\{i|\delta_i \neq 0\}| \tag{2.7}$$

Equation 2.7 shows the mathematical definition of the metric. In the equation δ_i is the perturbation for pixel i. Restrictions on this metric can be used to control how much of an adversarial example should remain unedited. Taken to the extreme, it is possible to successfully generate an adversarial image by only perturbing a single pixel [48].

ℓ_2 metric

The ℓ_2 metric gauges the euclidean distance of all pixel changes. This factors in both the number of pixels changed and the significance of them. The formal mathematical definition for ℓ_2 is defined in Equation 2.8.

$$\ell_2 = \sqrt{\sum_i \delta_i^2} \tag{2.8}$$

Given that ℓ_2 is more of a total metric for perturbations, it can be used to generate all-round better adversarial images. Restricting attacks by ℓ_2 provides adversarial images that are closer to the originals without setting any further requirements such as number of pixels perturbed.

ℓ_∞ metric

ℓ_∞ gauges the maximum perturbation among all pixels. In lay-mans terms, we find the pixel with the largest difference in value from the original, and that difference becomes ℓ_∞. Equation 2.9 shows the formal definition.

$$\ell_\infty = max_i|\delta_i| \tag{2.9}$$

The ℓ_∞ metric ensures that generated perturbations are consistent across the whole image. If one part of the image is more perturbed than others, the entire image might look unnatural. Used in conjunction with ℓ_2 or ℓ_0, ℓ_∞ can be used to provide smooth and natural-looking adversarial images.

2.3.4 Targeted and non-targeted attacks

When attacking a classification model, there are generally two main approaches to choose from depending on the objective. A *non-targeted attack* aims to simply cause the target model to misclassify the input image, without any further requirements regarding the classification itself.

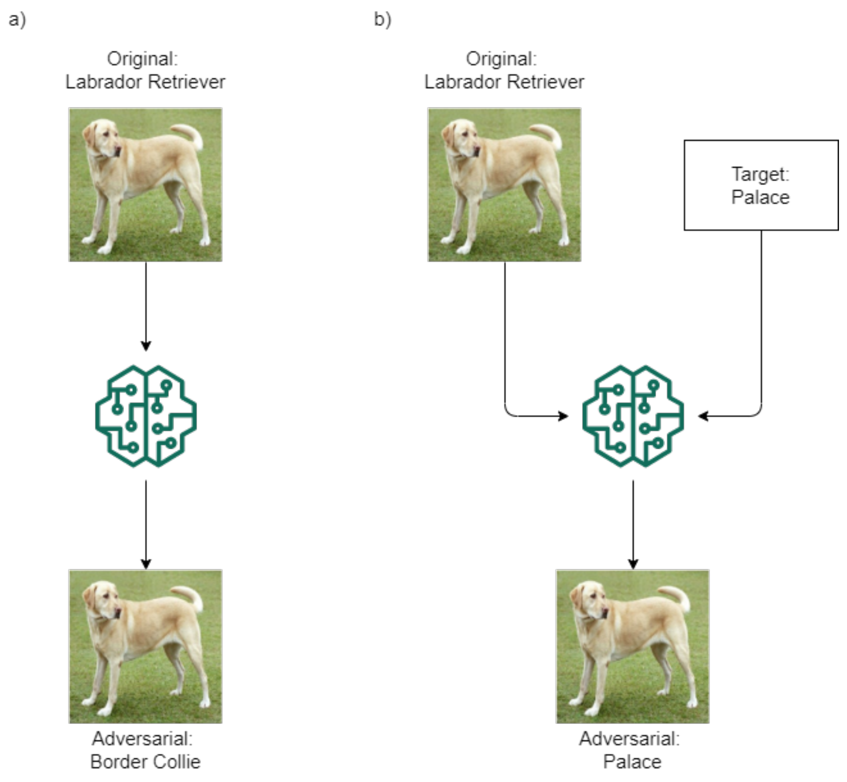
a)
Original: Labrador Retriever
Adversarial: Border Collie
b)
Original: Labrador Retriever
Target: Palace
Adversarial: Palace

Figure 2.15: Illustration of targeted adversarial attacks. a) Non-targeted attack finding the closest misclassification available. b) Targeted attack where the adversarial classification is defined in advance.

Some attack methods allow the attacker to specifically choose which class the image should be classified as. This is known as a *targeted attack*. The downside of targeted attacks is that the perturbations required to make them are generally larger since the decision boundary for the targeted class might be further away. Figure 2.15 shows an illustration of targeted vs non-targeted attacks.

2.3.5 Pixel Based Attacks

Pixel-based (or *pixel-space*) adversarial attacks is a subset of attacks that all perturb images on the pixel level. This means that the perturbation of each pixel is carefully considered independently of other neighboring pixels. Because of this, pixel-based attacks usually create adversarial images with very subtle perturbations, often indistinguishable from the original image by humans.

Pixel space attacks are the most common approach to generating adversarial examples and are typically restricted by ℓ_p metrics to ensure the visual quality of generated images. Famous attacks using this approach are the Fast Gradient Sign Method [7], the Basic Iterative Method [12] and Projected Gradient Descent [26], to name a few. Although adversarials generated by pixel-space methods closely resemble their originals, they are weak against defensive methods such as JPEG compression (see section 2.4.3), Feature Squeezing (see section 2.4.2) and Adversarial Training (see section 2.4.1).

Fast Gradient Sign Method

The Fast Gradient Sign Method [7] is a white-box attack that adds pixel perturbations based on the direction of the gradient. We control the magnitude of the perturbations using the variable ε. Whereas gradient descent is the minimization of the loss function with respect to weights, FGSM is the maximization of the loss function with respect to data. Essentially, our aim is to increase the loss function of the target model, however since we cannot change the weights of the model, we instead have to change the data itself.

$$X_{adv} = X + \varepsilon \cdot sign(\nabla_X J(X,Y)) \tag{2.10}$$

Equation 2.10 shows the idea behind the FGSM method. X and Y here are the original image and label, J is the loss function, and ∇_X is the gradient with respect to X. The FGSM method is not very computationally expensive, so it is commonly used to generate samples for adversarial training.

Basic Iterative Method

The Basic Iterative Method (BIM) [12] extends FGSM to an iterative algorithm where gradient-based distortion is gradually increased until an adversarial image is generated. Equation 2.11 shows the mathematical definition of this method.

$$X_0^{adv} = X, \;\; X_{N+1}^{adv} = Clip_{X,\in}\{X_N^{adv} + \varepsilon \cdot sign(\nabla_X J(X_N^{adv},Y))\} \tag{2.11}$$

Like with FGSM, X and Y are the image and label, J is the loss function and ∇_X is the gradient with respect to X.

Projected Gradient Descent

Projected Gradient Descent (PGD) [26] is a whitebox attack in which we use gradient information to move a perturbation in the direction of the largest loss within a ℓ_p ball. The initial perturbation is chosen at random and the algorithm stops once convergence is reached. While the attack is very similar to BIM, they differ in that BIM does not use a random initial perturbation.

2.3.6 Semantic Attacks

Semantic adversarial attacks, unlike pixel-space attacks, target image features or attributes rather than individual pixels. Modifying these properties sometimes result in large uniform changes to the image as a whole, without the new image looking fake or unnatural. An example of this could be perturbing image colors, something many semantic attacks do.

There exists a broad range of semantic attacks that produce very different results depending on the method used [53, 57, 32, 60, 34]. Many semantic attacks produce adversarial images with a clear distinction from the original in a side-by-side comparison, unlike typical pixel-based attacks. This is by design, as the images don't need to look indistinguishable from the original as long as they look natural to humans. As a result, many semantic attacks tend to be unrestricted by any ℓ_p boundary.

The larger perturbations generated from these methods also make defences such as JPEG Compression and Feature Squeezing less effective. The reason for this is that these defences are designed for small, almost unnoticeable perturbations that have to be very precise in order to cause misclassification. Larger perturbations, like ones created by semantic attacks, are not affected by these defences and are as a result harder to defend against.

On the other hand, unrestricted adversarial examples are more difficult to produce, as the larger perturbations make them more noticeable. To compensate for this, the perturbations need to make semantic changes to the image in order to make the images look natural. Editing image semantics to cause misclassification while keeping the image natural-looking is not easy. The upside is that if successful, the adversarial image is much harder to detect.

The most common way of defending against these types of attacks is Adversarial Training (see section 2.4.1).

2.4 Adversarial Defence

There are many defences designed to improve model robustness to adversarial perturbations. Different defences perform differently on various attacks, and all of them follow the no free lunch theorem. There is no universal 'best defence', nor does any one defence offer only advantages without drawbacks. The choice on which defences to select therefore must be done on a case-by-case basis. The following section will briefly describe some of the most commonly used defence methods against various adversarial attacks.

2.4.1 Adversarial Training

Adversarial training is a very strong but complex defence method where adversarial examples are made less effective by training a classifier to recognize them. Since adversarial training is implemented in the training phase of a neural network, its typically compatible with other defence methods that are applied at other stages.

The theory behind adversarial training is very simple. By training a classifier on both normal and adversarial images, it will be more used to the variations generated by different attacks, and thus harder to fool. This of course requires a dataset that contains a large ratio of adversarial images. Unlike regular datasets, adversarial datasets are attack-specific, meaning classifiers trained on one attack will not perform as well on other ones [47, 49]. As such, using a dataset made by someone else may not provide the expected results.

Of all defence methods developed against adversarial attacks, adversarial training might be the most versatile. Its potential ability to learn the concepts of nefariously doctored images makes it suitable for almost any attack, provided a proper adversarial dataset exists. Adversarial training also provides results that can be hard to exceed for other defence methods.

Although the advantages of adversarial training are many, its downsides are plentiful as well. The vast number of adversarial images needed to form a dataset usable for network training requires a large amount of computer resources and time to generate. For some tasks, proper adversarial training is simply not computationally feasible. One idea to solve this has been using adversarial datasets generated from PGD as a universal defence against ℓ_∞ perturbations [26], however this does not provide defence against other types of attacks. The idea of finding some universal training set however is very intriguing.

Another concern with adversarial training is the potential for negatively impacting model results on non-adversarial images [26, 51, 50]. It goes without saying that learning image perturbations should not take priority over learning image concepts.

2.4.2 Feature Squeezing

Feature Squeezing [33] is a defence framework intended to detect adversarial images while they are being processed by a classification model. The idea behind it all is to limit possible image perturbations by reducing the search space available to them. This is done using a combination of color bit reduction and spatial smoothing.

Color bit reduction is a method where the number of bits used to encode image colors are reduced. In a normal RGB image, color is comprised of red, green, and blue components. The allowed value range for each of these components is 0-255. Thus, the number of bits required to encode each component is 8 bits, making the final color encodable using 24 bits. We refer to the number of bits required per component as the bit depth. By reducing the bit depth, the total number of available colors is reduced exponentially, thereby reducing the color space that an attack method can take advantage of. Calculating new color values for an image with reduced color space is simple and requires very little computer resources.

Spatial smoothing (or *median smoothing* or *median filtering*) uses neighboring pixels to average out colors, creating an image blur effect. This effect removes sharp color differences to some degree, thus evening out any would-be perturbation on the image. The effect is applied using a filter, the size of which determines how many neighboring pixels to use when calculating the average, by extension increasing the magnitude of the blur.

Since both color bit reduction and spatial smoothing only take an image as input, the defence can be implemented at run-time during the pre-processing step. Feature Squeezing uses only simple techniques, thereby using very little computer resources. Figure 2.16 shows the effects of Feature Squeezing when applied to an image.

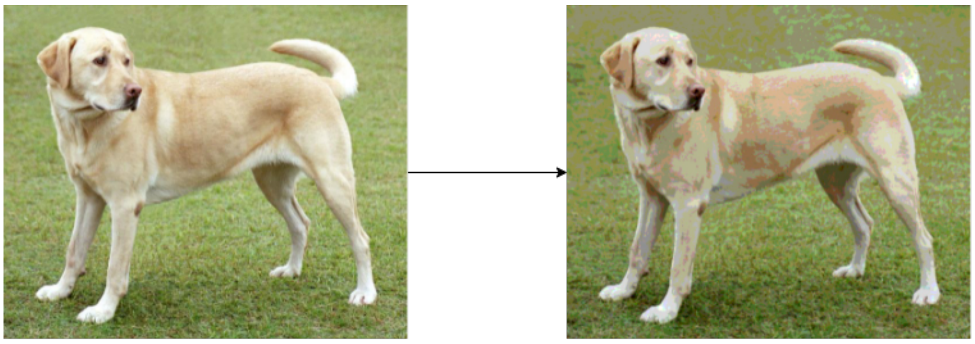

Figure 2.16: Example of Feature Squeezing defence using bit_depth=3 and filter_size=3.

To find out whether an image is adversarial, the classification model predicts the class of both the input image and the feature squeezed image. If the prediction is different, the image is assumed to be adversarial. This helps differentiate misclassifications caused by nefarious perturbations and ones caused by inherent model inaccuracies.

2.4.3 JPEG Compression

JPEG is a very common image file format that uses a compression algorithm for saving space on a disk. Although the compression algorithm used by JPEG is called JPEG Compression, the actual algorithm itself is used in many other applications as well. In adversarial defence, JPEG Compression is used to remove or ruin small perturbations from an image [10]. This defence is rather easy to implement and simply converting the image to a JPEG file format is in many cases all it takes. This is also very efficient and much like Feature Squeezing, the defence is applied at run-time.

Although the defence is efficient against small and precise perturbations, larger perturbations may be less affected depending on the compression magnitude [53]. Compression algorithms are designed to reduce file sizes and will generally try to avoid causing visual artifacts in an image while doing so. If a perturbation is large enough to cause a semantic change in the image, compression may not be able to make said perturbation go away.

Chapter 3

Related work

This chapter covers previous work related to the topics of this thesis. The first part covers existing adversarial toolboxes, while the second part covers published articles related to adversarial training.

3.1 Adversarial Toolboxes

Different adversarial toolboxes have been around for a few years already and provide both researchers and industries a method for testing and improving robustness against adversarial attacks. This section covers a few adversarial toolboxes that have been developed over the past few years.

3.1.1 Adversarial Robustness Toolbox

The Adversarial Robustness Toolbox (ART) [27] is a toolbox created by IBM with the purpose of providing both developers and researchers alike with easy-to-use adversarial attacks and defenses. The toolbox contains several different types of attacks, including evasion, poisoning, extraction, and inference attacks.

ART is developed for as many use-cases as possible, supporting many AI frameworks for images, audio, and video. It also covers adversarial generation for several different problems like classification, speech recognition, certification, etc. In order to properly support all of these different scenarios, ART offers a wide range of adversarial attacks. As a result, the adversarial attacks supported for image classification are mostly big-name attacks with ℓ_p-norm constraints.

3.1.2 AdvBox

AdvBox [55] is a toolbox developed by Baidu for generation, detection, and protection of adversarial attacks. The toolbox supports an assortment of common adversarial attacks such as FGSM [7], C&W [15], DeepFool [13], PDG [26] and BIM [12]. These attacks all use ℓ_p norm bounds in some way, and no unrestricted adversarial attacks have to this day been implemented in the toolbox. The adversarial defenses supported by AdvBox offer different approaches to improve model robustness.

3.1.3 AdverTorch

AdverTorch [37] is a PyTorch-based adversarial toolbox developed for robustness research. Like previous toolboxes, its main focus is on attacks bounded by ℓ_p norms. The toolbox features attacks such as C&W [15], FGSM [7], PDG [26] and BIM [12]. It also features one unrestricted adversarial attack named stAdv [32].

3.1.4 FoolBox

FoolBox [18] is an adversarial toolbox supporting several deep learning frameworks such as Keras, PyTorch, and Tensorflow. In addition, the toolbox features a long list of different adversarial attacks. According to its original paper, it is built around the idea of finding the minimal perturbations possible. As a result, most of the featured attacks rely on ℓ_p norms for minimizing perturbations.

3.1.5 CleverHans

CleverHans [29] is an adversarial toolbox featuring a multitude of different restricted adversarial attacks on many different ML frameworks. Like most previously mentioned toolboxes, CleverHans features big-name attacks like FGSM [7] and BIM [12], as well as many others.

3.1.6 Semantic Attack Implementations

While many adversarial toolboxes already exist, many of them provide solutions to the same problem. This can be demonstrated by looking at which attacks are implemented in each of them. Many of these toolboxes share the same attacks and defences, and sometimes even the same ML frameworks. This isn't that strange however, since attacks like FGSM [7] and PGD [26] are among the biggest and most recognized restricted adversarial attacks.

Unrestricted adversarial attacks however are not well represented in most toolboxes. Table 3.1 shows the current implementation status of various semantic attacks in different toolboxes. Looking at this table, it is clear that a toolbox specializing in unrestricted attacks can be of great benefit to further research in the field and improve robustness against semantic attacks in industrial settings.

3.2 Adversarial Training and Robustness for Multiple Perturbations

Tramèr and Boneh [49] is an interesting read on model robustness towards several different types of perturbations at once. The paper starts out by defining *mutually exclusive perturbations* as perturbations whose robustness implies vulnerability to another perturbation. It is later argued that ℓ_∞ and ℓ_1 perturbations are indeed mutually exclusive, and that ℓ_∞ and spatial transformations are very close.

The paper later attempts to show this empirically by using multi-perturbation adversarial training to see how much robustness can be achieved against these perturbation types simultaneously. Two different adversarial training schemes are proposed in this endeavor. These are as follows:

	ART	**AdvBox**	**AdverTorch**	**FoolBox**	**CleverHans**	**Public Implementations**
Engstrom et al. [38]	Yes	No	No	No	No	2 repos
stAdv [32]	No	No	Yes	No	Yes	1 repo
ADef [35]	No	No	No	No	No	2 repos
ManiFool [25]	No	No	No	No	No	0 repos
Hosseini et al. [24]	No	No	No	No	No	1 repo
cAdv [53]	No	No	No	No	No	1 repo
ColorFool [60]	No	No	No	No	No	1 repo
ACE [64]	No	No	No	No	No	1 repo
ReColorAdv [43]	No	No	No	No	No	1 repo
Inkawhich et al. [41]	No	No	No	No	No	0 repos
Xu et al. [63]	No	No	No	No	No	1 repo
SemanticAdv [57]	No	No	No	No	No	1 repo
Joshi et al. [42]	No	No	No	No	No	1 repo
tAdv [53]	No	No	No	No	No	0 repos
Zhao et al. [34]	No	No	No	No	No	2 repos
Jain et al. [56]	No	No	No	No	No	0 repos
Wang et al. [61]	No	No	No	No	No	0 repos
EdgeFool [58]	No	No	No	No	No	1 repo
FilterFool [59]	No	No	No	No	No	0 repos

Table 3.1: Summary of which semantic adversarial attacks are implemented in different toolboxes, along with online availability.

Max strategy Train on the strongest example across all the different attacks.

Average strategy Train on all examples generated from the different attacks.

Using both of these strategies, the paper shows that there is indeed a robustness trade-off between different perturbation types. The evidence used in this conclusion are accuracy results among differently trained classifiers, showing that multi-perturbation trained classifiers achieve lower results against individual attacks than their single-perturbation trained counterparts.

3.3 Robustness May Be at Odds with Accuracy

Tsipras et al. [50] seek to evaluate the costs of adversarial training, specifically whether robust classifiers are preferable to regular classifiers in every regard in terms of performance. The authors find that standard features and robust features might be inherently at odds and argues the existence of a robustness trade-off, where adversarial classifiers perform worse on standard datasets. They also argue that robust classifiers are more aligned with human perception.

The arguments posed in the paper are backed up by mathematical proof. In a distribution of images, there are different features correlated or not correlated with different labels. Some of these features will be highly correlated, whereas others will be weakly correlated. A standard classifier will rely upon both the highly and the weakly correlated features for correctly classifying images.

The authors claim that an ℓ_∞ bounded adversarial attack may be able to shift the correlation of weak features, effectively making them anti-correlated to a specific class. In such cases, the feature will have an adverse effect on standard accuracy, which relies upon them to at least some degree. The authors then propose a mathematically provable theorem stating that a trade-off between accuracy and robustness exists for ℓ_∞ bounded adversarial images.

Chapter 4

Method

This chapter is split into three main parts, where each part describes the methodology and setup used in each separate research question. The first part covers the preparation and setup of the literature review. Part two covers the design and implementation of the semantic adversarial toolbox. The final part describes the training and evaluation setup for the integrated adversarial training scheme.

4.1 RQ1: Literature Review

Adversarial attacks are a relatively new field, with semantic adversarial attacks being introduced only a few years ago. The past couple of years have seen a large number of new articles exploring the possibilities of introducing semantic changes to fool a classifier. Because the field is so young, the need for a literature review summarizing the current state-of-the-art was great, as it is constantly changing. This section covers how this literature review was conducted, along with descriptions of how decisions along the way were made.

4.1.1 Process

The purpose of the literature review was to collect articles on the topic of semantic adversarial attacks relevant to the research questions defined in section 4.1.2. Before the review could be conducted, minor details regarding the process needed to be defined, such as which sources to use and how to decide which articles were relevant. The sources used for finding relevant material were as follows:

- Reference Lists
- Citation Lists
- Google Scholar
- Oria

For citation lists, Google Scholar was primarily used, which contains a large catalog of scientific articles as well as having great search functionality. Oria, which is the go-to scientific search engine for NTNU, was primarily used for finding papers by keywords. When using search engines, different permutations of the keywords described in section 4.1.3 were used as search queries. Irrelevant articles were filtered using the methods described in section 4.1.4.

In order to keep the review structured, the process was divided into 3 stages. At stage one, keywords were extracted from initially provided research material. These keywords can be found in section 4.1.3 along with the method used to extract said keywords from relevant articles. At stage two, scientific search engines were used to manually find more relevant material. Keywords were used to create queries for searching, and these queries were used as inputs to both Google Scholar and Oria. For each search, the top 100 articles were quickly evaluated for their relevance to the review topic. Most searches had thousands of results, so evaluating all of them was simply not possible. Each article that initially looked relevant was added to a spreadsheet to keep track of them. Stage three involved forwards and backwards snowballing from the results of stage two. Snowballing is a method for finding similar or relevant articles by looking through citation and reference lists. From this, several new articles were identified, whose citations and references were again evaluated.

From these 3 steps, several articles were found, though not all were included in the final review. In order for the literature review to be accurate, it was deemed more important to have all relevant material rather than having no irrelevant material. As a result, some articles were initially included that were later removed following more thorough reading. To keep track of the review results, the final articles were added to a spreadsheet along with summarizing information about each one in regards to the research questions. This spreadsheet can be found in Appendix B.

4.1.2 Research Questions

Not to be confused with the research questions defined in section 1.3, the research questions for the literature review defined its purpose and were the questions to answer through its conduction. The three research questions defined for the review were as follows:

1. **RQ1.1:** What are the current state-of-the-art semantic attack technologies?
2. **RQ1.2:** What are the current state-of-the-art adversarial defence methods for semantic attacks?
3. **RQ1.3:** How well are semantic attacks represented in current adversarial toolboxes?

4.1.3 Inclusion Criteria

In order to decide which articles were relevant for the review, some form of criteria was needed. For this purpose, keywords were used. These were words that were frequently used in known relevant articles to describe their content and purpose. Before the review had begun, a couple of relevant articles had already been identified, which provided a decent starting point to identify inclusion criteria. Most of the keywords were collected from titles and abstracts of articles. The following list contains all the identified keywords that were used for the review:

- Adversarial
- Attack
- Semantic

- Feature
- Natural

Keywords such as adversarial, attack, and semantic were of course expected to yield relevant results given the review topic. Other keywords, like feature and natural were also found to be frequently used when describing adversarial attack methods. Semantic adversarial examples typically aim to generate natural-looking images and often perturbs image features in their efforts to do so. It was therefore reasonable for these keywords to be usable in querying for relevant material.

4.1.4 Exclusion Criteria

In addition to inclusion criteria, exclusion criteria were used to exclude certain articles from the review. This was a necessity as the keywords by themselves did not make a great filter for which articles were relevant. Since this review was on the topic of semantic adversarial attacks, articles describing adversarial attacks without applying any semantic perturbations were excluded.

Additionally, articles that did not propose some method for creating adversarial examples were also excluded. These were typically articles that focused on measuring adversarial stealthiness or robustness, rather than actually generating new examples. Some of these articles did have a focus on semantic adversarial examples, but were excluded anyway since they weren't relevant to any of the research questions.

Lastly, only semantic attacks applicable to image classification tasks were considered relevant. As a result, articles describing semantic attacks not applicable to image classification were excluded.

4.2 RQ2: Semantic Adversarial Toolbox

The toolbox for generating semantic adversarial examples was nicknamed the *Semantic Adversarial Toolbox*. This section covers how the Semantic Adversarial Toolbox was structured and implemented.

4.2.1 Toolbox design

Before the start of the development, a few key design decisions had to be made to ensure the quality of the toolbox. The following paragraphs touch on the most crucial decisions made regarding structure and design:

The defences and attacks needed to be interchangeable. One of the most important requirements of the toolbox was being able to pick and choose between which attack to run and which defence to use as a countermeasure. Every defence therefore needed to be compatible with every attack. To achieve this, each attack and defence needed to follow a set procedure and utilize the same input and output structures.

No attack could be dataset or image specific. Every attack was required to produce a result regardless of input content. Consider an attack that generates adversarial images by modifying facial features. Such an attack would undoubtedly be classified as a semantic attack and might even perform very well, but would be restricted to only working on images with faces. These kinds of attacks could not be implemented in a toolbox where any kind of image could potentially be used as input and were therefore excluded. Attacks assuming input of a specific dimension size were also excluded. The exception to this rule was the assumption of each image containing three color channels. This was seen as a necessary requirement to have, as color is a large part of image semantics.

Versatile adversarial training. Adversarial training is a complicated subject with many different variations and theoretical paradigms. Toolboxing adversarial training would therefore never be a straightforward task. It was important that the final implementation offered as much customization as possible to allow for as much versatility as possible. One of the requirements to achieve this was supporting multiple attacks simultaneously, which allowed for integrated adversarial training to be conducted.

PyTorch

PyTorch [45] was developed by Facebook and offers both high and low-level customization of neural networks. For adversarial attacks, the ability to make low-level changes to both models and training procedures made PyTorch a very viable option. In recent years PyTorch has seen widespread use in AI research papers, making it a very familiar framework among researchers and scholars. As a result, many adversarial attack implementations use PyTorch as well, which provided great benefits to the toolbox development. Lastly, personal experience with PyTorch had a big influence on the framework selection.

4.2.2 Attack Implementation Priority

Due to the sheer amount of different semantic adversarial attacks introduced in research literature in recent years, it was clear from the start that not all of them could be implemented in the available time frame. Deciding on the priority of attacks to implement therefore was an important issue to get right. The final attacks implemented in the toolbox were chosen by several factors regarding both their performance and recognition.

Attack similarity

The variations among different adversarial attacks were an important factor when choosing which of them to implement. Although no adversarial attacks are exactly similar, many of them target similar features. During the literature review, it was quickly discovered that many semantic attacks would target colors in some fashion. In a hypothetical scenario, if only two attacks could be chosen, it would be unwise for both of them to be based on color perturbations when other more varied attacks could take the place of one of them. The results of the literature review (see section 5.1) provided useful information in categorizing different adversarial attacks which were later used to select attacks to implement.

Success rate against adversarial defences.

The resistance to various defensive measures was greatly varied among different attacks. While some had success rates as low as 27%, others were upwards of 90%. Attacks that were more resistant to various defence methods would have a lot of potential for defensive improvement.

These attacks would also be a larger risk in a real-world scenario due to their robustness. For these reasons, attacks with higher success rates were prioritized.

Number of citations

One of the main goals behind the adversarial toolbox was to provide the ability to test model robustness and improve them through adversarial defence methods. With this goal in mind, the provided adversarial attacks needed to allow users to prepare for real-world attack scenarios. Although any attack could realistically be used in a real attack, ones with more widespread recognition would be more likely. As such, attacks with more citations were more prioritized to be implemented, as these were the ones most likely to be used in a real-world adversarial attack.

Implementation status among other adversarial toolboxes.

Since numerous adversarial toolboxes were already in existence at the start of the project, attacks already provided by one of these were less prioritized. Fortunately for most attacks, this was not an issue to consider (see Table 3.1).

Visual quality

Although no real metric for judging the quality of semantic adversarial examples has been decided upon as of writing, the visual aspect of generated images had to be of consideration. Adversarial examples with good-looking perturbations have a higher potential to cause real-world problems, so these attacks had to be favored more. Due to the lack of a metric for adversarial quality, the visual quality was judged by eye in correspondence with supervisors and assistants.

4.2.3 Implemented attacks

Based on the criteria defined in section 4.2.2, 4 different semantic adversarial attacks were eventually implemented in the final toolbox. To ensure the implemented attacks were sufficiently dissimilar, the first three implemented attacks all belonged to different categories defined in section 5.1. Table 4.1 shows a quick summary of the attacks in context of the different factors described in section 4.2.2. In this table, 'Visual Quality' and 'Citations' were simply given a rating of Low, Medium, or High where higher means better. For the 'Current Defence Success' row, the same rating system was used, however in this case lower means better. The 'Attack Category' refers to the attack categorization identified in section 5.1. Implementation status refers to which other Adversarial toolboxes have previously implemented the attack.

	HSV	RT	EdgeFool	ColorFool
Visual Quality	Medium	Medium	High	High
Citations	Medium	High	Low	Low
Current Defence Success	Medium	Medium	Low	Low
Implementation Status	None	ART [27]	None	None
Attack Category	Color	Geometric	Feature Manipulation	Color

Table 4.1: Summary of implemented attacks.

HSV Attack

The HSV attack [24] was the first attack implemented in the toolbox. As explained more deeply in section 5.2.1, this attack perturbs an image by uniformly adding random values to the colors in HSV color space. The visual quality of the adversarial images was found to be variable depending on the random perturbation values. While the original paper was one of the first to introduce semantic adversarial attacks, it had not been implemented in any previous adversarial toolbox. The attack itself was also a very good fit for the first attack implementation due to its inherent simplicity. Defensively, the original adversarial training scheme described in the paper provided decent results but still had room for improvement.

RT Attack

Among unrestricted adversarial attacks, the RT attack [38] is one of the most recognized, having been used in previous defence schemes [49]. The visual quality of the attack is good, and while the defence described in the original paper provided decent results, it also had room for improvement. Although the attack had already been implemented in the Adversarial Robustness Toolbox [27], it was still chosen due to a lack of better alternatives in the geometric transformation category (see section 5.1.1).

EdgeFool

While EdgeFool [58] may not have been as widely recognized as the previous two attacks, its generated perturbations were unique in style and of great quality. In terms of existing defences, the original paper never made any claims regarding adversarial training, but showed Feature Squeezing (section 2.4.2) to be somewhat ineffective.

ColorFool

ColorFool [59] was the second color-based adversarial attack implemented. Like EdgeFool, ColorFool was a more recently introduced adversarial attack with very good-looking perturbations. For this reason, the attack was also not implemented by any other toolboxes. The defence proposed in the original paper was shown to have large room for improvement.

4.2.4 Challenges

Image normalization is the process of 'centering' image color values using the mean and standard deviation of a dataset. For a long time this has been a part of the standard data preprocessing procedure and is known to improve model performance. As such, it was natural for the adversarial toolbox to reap the benefits of this by applying it to images to be perturbed. Doing so however, turned out to be a lot more difficult than originally anticipated.

A problem with normalizing images for adversarial attacks is that perturbations applied to these images make the normalization irreversible. In simple terms, image normalization is merely a mathematical operation which in most machine learning scenarios can be reverted by an inverse calculation. The problem with this approach in an adversarial attack context is that once the image is perturbed, the inverse function is invalid. This issue is illustrated in Figure 4.1, where a normalized image is perturbed in the color space. The resulting adversarial image looks unnatural and reverting the normalization function does not fix the issue.

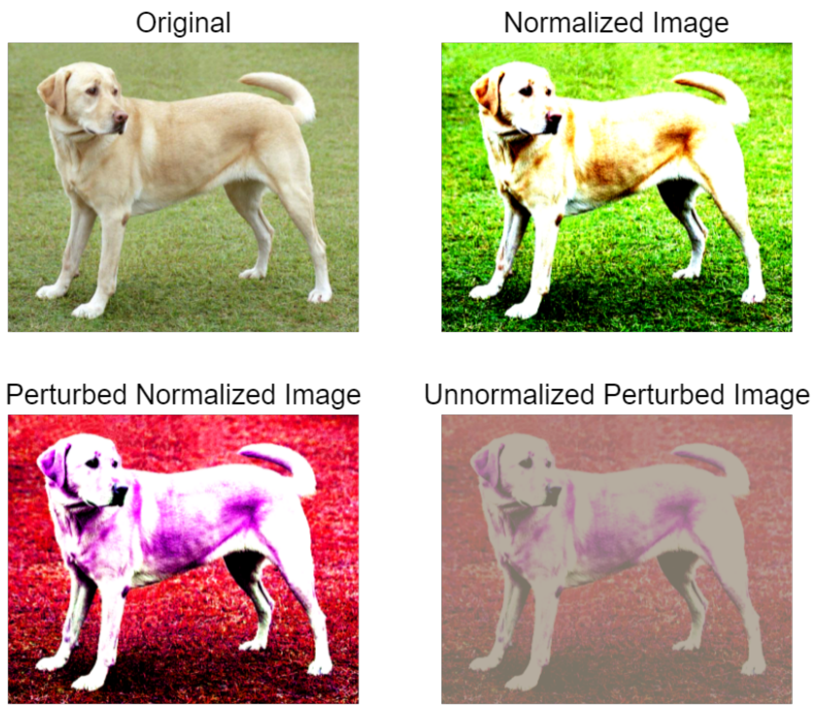

Figure 4.1: Effects of adversarial attacks on normalized images.

Of course, this problem only arose due to images being normalized before performing the attack. In PyTorch, image normalization is usually done during the initial loading of a dataset, however it is only needed during the forward pass of a neural network when calculating image predictions. As a result, if images were normalized immediately before being predicted, any perturbations made before that would be revertible. A side effect of this approach was requiring a separate normalization function as input for each attack in the toolbox. Although the solution could be considered confusing for many users, it did not have any performance-related impact.

Another challenge during the development was keeping track of checkpoints during training and generating adversarial images. Producing an entire dataset of perturbed images for adversarial training could potentially take a significant amount of time depending on the attack. A concern with implementing checkpoints however was that such functionality would be outside the scope of the project and cause a variety of I/O-related bugs. Eventually, a checkpoint system was still implemented out of necessity, as some attacks were not quick enough to generate an adversarial dataset in one sitting.

4.3 RQ3: Integrated Adversarial Training

This section covers the setup and theory behind the adversarial training scheme for defending against multiple adversarial attacks simultaneously. Both the training and evaluation setup is discussed, along with both training and attack parameters.

4.3.1 Training strategy

Our adversarial training scheme borrowed some elements from previous work [49] and from the field of Robust Optimization (see section 4.3.3). In total, three different unrestricted semantic adversarial attacks were targeted for adversarial training, both individually and collectively. The attacks in question were the HSV attack [24], the RT attack [38] and EdgeFool [58]. These attacks were chosen specifically for their diverse strategies of perturbation, as illustrated in Figure 4.2. The hypothesis was that these attacks were different enough to have almost no overlap in perturbations and would therefore be compatible in a multi-attack adversarial training scheme.

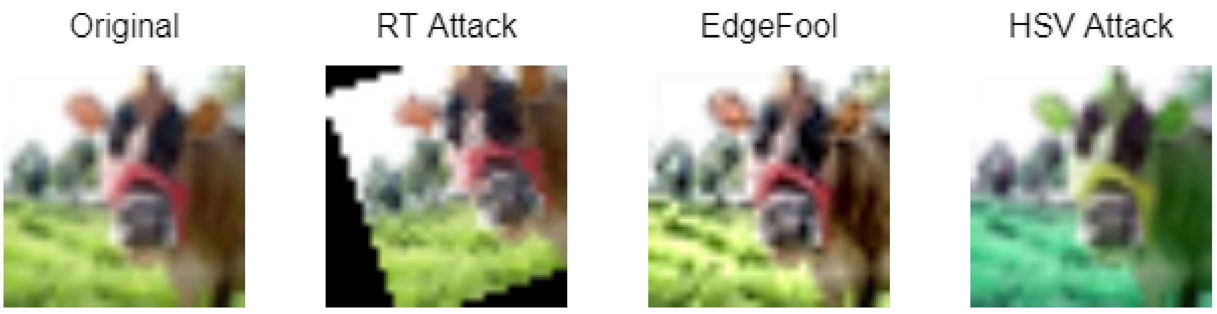

Figure 4.2: Different variations of same CIFAR100 image generated from different attacks.

The experiments were performed on both the CIFAR10 and CIFAR100 datasets. All experiments were performed using the ResNet50 architecture (see section 2.1.10). ResNet50 was chosen for being a good middle-of-the-pack architecture, neither too simple nor too complex.

In order to eliminate other potential factors in the results, all models were trained the exact same way using the same parameters, with the exception of datasets which of course included samples from different adversarial attacks. In total 6 different classifiers were trained for CIFAR100 and CIFAR10, of which 2 featured no adversarial training, 3 featured single-attack adversarial training, and 1 featuring multi-attack adversarial training. The multi-attack model was trained on samples from all three attacks simultaneously.

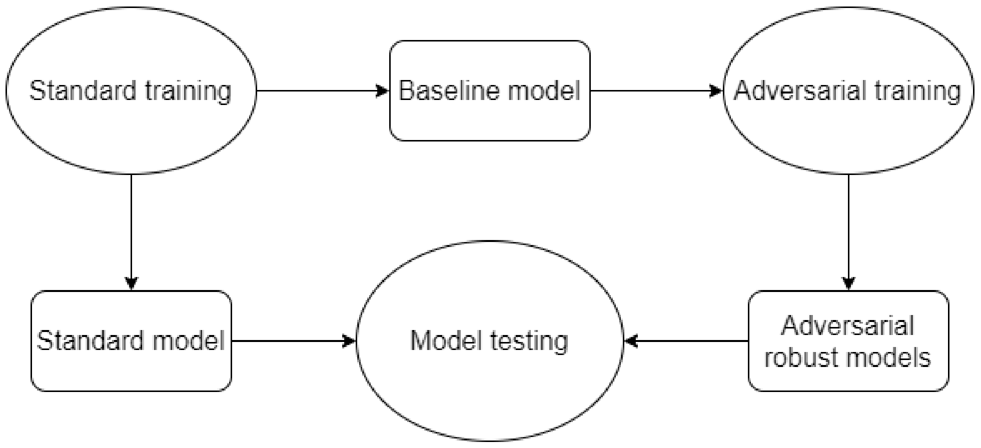

Figure 4.3: Flow chart of training and testing process.

In order to have a fair comparison between the standard model and the robust models, two separate baseline models were needed. One of these was used as the target model for the adversarial attacks and would therefore have terrible accuracy on the generated images, as the generated images were by definition adversarial to the target model. The other one was used when comparing results with the other robust models. Figure 4.3 visualizes this process from start to finish.

For the adversarial training, the ratio of adversarial images to non-adversarial images was 50% of the original dataset. Since both CIFAR10 and CIFAR100 consist of 50 thousand training images, the number of adversarial images generated was 25 thousand per attack. The adversarial images were merged with the original datasets without replacement. Single-attack adversarial datasets therefore consisted of a total of 75 thousand images, 25 thousand of which were adversarial images. The multi-attack dataset consisted of 125 thousand images, 75 thousand of which were adversarial images. 10% of the adversarial datasets were used as a validation set during training.

This strategy follows the 'avg' strategy introduced by Tramèr and Boneh [49], in which all samples from different adversarial attacks are trained on simultaneously. Previous work has shown that larger datasets generally provide better test results [19]. In the context of adversarial robustness specifically, it has been theorized that larger datasets can lead to more robust models [30]. As such, merging the original and adversarial dataset into one bigger mixed dataset is a reasonable strategy.

The HSV attack [24] was set to run for a maximum of 100 iterations. For the RT attack [38], the 'worst-of-k' method described in the original paper was used with a total of 100 iterations. The maximum rotation was set to $\pm 30\,\text{deg}$, whereas the maximum translation was set at $\pm$ 20% of both width and height. EdgeFool [58] was ran for a maximum of 1000 iterations, with a set minimum loss of 0.005 for accepting the adversarial.

4.3.2 Evaluation strategy

The standard model and the different robust models were all evaluated for both standard testing accuracy and adversarial testing accuracy. Standard accuracy was evaluated simply by testing on the regular testsets provided by both CIFAR10 and CIFAR100. These same testsets were then used to generate adversarial testsets for each different attack. Like with the training datasets, a separate baseline model was used as the target model during this process. The attack parameters used were the same for both the training set and the testing set. Each model was then evaluated on each adversarial testing set.

4.3.3 Robust Optimization

Although the usage of robust optimization in adversarial training is nothing new, it has been shown to yield better test results against nefariously perturbed images [26, 38, 9, 31, 16]. Robust optimization in an adversarial training sense can be seen as minimizing the loss of worst-case data inputs. In most cases, worst-case data is defined as the perturbed data that yields the maximum loss across all possible perturbations [31, 16, 26].

In our case however, this definition was not applicable due to the unrestricted perturbations generated by semantic attacks. In most previous work combining robust optimization with adversarial training, a general assumption had always been perturbations bounded by some ℓ_p norm [31, 16, 26]. Of course, in such a scenario, one could easily define a worst-case perturbation on the input while still ensuring the visual quality of the adversarial image. Unfortunately, a worst-case unrestricted perturbation would very likely not be of acceptable quality.

To counteract this issue, the definition of a worst-case input was changed to the adversarial image with the most natural-looking perturbation. This definition is justifiable, as any other definition would incentivize perturbations of poor visual quality which we know by the 'garbage in, garbage out' concept would produce sub-optimal results. To incentivize natural-looking adversarial images, each attack used a custom score function to judge the magnitude of the perturbation. For the HSV attack, which used randomly sampled perturbation values, the score function simply kept track of the smallest sampled value. The perturbation generated by the RT attack could be seen as the union of some rotation in degrees and some translation in pixels, which could then be summed to get a final score. EdgeFool was an exception and did not need a score-function, as the attack inherently looked for the best perturbation regardless.

4.3.4 Cifar10

For all models trained on CIFAR10, the following training parameters were used:

- **Epochs:** 200
- **Batch size:** 128
- **Optimizer:** Stochastic Gradient Descent
  - **Learning rate:** 0.1
  - **Momentum:** 0.9
  - **Weight Decay:** 0.0005
- **Loss function:** Cross-entropy
- **Scheduler:** Cosine Annealing
  - **Tmax:** 200

- **Transforms:**
  - Random Crop (Training)
  - Random Horizontal Flip (Training)
  - Normalization (Training and Testing)

Although these parameters yielded fairly good standard accuracy for CIFAR10, the most important results were the relative accuracy between the different adversarially trained classifiers.

4.3.5 Cifar100

The training parameters for CIFAR100 for all trained models were identical to avoid deviation of results caused by different training methodologies. The parameters used were as follows:

- **Epochs:** 200
- **Batch size:** 128
- **Optimizer:** Stochastic Gradient Descent
  - **Learning rate:** 0.1
  - **Momentum:** 0.9
  - **Weight Decay:** 0.0005
- **Loss function:** Cross-entropy
- **Scheduler:** Multi-step
  - **Gamma:** 0.2
  - **Milestones:** 60, 120, 160
- **Transforms:**
  - Random Crop (Training)
  - Random Horizontal Flip (Training)
  - Random Rotation (Training)
  - Normalization (Training and Testing)

In addition to the normal training scheduler, an additional warm-up scheduler was also used for a single iteration. As mentioned in section 4.3.4, the baseline performance was not the focus of the experiments and the training parameters may therefore not be perfectly optimized.

Chapter 5

Results

This chapter describes the results of each separate research question. The first part of the chapter covers the results of the literature review, followed by a detailed description of the final state of the Semantic Adversarial Toolbox. Finally, the results of the integrated adversarial training is described.

5.1 RQ1: Literature Review

Throughout the three stages of finding material, 33 articles were found to be initially relevant. On closer inspection, 14 of these were later found to be irrelevant to the research topic, leaving 19 final articles for the literature review. Table 5.1 shows the number of articles found from each source defined in section 4.1.1.

Source	Num articles
Backwards Snowballing	4
Forwards Snowballing	4
Oria	7
Google Scholar	4

Table 5.1: Literature review results per source.

These numbers can be deceiving, as several articles were found in search results from both Oria and Google Scholar. In these instances, the first appearances of the articles were the ones that counted. To make the search as systematic as possible, Oria was given first priority, with Google Scholar only being utilized afterward. As a result, the number of articles stemming from Oria was deceivingly larger than the ones stemming from Google Scholar. Likewise, since snowballing was not utilized until stage three, these numbers were also smaller compared to Oria.

The papers were grouped in three main categories identified during the search and encompassing all of the papers in some way. These categories were geometric transformation, color modification, image feature manipulation. Most of the articles were placed in the image feature manipulation category, encompassing 10 of the 19 papers. Only 4 papers described adversarial attacks using geometric transformations, leaving 5 papers in the color modification group.

For each category, there is a quick overview summarizing some parts of the different attacks. These are Table 5.2, Table 5.3 and Table 5.4. The tables contain information on whether the attacks are whitebox or blackbox, whether they support targeted attacks, and information about defence. For the "defence metric" columns, the metric provided is the greatest robustness achieved against a given attack, regardless of network architecture. Some papers provide different statistics for attacks using different hyperparameters. In these cases, the metric provided is based on the best performing attack, essentially pitting the best attack against the best possible defence.

5.1.1 Geometric Transformation

Although geometric transformations were not the most popular subject for adversarial attack purposes, they stood out among other approaches for their usage of simple image processing methods. This category encompassed transformations primarily to an image's rotation, size, and shape. Figure 5.1 shows a few examples of adversarial images using this approach, originally created by Engstrom et al [38].

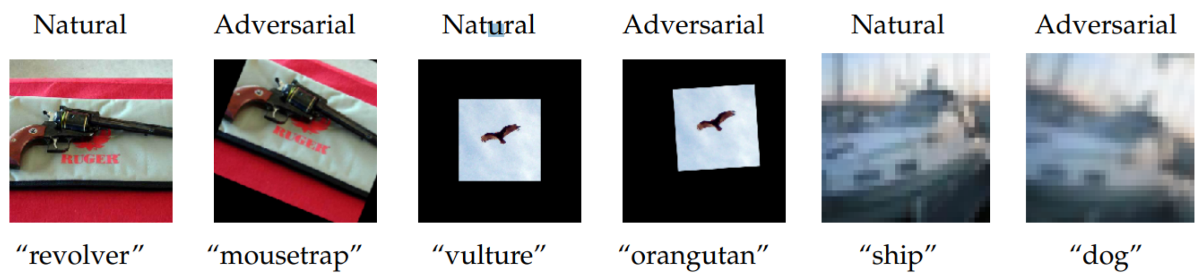

Figure 5.1: Adversarial examples using geometric transformations. The figure originates from work made by Engstrom et al. [38]

Table 5.2 shows a quick summary of all the attacks in this category. The attacks are generally of the whitebox variety, aside from the attack described by Engstrom et al. [38], which provides both whitebox and blackbox approaches. Papers focusing on geometric transformation attacks seem to be generally older than papers belonging to other categories, which could explain why some of the attacks feel a bit simplistic compared to many other attacks in the other categories.

	Attack Type	Defence	Defence Metric	Targeted class?
Engstrom et al. [38]	Whitebox and Blackbox	Adversarial Training	52% (ImageNet) 71% (CIFAR10) 97% (MNIST)	No
stAdv [32]	Whitebox	Flow Gradient Regularization	64% (CIFAR10) 37% (CIFAR100)	Yes
ADef [35]	Whitebox	Adversarial Training	93% (MNIST)	Yes
ManiFool [25]	Whitebox	Adversarial Training	∼34% (CIFAR10)	No

Table 5.2: Geometric transformation attack overview

EXPLORING THE LANDSCAPE OF SPATIAL ROBUSTNESS [38]
What: Engstrom et al. explores the possibilities of using image transformations such as rotation, scaling, and shearing to transform images into adversarial examples.
How: The paper proposes an attack in which parameters δu, δv and θ, denoting image translation and rotation, are optimized. By rotating the image θ degrees and translating it by $\delta u, \delta v$ pixels, a classifier should incorrectly classify the transformed image. The adversarial images themselves are generated by maximizing these arguments using a loss function $L(x^{'}, y)$ where $x^{'}$ is the transformed image and y is the correct label. The attack is then applied using either a First-Order method, where gradients are used to move towards a local maximum of the loss function, or using a grid search that evaluates all possible parameter combinations.
Defence: Worst-of-10 adversarial training is used to increase spatial robustness and does significantly increase the accuracy of the models. It is shown that the worst-of-10 method on CIFAR10 achieves 92% accuracy for random sampling and 71% for grid search.
Code implementations: The attack has open-source implementations for both TensorFlow and PyTorch. It is also implemented in the Adversarial Robustness Toolbox (ART).

STADV [32]
What: stAdv uses pixel displacement to generate adversarial images. Pixel displacement is a simple concept where pixels are moved around from the original image to another location in the adversarial image. Essentially, the exact same pixels are present in both images, but some pixels could be placed 1 cm to the right, and some others to the left, etc.
How: The objective of the proposed attack is to find a minimal flow field that causes misclassification. This is done through differentiable bilinear interpolation in which every pixel location is calculated by looking at neighboring pixel values. The adversarial image is completed once all the pixel locations have been calculated. Adversarial images can have targeted classes and are generated using an adversarial loss function in combination with a flow loss function that ensures the pixel displacement is minimal.
Defence: Adversarial training is demonstrated to achieve low defence against stAdv, with about 40% of attacks being successful. Later research on defence against adversarial deformations found that flow gradient regularization provided a decent defence with up to 76% accuracy on CIFAR10 [62].
Code implementations: The attack fully toolboxed by both CleverHans and AdverTorch.

ADEF [35]
What: ADef generates adversarial images by deforming them, essentially moving the pixels around to different locations.
How: The key to the attack is finding a deformation that leads to misclassification. Several iterations may be required, and the algorithm gradually increases the image deformation until an adversarial image is created. The deformation itself is done using vector fields.
Defence: The paper shows that networks trained on adversarial examples generated by PGD provide a good defence against the attack, achieving 93% accuracy on MNIST.
Code implementations: Two different attack implementations are present in different GitHub repositories (here and here).

MANIFOOL [25]
What: ManiFool aims to fool classifiers by moving towards the decision boundary using image transformations.
How: A manifold of transformed images should contain some transformed image outside the decision boundary. To find this image, an algorithm iteratively moves towards this boundary whilst staying within the manifold of images. Once the decision boundary is crossed, the algorithm finishes.
Defence: The paper shows that adversarial training improves model robustness against the attack, achieving roughly 34% accuracy on CIFAR10
Code implementations: At the time of writing, there is no public code implementation of the attack.

5.1.2 Color Modification

Color modification attacks are semantic attacks that target an image's color space. Certain colors have a high correlation with specific objects and classes, and these attacks exploit this property to generate adversarial examples with natural-looking colors. Figure 5.2 shows an example of adversarial images that have been generated using color modification, originally published in work by Shamsabadi et al. [60]

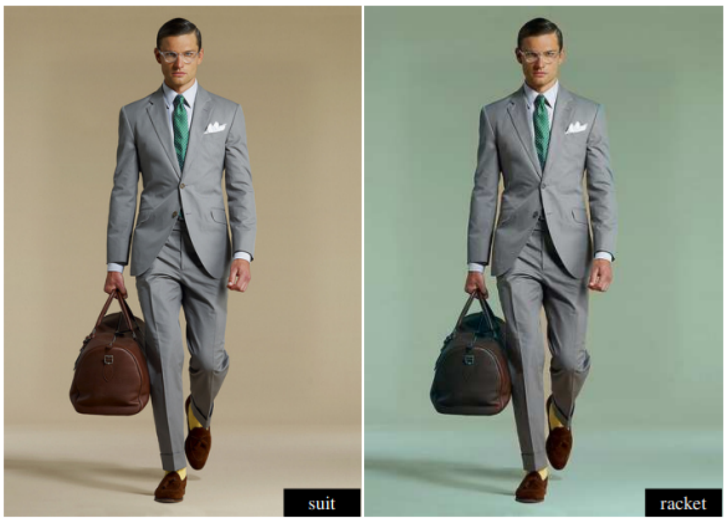

Figure 5.2: Adversarial examples using color modification. The figure originates from previous work by Shamsabadi et al. [60]

From table 5.3, we can note that most color-based adversarial attacks do not offer targeted attacks. Changing image colors can be a challenging affair, as the resulting image runs the risk of looking very unnatural. Targeted attacks add more restrictions to the attack, generally leading to larger changes, which might be undesirable for most color-based approaches. The cAdv attack [53] uses ground truth hints to constrain the image colorization closer to the original image, which may allow them to use targeted attacks without ending up with unnatural colors.

	Attack Type	**Defence**	**Defence Metric**	**Targeted class?**
Hosseini et al. [24]	Blackbox	Adversarial Training	69% (CIFAR10)	No
cAdv [53]	Whitebox	Adversarial Training	88% (ImageNet)	Yes
ColorFool [60]	Blackbox	Re-quantization	∼25% (ImageNet) ∼30% (CIFAR10) ∼30% (Places365)	No
ACE [64]	Whitebox	Unknown	Not specified	No
ReColorAdv [43]	Whitebox	Adversarial Training	59% (CIFAR10)	No

Table 5.3: Color modification attack overview

SEMANTIC ADVERSARIAL EXAMPLES [24]

What: Hosseini et al. introduces a semantic color attack that uses HSV (Hue, Saturation, Value) color space to generate adversarial images. Hue and saturation are given new values such that the new image is misclassified.
How: The attack is realized through random search where values for hue and saturation are chosen from a uniform distribution. If the resulting image is adversarial, the attack stops, otherwise new values for hue and saturation are chosen. The attack is very simple and only supports non-targeted attacks. This inherent simplicity also means that some images have unnatural-looking

colors.
Defence: Networks trained on color-shifted adversarial images do fairly well, with a 69% success rate.
Code implementations: One GitHub repository that implements the attack in Keras, but does not toolbox it.

CADV [53]
What: cAdv uses an image colorization model to create adversarial examples from grayscale images. The paper describes the hypothesis best when it claims that *"Since a colorization network learns to color natural colors that conform to boundaries and respect short-range color consistency, we can use it to introduce smooth and consistent adversarial noise with a large magnitude that looks natural to humans"*.
How: The attack is implemented using a colorization model first proposed by Zhang et al [20]. This model uses colored input hints along with masks indicating the location of said hints. By updating this model to minimize a cross-entropy adversarial loss function J_{adv}, the authors manage to use it for generating colorized adversarial images. The loss function also takes a target class as input, allowing cAdv to perform targeted attacks. Increasing the number of hints gives more natural-looking adversarials, but this works the best when hints are spread out across all segments of the image. To find these segments, K-Means clustering is used.
Defence: cAdv is strong against both JPEG compression and adversarial training. There is however a dropoff between adversarial quality and attack robustness.
Code Implementation: The attack has never been toolboxed, but there exists one GitHub repo implementing the attack for demonstration purposes.

COLORFOOL [60]
What: ColorFool utilizes a methodical approach to color modification, where the first step is determining which parts of the image can be modified without semantic loss. Features like water, vegetation, people, and the sky are not touched, as these are deemed as semantically important to the image. This leaves only the parts of the image whose color is not significantly important, which are then perturbed to generate adversarial images.
How: The first step is to segment the image into sensitive and non-sensitive regions. To achieve this, a Cascade Segmentation module trained on the ADE20K dataset is used. Colors in non-sensitive regions are then modified in the CIELAB color space, where perturbations of the a and b channels are chosen from a natural color range. This process gradually introduces more perturbations until an adversarial image is generated.
Defence: Adversarial defences such as JPEG compression and median filtering are shown to be much less effective on ColorFool than other adversarial attacks, with ColorFool getting approx 70% success rate on these. It is also shown to be quite strong against adversarial training.
Code Implementation: The authors have made an official code implementation that demonstrates the attack in action.

ADVERSARIAL COLOR ENHANCEMENT [64]
What: ACE generates adversarial images using a color filter to introduce natural-looking perturbations. This is done by optimizing commonly used image filters to achieve minimal impact on quality.

How: To make the attack work, the filter is divided into K pieces by a mapping function. This provides K parameters that are optimized using gradient descent. Each piece is adjusted until misclassification occurs.
Defence: No experiments against common adversarial defences are provided.
Code Implementations: The attack is demonstrated using a jupyter notebook. There is also a fully developed script that generates adversarial images using PyTorch.

RECOLORADV [43]
What: ReColorAdv proposes a color modification attack that uniformly perturbs color in an input image using a perturbation function. The attack works for both RGB and CIELUV color spaces.
How: The perturbation function generates the adversarial image by perturbing the color of each input pixel in the same way. This creates an output image that looks more natural. The perturbation function uses a grid of points for which the output parameter is defined. For points outside any of these grids, the function uses trilinear interpolation. An adversarial loss function is used to ensure the output is adversarial.
Defences: The paper shows that ReColorAdv does fairly well against several defence methods such as different adversarial training schemes and the TRADES [52] defence. In particular, combinations of ReColorAdv and other attacks are shown to be better than most other attack methods by themselves.
Code Implementations: The official code implementation offers a good demonstration of the described attack. Currently not toolboxed.

5.1.3 Image Feature Manipulation

The image feature manipulation category encompassed a few different approaches, but with some common elements. Papers in this category all focused on perturbations in the feature space in some way. Typically, an image would be converted into some data structure representing its features, which would then be perturbed. Figure 5.3 shows a few examples of perturbed images generated using this approach, originally published by Qiu et al. [57]

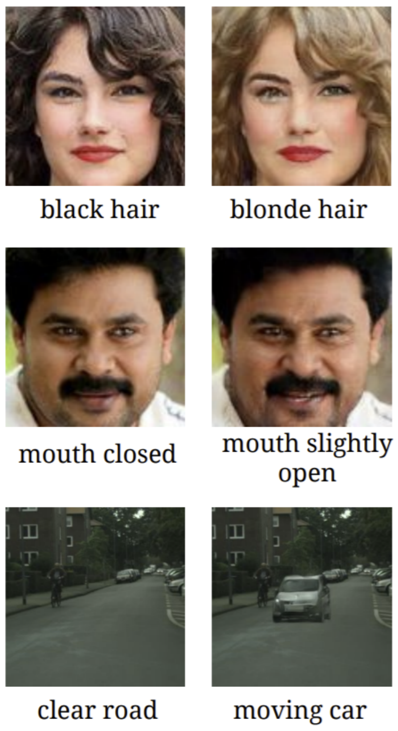

Figure 5.3: Adversarial examples using image feature manipulation. The figure originates from previous work by Qiu et al. [57]

In table 5.4, we see a quick summary of all attacks belonging to the feature manipulation category. Most of these are whitebox attacks, and about 50% of them support targeted attacks. As with most adversarial attacks, adversarial training generally seems to be a good defence against them.

	Attack Type	Defence	Defence Metric	Targeted class?
Inkawhich et al. [41]	Whitebox	Unknown	Not specified	Yes
Xu et al. [63]	Whitebox	Adversarial Training	27% (CIFAR10)	Yes
SemanticAdv [57]	Whitebox	Adversarial Training	10% (CelebA)	Yes
Joshi et al. [42]	Whitebox	Adversarial Training	69% (CLEVR-Singles)	No
tAdv [53]	Whitebox	Adversarial Training	93% (ImageNet)	Yes
Zhao et al. [34]	Blackbox	Unknown	Not specified	No
Jain et al. [56]	Whitebox	Adversarial Training	95% (ShapeNet)	No
Wang et al. [61]	Blackbox	Input De-noising	Not specified	No
EdgeFool [58]	Whitebox	Feature Squeezing	21% (Places365) 21% (ImageNet)	No
FilterFool [59]	Whitebox	Ensemble Defence	68% (ImageNet)	No

Table 5.4: Image feature manipulation attack overview

FEATURE SPACE PERTURBATIONS YIELD MORE TRANSFERABLE ADVERSARIAL EXAMPLES [41]
What: Inkawhich et al. propose an attack that targets the activation pattern of a chosen layer L. Given the image of a dog, we aim to make the activation pattern of the image at layer L similar to that of an image of the target class.
How: The attack itself is similar to that of FGSM (see section 2.3.5) with momentum, but it uses a different loss function that simply calculates the difference in activations at layer L. One of the key focuses of the attack is to make the resulting adversarial example more transferable to a blackbox model. An important distinction of this attack is that the classification error of the target model is implicit, as we do not explicitly aim to create adversarial images. Rather, the adversarials are simply a byproduct of altering the representation at layer L.
Defence: Because the focus of this paper is more about transferability than robustness, there is little discussion around adversarial defence.
Code Implementations: At the time of writing, there are no known public code implementations of the attack.

TOWARDS FEATURE SPACE ADVERSARIAL ATTACK [63]
What: In this attack, the style of an image of a target class is applied to a victim image using style transfer methods.
How: The style is defined as the mean and variance of activations within a layer, whereas the content is defined as the activations themselves. A CNN encoder translates two images (style and content) into features. An integrated embedding is then generated, which retains the content features (activations) of the content image while having the style features (mean and variance) of the style image. A decoder uses this embedding to reconstruct a new image that will look like the content image, but contains the style of the style image. These steps are all part of the training phase. To perform an attack, the image to be perturbed goes into the same system, which outputs a style-transferred image. The target model is then used to classify this new image, and the results are used to calculate an adversarial loss. The adversarial loss is used in conjunction with the style transfer loss to update the embeddings of the system until a successful adversarial image is generated.
Defence: The paper demonstrates that the attack is capable of avoiding two separate detections, O2, [46] and DkNN [28]. It also showcases a select group of defence approaches such as TRADES [52] that fails to defend against the attack. Adversarial defence is shown to be somewhat ineffective, only achieving 27% accuracy on CIFAR 10
Code Implementations: Along with the paper itself, there is an official code implementation which demonstrates the attack in action. Currently, the attack has not been toolboxed.

SEMANTICADV [57]
What: SemanticAdv uses an attribute-conditioned image generator that takes an image and generates a new one based on attributes. For example, we could input an image of a blue car and add the attribute 'red paint' and have a new image generated of the same car but with red paint.
How: The image generator is based on work in the field of semantic image editing and is comprised of an encoder and a decoder, where the encoder creates feature maps from an image, and the decoder synthesizes a new image from an attribute-edited feature map. Adversarial images are generated by using adding attributes from target images to victim images through interpola-

tion.
Defence: The paper compares SemanticAdv to C&W against defence methods such as blurring, JPEG compression, and feature squeezing. SemanticAdv is shown to achieve greater results. It also shows that adversarially trained models only achieve an accuracy of 10%, besting many other adversarial attacks.
Code Implementations: The paper includes an official PyTorch implementation that demonstrates the attack using predefines models. Currently, the attack is not toolboxed.

SEMANTIC ADVERSARIAL ATTACKS: PARAMETRIC TRANSFORMATIONS THAT FOOL DEEP CLASSIFIERS [42]
What: This attack describes an approach where parametric transformation models are used to modify attributes of an image such that adversarial examples are generated.
How: Attributes are defined in a semantic parameter vector where each parameter corresponds to a modifiable attribute. The parametric transformation model is a generative model that is able to perturb some semantic attribute while keeping invariant data unmodified. To realize the attack, the paper describes using either Fader Networks [17] or Attribute GAN [40], both of which are encoder-decoder architectures.
Defence: The paper does not showcase or discuss any defence methods against their described attacks. Gokhale et al. [54] propose an adversarial training method that can defend against this attack, achieving 69% accuracy on the CLEVR-Singles dataset.
Code Implementations: Along with the paper, there is an official code implementation that demonstrates the attacks. Currently, the attack is not toolboxed.

TADV [53]
What: Texture transferring is the idea of transferring textures from one image and applying it to another image. As shown by Bhattad et al, texture transferring can also be leveraged to perform adversarial attacks. This idea stems from the research of Geirhos et al [39], which showed that textures might play a large part in neural networks to determine the classification.
How: The attack itself uses a CNN to extract textural features from a target image. A victim image is subsequently optimized by adding these textural features to it using gram matrices. Since texture transferring is usually used to produce artistic images, the gram matrix of the victim image is constrained to have less variation. The objective function is comprised of a texture loss function and a cross-entropy loss function. It also controls the magnitude of the texture transfer, where a higher magnitude produces more visible perturbation. The target image to be used in the texture transfer is chosen by using nearest neighbors to find the image closest to the victim image in the target class.
Defence: The paper shows that adversarial training is an efficient defense against the tAdv attack, with a misclassification rate of only 4-7%.
Code Implementations: At the time of writing, there are no known public code implementations of the attack.

GENERATING NATURAL ADVERSARIAL EXAMPLES [34]
What: In this attack, a GAN is used to generate new adversarial images from a perturbed vector representation.
How: An image is inverted into a vector, which is then perturbed and reconstructed using

a generator, generating an adversarial image. Two different approaches are presented, iterative stochastic search and hybrid shrinking search. In the iterative stochastic search approach, the magnitude of perturbations are increased until adversarial examples are generated. Hybrid shrinking search searches a wider range of perturbations which are shrunk to find adversarial examples.
Defence: The paper does not discuss the proposed attack against any particular defense method.
Code Implementations: Two different public code implementations exist (here and here), both of which demonstrates the attack in action.

ANALYZING AND IMPROVING NEURAL NETWORKS BY GENERATING SEMANTIC COUNTEREXAMPLES THROUGH DIFFERENTIABLE RENDERING [56]
What: This attack outlines a general method of turning a traditional adversarial algorithm into a semantic equivalent using differential rendering. Differential rendering allows us to "de-render" an image into a vector of image parameters, which can then be tweaked and reconstructed, causing miss-classification.
How: Turning general adversarial attack methods into semantic attacks is achieved in 3 generalized steps. In step 1, the perturbation δ is replaced with the optimized value π. Step 2 replaces $x+\delta$ with $\theta+\pi$. In step 3, the chain rule is used to compute the gradients of terms that involve the R(0) where R is the differentiable renderer.
Defence: The paper shows that adversarial training can help to increase robustness against the proposed attack, achieving 95% accuracy on ShapeNet.
Code Implementations: At the time of writing, there are no known public code implementations of the attack.

GENERATING SEMANTIC ADVERSARIAL EXAMPLES VIA FEATURE MANIPULATION [61]
What: Wang et al. perturbs theme-independent semantic attributes of images in a way that causes misclassification and creates adversarial examples.
How: The two main approaches discussed are vector-based and feature map-based. A disentangled code vector for a given image is produced using Common Feature Variational Autoencoders (CF-VAE). For the vector-based approach, only one CF-VAE is used, whereas the feature map approach uses several. The code vector is then perturbed and decoded, giving an adversarial image.
Defence: The paper describes using input de-noising to defend the attack, but does not give a concrete metric.
Code Implementations: At the time of writing, there are no known public code implementations of the attack.

EDGEFOOL [58]
What: EdgeFool uses a fully convolutional neural network (FCNN) to enhance image details in such a way that the resulting detail-enhanced image is adversarial.
How: A mapping of image details is extracted from the original image using an FCNN and a multitask loss function. The loss function combines both image detail loss and adversarial loss. Before any perturbation is made, the image is converted into CIELAB color space. Only the L-channel of the image is enhanced, which is then combined with the original A and B channel before the image is converted back into RGB color space.

Defence: In the paper, EdgeFool is compared to a range of constrained adversarial attacks, as well as the HSV attack [24] against a feature squeezing framework. EdgeFool is found to be less detectable than constrained methods, but more detectable than SemanticAdv. Although many different metrics are given for different models and parameters, the best defence specified grants a detection rate of 21% for both ImageNet and Places365.
Code Implementations: One official GitHub repository exists, which provide demonstrations of the attack.

FILTERFOOL [59]
What: FilterFool aims to mimic the output of image filters in such a way that the mimicked filter generates an adversarial example. Three different filters are used in this endeavor, namely gamma correction, log transformation, and detail enhancement.
How: The attack is made possible using an FCNN that learns to mimic filter outputs. To achieve this, an objective function composed of a structure loss function and an adversarial loss function is used in backpropagation to update network parameters. The structure loss ensures the output closely mimics that of a selected filter, whereas the adversarial loss function ensures that the output is adversarial.
Defence: The paper shows that feature squeezing and JPEG compression used simultaneously provides a decent defence against FilterFool on lower filter weights. For higher filter weights, the defence mechanism has a harder time defending against the attack, achieving an accuracy of 68% on ImageNet, though this also causes larger perturbations in the adversarial images.
Code Implementations: At the time of writing, there are no known public code implementations of the attack.

5.2 RQ2: Semantic Adversarial Toolbox

This section describes the implementation details the different attacks and defences implemented in the toolbox. Links to the source code, as well as an installation guide can be found in Appendix A.

5.2.1 Featured Attacks

HSV Attack

The HSV Attack [24] is a color-based attack that randomly tweaks the hue and saturation of an image, hopefully generating an adversarial example. Though it might seem like a very simple attack strategy, its effectiveness was surprisingly high, and it can produce some good-looking adversarial images with a lucky perturbation. Many images however, ended up looking like the one in Figure 5.4, where the coloring looks unnatural. This is simply due to how the HSV color space is built (see section 2.2.1). The 'hue' channel of the HSV model contains a very large spectrum of colors, meaning even minor perturbations to this channel could largely affect the output image.

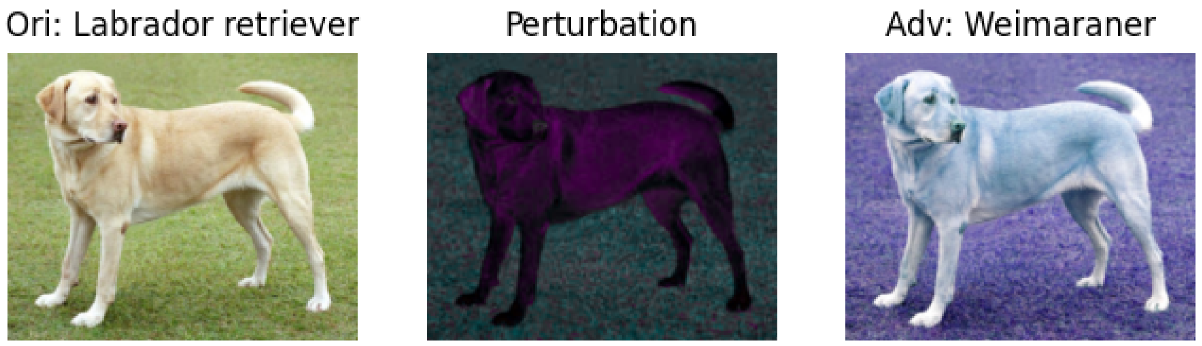

Figure 5.4: Demonstration of adversarial examples generated using HSV Attack.

Obviously, since the result is somewhat dependant on random values, there needs to be some mechanism handling situations in which perturbed images are not adversarial. This is solved by repeating the perturbation process until either the result is adversarial or a set number of attempts have been unsuccessful. Each iteration also increases the perturbation magnitude of the saturation value, which is weighted by the number of attempts. Algorithm 1 shows a detailed explanation of the attack as implemented in the toolbox. In some cases, perturbed values can be incorrect, such as hue having a value below 0 or above 1. These situations are handled either by clipping the excess values or using a modulo operation with 1 as the divisor. The modulo operation is useful for situations where more extreme values are unwanted, such as when perturbing saturation values.

Algorithm 1: HSV Attack

```
input  : Classifier $C$, Image $I$, Label $L$, Max iterations $iter_{max}$
output: Image that is adversarial to $C$
$I_{hsv} \leftarrow$ toHSV($I$);
$iter \leftarrow 1$;
$S \leftarrow \infty$;
repeat
    $weight \leftarrow iter \div iter_{max}$;
    $hue \leftarrow$ UniformValue(0, 1);
    $saturation \leftarrow$ UniformValue(-1, 1) $\cdot weight$
    $I_{hsv}[:,:,0] = I_{hsv}[:,:,0] + hue$;
    $I_{hsv}[:,:,1] = I_{hsv}[:,:,1] + saturation$;
    $I_{adv} \leftarrow$ toRGB($I_{hsv}$);
    $iter \leftarrow iter + 1$;
    if isAdversarial($I_{adv}$, $C$, $L$) and $hue + |saturation| < S$ then
        $X_{adv} \leftarrow I_{adv}$;
        $S \leftarrow hue + |saturation|$;
    end
until $iter \geq iter_{max}$;
return $X_{adv}$;
```

One of the main advantages of the HSV Attack is the low resource cost in generating images. Each iteration uses quick mathematical calculations to perturb the image. In addition to this, the attack can be easily modified to operate on batches rather than single images. During each iteration, we choose several uniform values at once and apply them to each image in the batch simultaneously. This drastically improves the running speed of the algorithm when generating multiple images at once.

The toolbox implementation of the attack took inspiration from the official code implementation provided by the author of the original paper [24]. Although the code was modified a bit to fit in with the standardized inputs and outputs of the toolbox, the fundamental attack strategy remained the same. Several optional hyperparameters such as whether to accept unsuccessful adversarials were added to improve attack customization and make the attack compatible with adversarial training. Additionally, a separate score function was implemented in order to incentivize smaller, more natural perturbations when needed.

Rotation and Translation Attack

The aptly named Rotation and Translation attack (or RT Attack) [38] uses a mixture of image rotations and translations to generate adversarial examples. Figure 5.5 shows an example output from the attack. Most of the generated images from this attack ended up quite good-looking. This is to be expected as the generated images are just slightly geometrically modified versions of the same input image. The inclusion of max rotation and max translation parameters prevents the generated images from having ridiculous transformations like being shifted all the way out of frame.

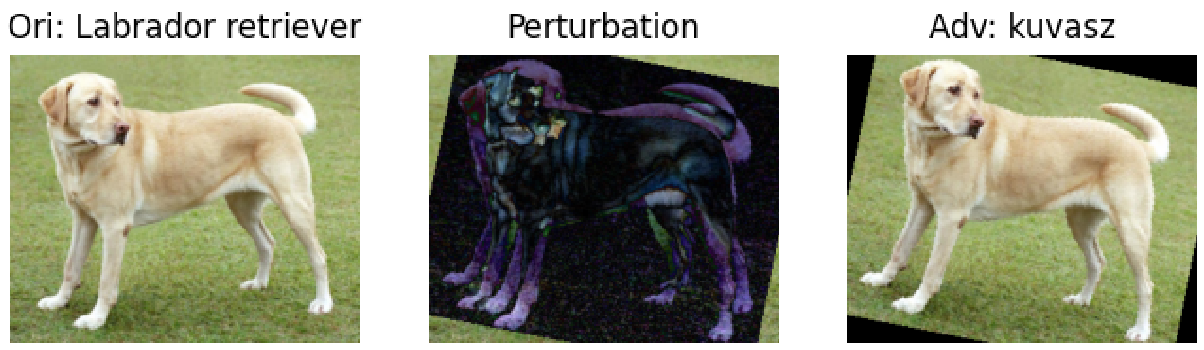

Figure 5.5: Adversarial examples using the Rotation and Translation Attack.

The original paper describes three different approaches to using affine transformations for adversarial example generation, two of which were implemented in the toolbox. The first one, grid search, parameterizes all possible transformations, which is computationally feasible because both rotations and translations have a discrete set of possible variations. Moreover, using this method guarantees a globally optimal solution, as all possible solutions are evaluated. The downside is that evaluating all parameters is a resource-heavy process.

Our implementation of the grid search algorithm is described below in algorithm 2. In order to find all possible parameter combinations, the algorithm calculates the cartesian product of all rotations and translations. Each geometric transformation is then applied sequentially. Since both the rotations and translations are defined as integers in this implementation, the cartesian product can be sorted by the absolute sum of its elements to ensure the output is globally optimal.

Algorithm 2: Rotation and Translation Attack with Grid Search

```
input  : Classifier $C$, Image $I$, Label $L$
output: Image that is adversarial to $C$
1  $rotations \leftarrow range(-180, 180)$;
2  $translations \leftarrow range(-100, 100)$;
3  $parameter\_list \leftarrow$ CartesianProduct ($rotations, translations, translations$) ;
4  $parameter\_list \leftarrow$ Sort ($parameter\_list$)
5  foreach $rotation, dx, dy$ in $parameter\_list$ do
6      $I_{adv} \leftarrow$ Affine ($I, rotation, dx, dy$) ;
7      if isAdversarial ($I_{adv}, C, L$) then
8          return $I_{adv}$;
9      end
10 end
```

The second approach, "worst of k", uses randomly sampled values for both rotation and translation, which is repeated a total of "k" times. The best performing adversarial from the "k" number of trials is then selected. While this does not guarantee a solution at the global minimum, the results are generally good. The greatest advantage of this approach is the low computational cost. Since only a handful of attempts are necessary, it vastly outperforms grid search in terms of speed.

The last approach, "first-order", was not implemented in the toolbox, as the original paper found it to perform poorly despite the number of trials allowed. Originally, this approach would use an adversarial loss function to gradually converge on an adversarial example. The introduction of white-box elements also made this approach more complex compared to both grid search and worst of k. For these reasons, the first-order method was considered unnecessary and was not implemented. Both the 'worst of k' and 'grid search' methods were developed from the ground up using attack descriptions from the original paper. Doing so allowed the attack to support image batches, increasing the attack speed.

EdgeFool

EdgeFool [58] finds and enhances edges of images, generating adversarial examples. Figure 5.6 shows an adversarial example generated from EdgeFool. The perturbations are intentionally made to look something like a stylistic filter, and the generated images are of high visual quality as a result.

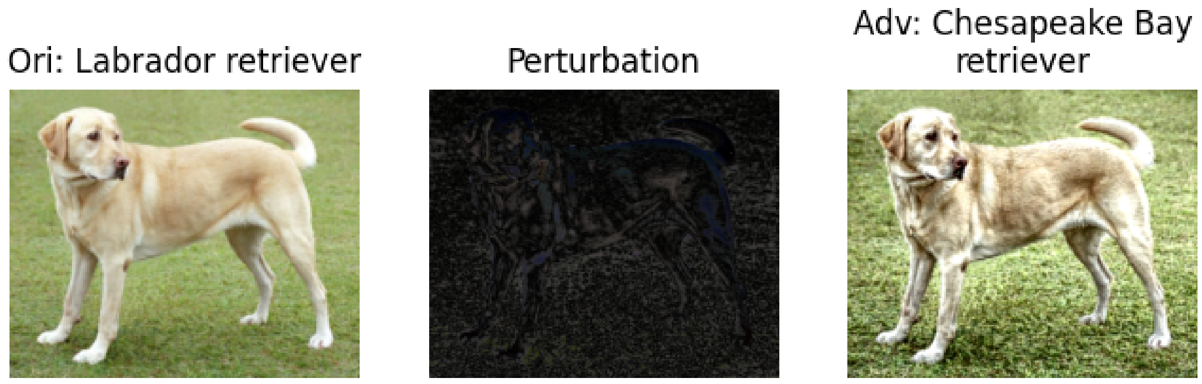

Figure 5.6: Example adversarial example generated from EdgeFool.

Adversarial generation with EdgeFool is a two-step process in which images are first smoothened and then enhanced. The image smoothening is done using L_0 gradient minimization [2], an image editing method for removing minor edges while maintaining larger ones. Figure 5.7 shows the effects of L_0 gradient minimization on an image. The major edges, such as the outline of the dog, are completely intact while minor edges such as individual hairs in the fur have been smoothened.

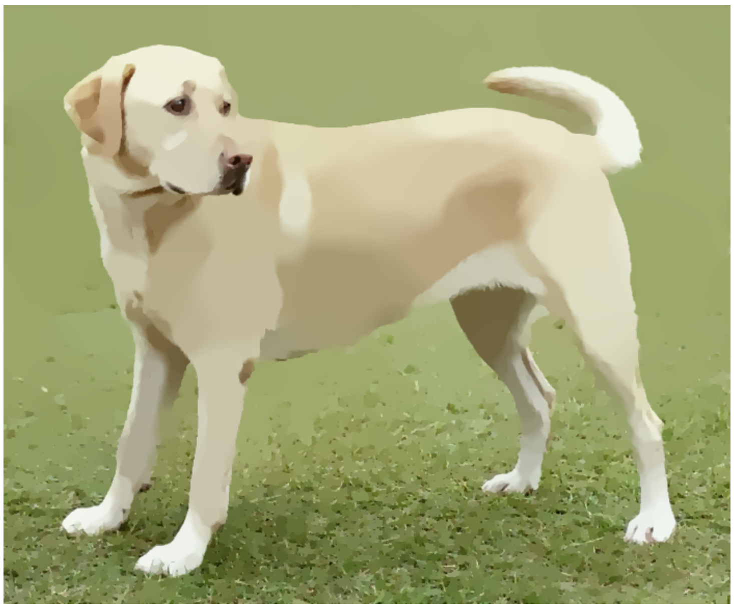

Figure 5.7: Example of smoothened image used by EdgeFool to enhance edges.

EdgeFool seeks to train a filter to decompose the original image into a structured component and a residual or detail component. The smoothened image is used as the ground truth for the structural component. In order to ensure both visual quality and misclassification, two different loss functions are combined while training the filter. The first one tracks the loss of the structural component, while the other tracks the adversarial loss. Using the learned structural component of the image, EdgeFool is able to isolate the details of the original image by comparing the differences. These details are then enhanced, resulting in a filter-like image effect. Algorithm

3 describes the toolbox implementation of EdgeFool, which is largely based on the official implementation created by the author of the original paper [58]. The 'FindResidual' function at line 6 separates the residual component from the structural component by calculating the difference between the original image and the smoothened image. This component is then enhanced mathematically and merged with the original structure component to form an enhanced image in line 7.

Algorithm 3: EdgeFool

```
input : Classifier $C$, Filter $F$, Image $I$, Label $L$, Max iteration $iter_{max}$, Target loss $loss_{target}$
output: Image that is adversarial to $C$
1  $I_{smooth} \leftarrow$ L0_minimization($I$);
2  while $iter \leq iter_{max}$ do
3      $X_{smooth} \leftarrow F(I, I_{smooth})$;
4      $X_{smooth} \leftarrow$ toLAB($X_{smooth}$);
5      $X_{adv} \leftarrow$ toLAB($I$);
6      $res \leftarrow$ FindResidual($X_{adv}, X_{smooth}$);
7      $X_{adv} \leftarrow$ EnhanceResidual($X_{adv}, res$);
8      $pred \leftarrow C(X_{adv})$;
9      $loss_{smooth} \leftarrow$ MSELoss($X_{smooth}, I_{smooth}$);
10     $loss_{adv} \leftarrow$ AdvLoss($pred, L$);
11     Backpropagate($F, loss_{smooth}, loss_{adv}$);
12     if isAdversarial($X_{adv}, C, L$) $and\ loss_{smooth} \leq loss_{target}$ then
13         return $X_{adv}$;
14     end
15     $iter += 1$;
16 end
```

While EdgeFool creates impressive perturbations, it is also very resource-heavy. Compared to attacks such as the HSV attack or RT attack, EdgeFool was found to be several factors slower. This is mostly due to the fact that each image had to be generated individually rather than in batches. In addition to this, the filter would in some cases have issues converging to the loss function, failing the attack as a result.

ColorFool

ColorFool [60] is another color-based adversarial attack, similar to that of the HSV attack (see section 5.2.1). In comparison to the HSV attack however, ColorFool is more sophisticated and will perturb different semantic regions of an image differently. This can be seen in Figure 5.8, where the color of the dog and the grass is differently perturbed. In ColorFool, perturbed images of grass will always be some variation of green to maintain realism. Likewise, other regions like water and sky will also be perturbed with natural colors. People however are not perturbed at all, as they are considered too complex.

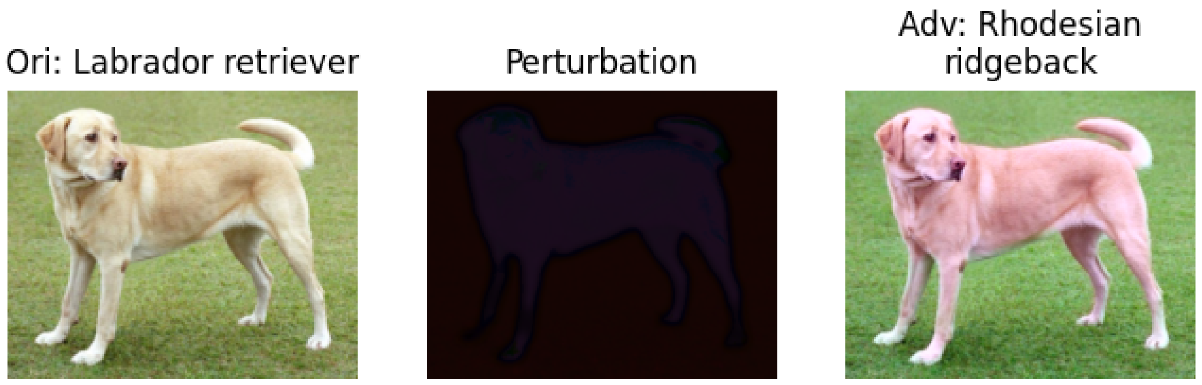

Figure 5.8: ColorFool attack adversarial example.

In order to perform different perturbations on different regions of an image, ColorFool uses semantic segmentation [21] to identify the image contents. The example in Figure 5.8 used the different image regions identified in Figure 5.9 to separate the dog from the grassy background.

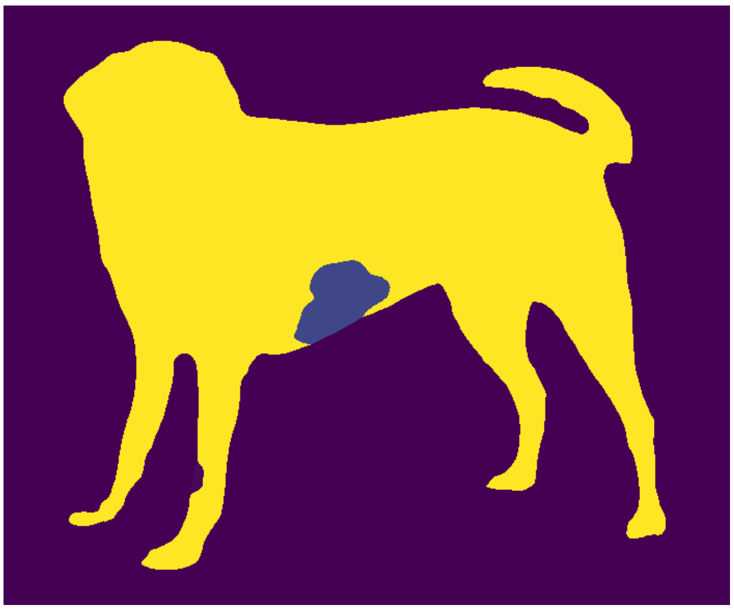

Figure 5.9: Example of semantic segmentation used to find which image regions should be perturbed.

Once the different semantic regions are identified, ColorFool iteratively perturbs the different regions in a similar manner as the HSV attack. The perturbation values are drawn randomly from a uniform distribution, scaled by the iteration number such that later iterations generate larger magnitudes of perturbation. These perturbations are made in the LAB color space (see section 2.2.1), which is specifically chosen due to its connection with human perception. Using LAB, the randomly chosen perturbation values can be drawn from a distribution of colors that are more natural to humans. As a result, adversarial images generated by ColorFool also look more natural. A more in-depth explanation of the attack can be seen in Algorithm 4. The ’PerturbRegion’ function on line 8 finds random color values for a given region mask and returns the perturbation for the *a* and *b* channels.

Algorithm 4: ColorFool

```
input  : Classifier $C$, Segmentation Module $S$, Image $I$, Label $L$, Max iterations $iter_{max}$
output: Image that is adversarial to $C$
masks ← Segment(I);
$I_{lab}$ ← toLAB(I);
iter ← 0
while $iter \le iter_{max}$ do
    noise_mask ← array of zeros of shape $I_{lab}$;
    weight ← $(iter + 1) \div iter_{max}$;
    foreach mask in masks do
        A, B ← PerturbRegion(mask, $I_{lab}$, weight);
        noise_mask[;,;,1] ← noise_mask[;,;,1] + A;
        noise_mask[;,;,2] ← noise_mask[;,;,2] + B;
    end
    $I_{adv}$ ← $I_{lab}$;
    $I_{adv}$[;,;,1] ← $I_{adv}$[;,;,1] + noise_mask[;,;,1];
    $I_{adv}$[;,;,2] ← $I_{adv}$[;,;,2] + noise_mask[;,;,2];
    $I_{adv}$ ← toRGB($I_{adv}$);
    if isAdversarial($I_{adv}$, C, L) then
        return $I_{adv}$;
    end
    iter+ = 1;
end
```

Much like EdgeFool, ColorFool is an adversarial attack that perturbs images one by one. This means large tasks like generating data for adversarial training are very slow. Additionally, the attack poses a couple of restrictions on the allowed perturbations such as grass always being some variation of green, etc. These restrictions, while benefiting the overall adversarial quality, also make finding an adversarial example more difficult. Since each iteration of ColorFool increases the perturbation magnitude, the added restrictions can actually have an adverse effect on quality, though this rarely seemed to be an issue in testing.

5.2.2 Featured Defences

Adversarial Training

Because adversarial training is such a complex subject, it was important that its implementation in the toolbox was versatile and left room for creativity. As a result, the entire adversarial training process was divided into three separate steps, which were defined as follows:

1. Generate list of adversarial examples from a given dataset.
2. Create adversarial dataset comprising of both adversarial and non-adversarial images.
3. Train classifier on adversarial dataset.

Step one involved generating the images themselves. The general procedure for this was rather simple. Given a dataset of images D and a target number T, the function would return a list of adversarial images of length T. To increase versatility, multiple attacks were supported simultaneously. In such cases, each attack would be run sequentially, generating the same amount of adversarial images before being merged with the outputs of other attacks.

The reason why step one was made separate from step two was to allow for faster experimentation of hyperparameters. Generating a large number of adversarial images can be time-consuming. By keeping this process separate from other parts, it would be possible to try out several different configurations in later steps without having to generate an entirely new set of images.

Step two involved converting the list of adversarial images into a complete dataset ready for training. Although this was not a very difficult process by itself, the number of customizable options made it somewhat tedious.

The final step was the training itself. Training any classifier in PyTorch usually requires a specifically tailored training scheme for best results. It would therefore in most cases be best for users to implement the training procedure by themselves. This was also a big reason why the training step was kept separate from the previous two steps. Nevertheless, a generic training algorithm was still implemented for anyone who would want to use it.

JPEG Compression

The idea behind JPEG Compression is rather simple (see section 2.4.3). In simple terms, an image is compressed using the compression algorithm of JPEG. The quickest implementation of this was simply storing the image as a JPEG file, as modern computers are fully capable of doing the conversion automatically. Afterwards, by loading the converted image back into a python variable, the compression would already be completed. The entire implementation can be seen in Algorithm 5. Pillow [6], a python image library with great compatibility with PyTorch, supports storing image objects to memory instead of disk, which made the entire process a lot faster. The lack of having to write individual files to disk was one of the main contributing factors to making this implementation viable.

Algorithm 5: JPEG Compression

```
input  : Image $I$, Compression Quality Measure $Q$
output: Compressed Image
1 $M \leftarrow$ memory object interface;
2 SaveJPEG($I, Q, M$);
3 $I_{jpeg} \leftarrow$ LoadJPEG($M$);
4 return $I_{jpeg}$;
```

Feature Squeezing

The implementation of Feature Squeezing (see section 2.4.2) was divided into two main parts: bit reduction and median smoothing. Bit reduction was implemented using simple math, as seen in Algorithm 6. First, the maximum number of different colors encodable by the reduced bit depth is calculated. Afterwards, the existing color values are converted to the closest values in the new range. The median smoothing was slightly more complex to implement. To apply median filtering, a sliding window operation was used to extract each window from the image. A median operation was then applied to each of these windows in order to find the median values. The image was padded in order to keep the dimension size from deteriorating due to the sliding window.

```
Algorithm 6: Feature Squeezing
input  : Image $I$, Bit Depth $B$, Filter Size $F$
output: Compressed Image
1 $B_{value} \leftarrow 2^B - 1$;
2 $I_{fs} \leftarrow \lfloor I \cdot B_{value} \rfloor \div B_{value}$;
3 $I_{fs} \leftarrow$ padded image $I_{fs}$;
4 $L_w \leftarrow$ sliding windows for size $F$ on image $I_{fs}$;
5 $I_{fs} \leftarrow$ median operations on all windows in $L_w$;
6 return $I_{fs}$;
```

5.3 RQ3: Integrated Adversarial training

This section covers the results of the adversarial training experiments described in section 4.3. All results in this section show model accuracy (higher is better) for different adversarial attacks. The attack parameters used were the same for all the attacks, and a description of these can be found in section 4.3.2. For accuracy results using different attack parameters like random perturbations, see Appendix C.

5.3.1 Cifar10 Results

Table 5.5 shows the evaluation results of the different classifiers trained on CIFAR10. The first row shows the results of standard training, whereas the following rows show the results of adversarial training. The last row shows the results of the classifier trained on multiple attacks. The columns refer to the different testsets described in section 4.3.2. Colored cells highlight the most important results.

	Standard	HSV	RT	EdgeFool
Standard Trained	94.81%	47.04%	51.92%	67.26%
HSV AdvTrained	94.10%	91.31%	45.32%	61.18%
RT AdvTrained	**95.09%**	50.68%	**91.42%**	76.62%
EdgeFool AdvTrained	**95.22%**	46.56%	73.00%	**93.19%**
Integration AdvTrained	**95.41%**	**92.57%**	**91.67%**	92.06%

Table 5.5: Accuracy of classifiers trained on various CIFAR10 adversarial datasets.

There are a few noteworthy discoveries in these results. The accuracy of the integrated adversarial model suggests that the three attacks in question are indeed very compatible, and that robustness to one of them does not necessarily imply vulnerability to others. For the HSV and RT attacks, the integrated model seemingly performs even better than the HSV and RT robust models. Against EdgeFool however, the integrated model seems to perform slightly worse than the single-attack-trained model. The difference is so small however that it could simply be caused by variance in the model training,

One interesting finding is the apparent connection between the RT attack and EdgeFool. The robustness of the EdgeFool-trained classifier against RT attacks, with an accuracy of 73%, is surprising to see. Likewise, the RT-trained classifier seems more robust to perturbations generated by EdgeFool, attaining 76% accuracy. This might suggest some overlap between the two, however it is hard to imagine why looking at the perturbations generated by both attacks.

Looking at the data, it is also apparent that the adversarial training did not negatively impact the standard accuracy of the models, with several of the robust models actually performing better than the standard one. Early research into adversarial examples argued that adversarial training could improve standard accuracy by providing additional regularization [3, 7]. This could also help explain why the integrated model performed better against the HSV and RT attacks than their single-attack trained counterparts.

5.3.2 Cifar100 Results

Table 5.6 shows the accuracy results for models trained on CIFAR100. Although the standard accuracy on this dataset is lower, the results in many ways mimic the ones from CIFAR10. Again, the robust training against several unrestricted adversarial attacks does not seem to negatively impact the robustness against each individual attack or the standard accuracy. One difference however is the EdgeFool results, which on CIFAR10 were slightly worse for the integration model than the EdgeFool trained model.

Very interestingly, the integrated model performed the best against every attack in the experiments, which again likely comes down to increased regularization caused by the additional perturbed samples in the training set.

	Standard	HSV	RT	EdgeFool
Standard Trained	**74.95%**	34.67%	48.52%	43.14%
HSV AdvTrained	74.85%	**72.97%**	47.67%	40.26%
RT AdvTrained	74.41%	37.87%	61.05%	44.95%
EdgeFool AdvTrained	74.57%	35.84%	51.23%	**70.75%**
Integration AdvTrained	**75.34%**	**73.23%**	**63.43%**	**71.31%**

Table 5.6: Accuracy of classifiers trained on various CIFAR100 adversarial datasets.

Chapter 6

Discussion

6.1 RQ1: Literature Review

To date, there are no published literature reviews focusing on the topic of semantic adversarial attacks. This makes our literature review the first of its kind, and may help further research on the topic. New papers focusing on semantic adversarial attacks are published regularly, so there may be new attacks in the not-so-distant future that are not included in our review. Keeping up to date with new discoveries can be difficult, and literature reviews serve an important role in making it easier.

6.2 RQ2: Semantic Adversarial Toolbox

Although many different adversarial toolboxes are in existence already, the Semantic Adversarial Toolbox serves a specific subgroup of attacks not yet covered by others. This differentiates the toolbox from the rest and will allow both academic and industrial settings to gain insight into unrestricted adversarial attacks. The semantic toolbox at its current state however may lack different features available in other toolboxes due to time constraints. Future work for the toolbox will be discussed more deeply in section 7.1, but in short, there should be more attacks, more defences, and more robustness metrics. Currently, there are no universally agreed upon robustness metric for unrestricted attacks, and while other adversarial defence methods such as label smoothing does exist, their effectiveness against semantic attacks are not well documented.

6.3 RQ3: Integrated Adversarial Training

Tramèr et al. [49] perform similar experiments to ours, combining both ℓ_∞ and RT attacks in one training scheme, among other contributions. Their results indicate that these two perturbation types are incompatible and that achieving optimal robustness towards both of them simultaneously is not possible. Although the paper did not make any claims on similar experiments for multiple unrestricted perturbations, the contrary result of doing so is very promising.

On the topic of standard versus robust accuracy, it may seem that noticeably large perturbations do not impact standard accuracy the same way Tsipras et al. [50] describes for ℓ_p bounded perturbations. In fact, it seems that these unrestricted adversarial samples help the models generalize better than standard training, allowing for better testing results, though not by a lot. Schmidt et al. [30] show that robust generalization is a more complex task than standard generalization, and argues that more data might be required for proper robust generalization. Our multi-perturbation robust models are trained on datasets of 125 thousand images, of which 75 thousand are adversarial samples generated from the standard CIFAR datasets. Although it can be argued that the additional adversarial samples are simply variations of the same data rather than more data, we argue that for sufficiently large perturbations, such as the ones utilized in this research, the effects are practically the same. Consider for instance a data sample perturbed using the HSV attack described in section 5.2.1. The large uniform difference in color allows a classifier to generalize to color-based adversarial attacks the same way a perturbed entirely new data sample would. The original data sample still helps the classifier generalize to standard image features without impacting the robust generalization. Although it is reasonable to suspect that this could lead to overfitting, Figure 6.1 shows that such a scenario did not occur.

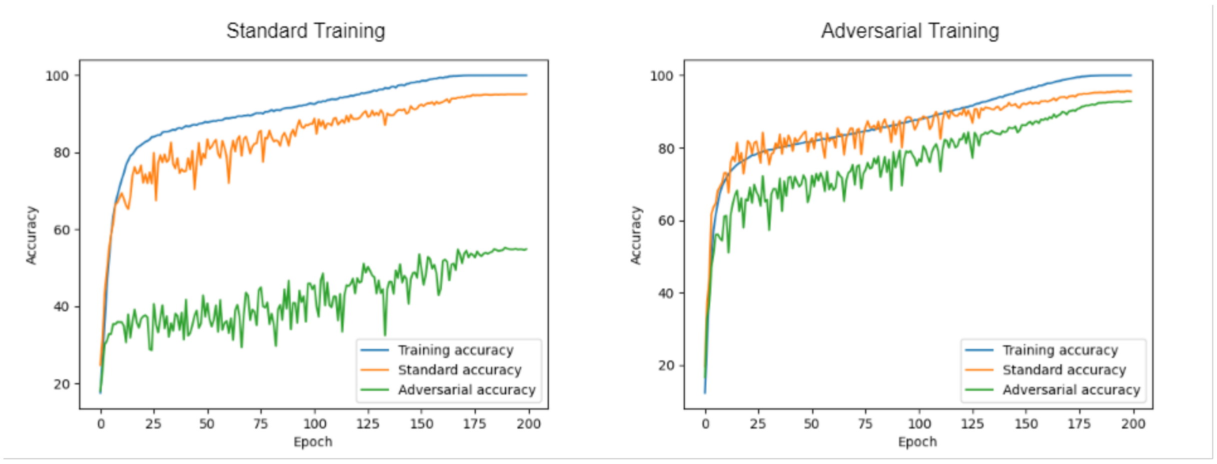

Figure 6.1: Generalization of standard training vs. integrated adversarial training. The left plot visualizes training, testing and adversarial accuracy for standard training using the parameters defined in section 4.3.4. The right plot visualizes the same data for multi-perturbation adversarial training, using the same parameters.

For industry purposes, integrated adversarial training might be very useful for boosting model robustness. A very common concern for most industrial settings is the drawbacks that often come with adversarial training in the form of reduced standard accuracy. The main drawback of our adversarial training scheme is the additional computer resources required to train the classifier using extra training data. Generating the extra adversarial samples can also be a resource-heavy process, though both the HSV and RT attacks are fairly lightweight algorithms.

Although only three different attacks were evaluated in this research, it is not unthinkable that the robustness acquired against these perturbations may transfer over to others that were not part of the experiments. The adversarial training against HSV perturbations may for instance also have inadvertently increased robustness to other color-based adversarial attacks. On the other hand, since rotations and translations are known to be incompatible with ℓ_∞ bounded perturbations [49], it is also worth testing the vulnerabilities of the multi-perturbation robust model to other, more common attacks. Finally, the possibilities of training against even more than three unrestricted attacks simultaneously should be explored.

Chapter 7

Conclusion

In this thesis, we have provided a few different contributions to the field of adversarial attacks. A systematic literature review has been conducted, mapping the current landscape of semantic adversarial attacks and allowing for deeper understanding of its current state. In addition, a toolbox has been developed for generating and defending against semantic adversarial attacks. Lastly, it has been shown that adversarial training can defend against multiple unrestricted adversarial attacks with relatively few drawbacks.

What is the current landscape of semantic adversarial examples? The current state-of-the-art semantic adversarial attacks are divided into categories of color modification, geometric transformation, and feature manipulation. Most of the semantic attacks were located in the feature manipulation category, where perturbations were applied to some feature representation. In total, 19 different semantic attacks were identified, 10 of which utilized feature manipulation, 5 of which featured color modification, and 4 of which applied some geometric transformation.

Can we make a specialized toolbox for generating and defending against semantic adversarial attacks? The Semantic Adversarial Toolbox was developed for this purpose and currently supports four different unrestricted adversarial attacks. Additionally, the implemented defences allow users to improve model robustness to these attacks. All in all the toolbox performs well and could be of help to future research on semantic adversarial attacks.

Is it possible to effectively defend against multiple semantic adversarial attacks using adversarial training, and what are the drawbacks to using this approach? It turned out that multi-perturbation adversarial training was more effective on unrestricted attacks than on restricted ones. Compared to similar previous research using restricted attacks, our results showed better robustness to each individual attack as well as better standard accuracy.

The goal of this thesis was to improve model robustness to semantic adversarial attacks using a customized toolbox. With the final results of both the toolbox and the integrated adversarial training in mind, we must argue that this goal has been achieved to some degree. Of course, adversarial robustness can always be improved further, and our results can not be considered more than a step in the right direction. In the future, hopefully model robustness will be good enough to not be as much of a concern as it is today.

7.1 Future Work

Over the course of this thesis, many interesting paths were left unexplored due to time constraints. These topics are covered in this section, along with some potential improvements to the existing work.

7.1.1 Improvements to the Adversarial Toolbox

The main weakness of the Semantic Adversarial Toolbox currently is the number of supported attacks. Given more time, several more attacks could be implemented, further increasing the usefulness of the toolbox. Additionally, the toolbox can be expanded to support more deep learning frameworks, such as TensorFlow [8] and Keras [5]. For the purposes of this thesis, only PyTorch [45] was supported due to time constraints.

7.1.2 More Extensive Testing With Integrated Adversarial Training

Given more time to evaluate the integrated adversarial training, there are a few experiments worth testing. Firstly, the adversarial training experiments should be extended to the ImageNet dataset as well in order to determine its effectiveness in a more realistic real-world setting. With the time constraints present in this thesis, it was not possible to generate enough adversarial samples to properly apply adversarial training to ImageNet. It is however our opinion that doing so should yield comparable results to both CIFAR10 and CIFAR100.

Furthermore, since all robust models trained in this thesis used a validation set comprising of 10% of the training set, it begs the question of how the results would change when keeping the whole training set intact. It is our belief that the added training samples would allow for even better generalization and model accuracy. It is a shame that this idea did not spring to mind during the model training phase of this project, however it should be fairly simple to rectify given more time.

References

[1] "HSV Color Space". In: *Encyclopedia of Microfluidics and Nanofluidics*. Ed. by Dongqing Li. Boston, MA: Springer US, 2008, pp. 793–793. ISBN: 978-0-387-48998-8. DOI: 10.1007/978-0-387-48998-8_656. URL: https://doi.org/10.1007/978-0-387-48998-8_656.

[2] Li Xu et al. "Image Smoothing via L0 Gradient Minimization". In: *ACM Transactions on Graphics (SIGGRAPH Asia)* (2011).

[3] Mohamad Ali Torkamani and Daniel Lowd. "Convex Adversarial Collective Classification". In: June 2013.

[4] Ming Ronnier Luo. "CIELAB". In: *Encyclopedia of Color Science and Technology*. Ed. by Ronnier Luo. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 1–7. ISBN: 978-3-642-27851-8. DOI: 10.1007/978-3-642-27851-8_11-1. URL: https://doi.org/10.1007/978-3-642-27851-8_11-1.

[5] François Chollet. *keras*. https://github.com/fchollet/keras. 2015.

[6] Alex Clark. *Pillow (PIL Fork) Documentation*. 2015. URL: https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf.

[7] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and Harnessing Adversarial Examples". In: *International Conference on Learning Representations*. 2015. URL: http://arxiv.org/abs/1412.6572.

[8] Martın Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[9] Uri Shaham, Yutaro Yamada, and Sahand Negahban. "Understanding Adversarial Training: Increasing Local Stability of Neural Nets through Robust Optimization". In: *Neurocomputing* 307 (Nov. 2015). DOI: 10.1016/j.neucom.2018.04.027.

[10] Gintare Dziugaite, Zoubin Ghahramani, and Daniel Roy. "A study of the effect of JPG compression on adversarial images". In: (Aug. 2016).

[11] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[12] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. "Adversarial examples in the physical world." In: *CoRR* abs/1607.02533 (2016). URL: http://dblp.uni-trier.de/db/journals/corr/corr1607.html#KurakinGB16.

[13] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. "DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks". In: June 2016, pp. 2574–2582. DOI: 10.1109/CVPR.2016.282.

[14] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. *Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples*. 2016. arXiv: 1605.07277 [cs.CR].

[15] Nicholas Carlini and David Wagner. "Towards Evaluating the Robustness of Neural Networks". In: May 2017, pp. 39–57. DOI: 10.1109/SP.2017.49.

[16] J. Kolter and Eric Wong. "Provable defenses against adversarial examples via the convex outer adversarial polytope". In: (Nov. 2017).

[17] Guillaume Lample et al. "Fader Networks:Manipulating Images by Sliding Attributes". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/3fd60983292458bf7dee75f12d5e9e05-Paper.pdf.

[18] Jonas Rauber, Wieland Brendel, and Matthias Bethge. "Foolbox: A Python toolbox to benchmark the robustness of machine learning models". In: *Reliable Machine Learning in the Wild Workshop, 34th International Conference on Machine Learning*. 2017. URL: http://arxiv.org/abs/1707.04131.

[19] Chen Sun et al. "Revisiting Unreasonable Effectiveness of Data in Deep Learning Era". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. Oct. 2017.

[20] Richard Zhang et al. "Real-Time User-Guided Image Colorization with Learned Deep Priors". In: *ACM Transactions on Graphics (TOG)* 9.4 (2017).

[21] Bolei Zhou et al. "Scene Parsing through ADE20K Dataset". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 5122–5130. DOI: 10.1109/CVPR.2017.544.

[22] K. Eykholt et al. "Robust Physical-World Attacks on Deep Learning Visual Classification". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 1625–1634.

[23] Chuan Guo et al. "Countering Adversarial Images using Input Transformations". In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=SyJ7ClWCb.

[24] H. Hosseini and R. Poovendran. "Semantic Adversarial Examples". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2018), pp. 1695–16955.

[25] Can Kanbak, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. "Geometric Robustness of Deep Networks: Analysis and Improvement". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.

[26] Aleksander Madry et al. "Towards Deep Learning Models Resistant to Adversarial Attacks". In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=rJzIBfZAb.

[27] Maria-Irina Nicolae et al. "Adversarial Robustness Toolbox v1.2.0". In: *CoRR* 1807.01069 (2018). URL: https://arxiv.org/pdf/1807.01069.

[28] Nicolas Papernot and Patrick McDaniel. "Deep k-Nearest Neighbors: Towards Confident, Interpretable and Robust Deep Learning". In: (Mar. 2018).

[29] Nicolas Papernot et al. "Technical Report on the CleverHans v2.1.0 Adversarial Examples Library". In: *arXiv preprint arXiv:1610.00768* (2018).

[30] Ludwig Schmidt et al. *Adversarially Robust Generalization Requires More Data*. Apr. 2018.

[31] Aman Sinha, Hongseok Namkoong, and John Duchi. “Certifiable Distributional Robustness with Principled Adversarial Training”. In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=Hk6kPgZA-.

[32] Chaowei Xiao et al. “Spatially Transformed Adversarial Examples”. In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=HyydRMZC-.

[33] Weilin Xu, David Evans, and Yanjun Qi. “Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks”. In: Jan. 2018. DOI: 10.14722/ndss.2018.23210.

[34] Zhengli Zhao, Dheeru Dua, and Sameer Singh. “Generating Natural Adversarial Examples”. In: *International Conference on Learning Representations (ICLR)*. 2018.

[35] Rima Alaifari, Giovanni S. Alberti, and Tandri Gauksson. “ADef: an Iterative Algorithm to Construct Adversarial Deformations”. In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=Hk4dFjR5K7.

[36] Ambra Demontis et al. “Why Do Adversarial Attacks Transfer? Explaining Transferability of Evasion and Poisoning Attacks”. In: *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 321–338. ISBN: 978-1-939133-06-9. URL: https://www.usenix.org/conference/usenixsecurity19/presentation/demontis.

[37] Gavin Weiguang Ding, Luyu Wang, and Xiaomeng Jin. “AdverTorch v0.1: An Adversarial Robustness Toolbox based on PyTorch”. In: *arXiv preprint arXiv:1902.07623* (2019).

[38] Logan Engstrom et al. *A Rotation and a Translation Suffice: Fooling CNNs with Simple Transformations*. 2019. URL: https://openreview.net/forum?id=BJfvknCqFQ.

[39] Robert Geirhos et al. “ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness.” In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=Bygh9j09KX.

[40] Zhenliang He et al. “AttGAN: Facial Attribute Editing by Only Changing What You Want”. In: *IEEE Transactions on Image Processing* 28.11 (2019), pp. 5464–5478. DOI: 10.1109/TIP.2019.2916751.

[41] Nathan Inkawhich et al. “Feature Space Perturbations Yield More Transferable Adversarial Examples”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[42] Ameya Joshi et al. “Semantic Adversarial Attacks: Parametric Transformations That Fool Deep Classifiers”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2019.

[43] Cassidy Laidlaw and Soheil Feizi. “Functional Adversarial Attacks”. In: *NeurIPS*. 2019.

[44] Xiang Ling et al. “DEEPSEC: A Uniform Platform for Security Analysis of Deep Learning Model”. In: May 2019, pp. 673–690. DOI: 10.1109/SP.2019.00023.

[45] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[46] Kevin Roth, Yannic Kilcher, and Thomas Hofmann. “The Odds are Odd: A Statistical Test for Detecting Adversarial Examples”. In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, Sept. 2019, pp. 5498–5507. URL: `http://proceedings.mlr.press/v97/roth19a.html`.

[47] Lukas Schott et al. “Towards the first adversarially robust neural network model on MNIST”. In: *International Conference on Learning Representations*. 2019. URL: `https://openreview.net/forum?id=S1EHOsC9tX`.

[48] J. Su, D. V. Vargas, and K. Sakurai. “One Pixel Attack for Fooling Deep Neural Networks”. In: *IEEE Transactions on Evolutionary Computation* 23.5 (2019), pp. 828–841.

[49] Florian Tramer and Dan Boneh. “Adversarial Training and Robustness for Multiple Perturbations”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: `https://proceedings.neurips.cc/paper/2019/file/5d4ae76f053f8f2516ad12961ef7fe97-Paper.pdf`.

[50] Dimitris Tsipras et al. “Robustness May Be at Odds with Accuracy”. In: *International Conference on Learning Representations*. 2019. URL: `https://openreview.net/forum?id=SyxAb30cY7`.

[51] Cihang Xie et al. “Feature Denoising for Improving Adversarial Robustness”. In: June 2019, pp. 501–509. DOI: `10.1109/CVPR.2019.00059`.

[52] Hongyang Zhang et al. “Theoretically Principled Trade-off between Robustness and Accuracy”. In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, Sept. 2019, pp. 7472–7482. URL: `http://proceedings.mlr.press/v97/zhang19p.html`.

[53] Anand Bhattad et al. “Unrestricted Adversarial Examples via Semantic Manipulation”. In: *International Conference on Learning Representations*. 2020. URL: `https://openreview.net/forum?id=Sye_OgHFwH`.

[54] Tejas Gokhale et al. *Attribute-Guided Adversarial Training for Robustness to Natural Perturbations*. 2020. arXiv: `2012.01806 [cs.CV]`.

[55] Dou Goodman et al. *Advbox: a toolbox to generate adversarial examples that fool neural networks*. 2020. arXiv: `2001.05574 [cs.LG]`.

[56] Lakshya Jain et al. *Analyzing and Improving Neural Networks by Generating Semantic Counterexamples through Differentiable Rendering*. 2020. arXiv: `1910.00727 [cs.LG]`.

[57] Haonan Qiu et al. “Semanticadv: Generating adversarial examples via attribute-conditioned image editing”. In: *ECCV*. 2020.

[58] A. S. Shamsabadi, C. Oh, and A. Cavallaro. “Edgefool: an Adversarial Image Enhancement Filter”. In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2020, pp. 1898–1902. DOI: `10.1109/ICASSP40776.2020.9054368`.

[59] Ali Shahin Shamsabadi, Changjae Oh, and Andrea Cavallaro. *Semantically Adversarial Learnable Filters*. 2020. arXiv: `2008.06069 [cs.CV]`.

[60] Ali Shahin Shamsabadi, Ricardo Sanchez-Matilla, and Andrea Cavallaro. “ColorFool: Semantic Adversarial Colorization”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

[61] Shuo Wang et al. *Generating Semantic Adversarial Examples via Feature Manipulation*. 2020. arXiv: `2001.02297 [cs.LG]`.

[62] Pengfei Xia and Bin Li. *Improving Resistance to Adversarial Deformations by Regularizing Gradients*. 2020. arXiv: `2008.12997 [cs.LG]`.

[63] Qiuling Xu et al. *Towards Feature Space Adversarial Attack*. 2020. arXiv: `2004.12385 [cs.LG]`.

[64] Zhengyu Zhao, Zhuoran Liu, and Martha Larson. *Adversarial Color Enhancement: Generating Unrestricted Adversarial Images by Optimizing a Color Filter*. 2020. arXiv: `2002.01008 [cs.CV]`.

Appendices

Appendix A

Installation Guide

GitHub:	https://github.com/larsksy/Semantic-Adversarial-Toolbox
Pretrained Models:	Google Drive Link

TOOLBOX SETUP

THE FOLLOWING STEPS ARE REQUIRED TO SETUP THE TOOLBOX:

1. Download or clone GitHub repo. Link to repo is provided above.
2. Install required Python packages specified in the 'requirements.txt' file.
3. Check out the README or 'examples' folder for basic usage instructions.

TEST ADVERSARIAL MODELS

1. Download pretrained models from link provided above. The model files are too huge for GitHub and must be downloaded separately.
2. Place all files in the archive in a folder called 'pretrained' at the root level of the repository.
3. Run premade testing scripts in 'examples' folder. Make sure the root of the repository is set as the working directory, otherwise the model files will not be found.

Appendix B

Literature Review Results

Title	Author	Year	Found by	Included
Semantic Adversarial Deep Learning	Dreossi et al.	2018	ORIA: SEMANTIC ADVERSARIAL	No, does not propose attack method
Semantic Adversarial Examples	Hosseini et al.	2018	ORIA: SEMANTIC ADVERSARIAL	Yes
ColorFool: Semantic Adversarial Colorization	Shamsabadi et al.	2020	ORIA: SEMANTIC ADVERSARIAL	Yes
Generating Semantic Adversarial Examples via Feature Manipulation	Wang et al.	2020	ORIA: SEMANTIC ADVERSARIAL	Yes
SADA: Semantic Adversarial Diagnostic Attacks for Autonomous Applications	Hamdi et al.	2019	ORIA: SEMANTIC ADVERSARIAL	No, more focus on XAI than adversarial attacks.
BREAKING CERTIFIED DEFENSES: SEMANTIC ADVERSARIAL EXAMPLES WITH SPOOFED ROBUSTNESS CERTIFICATES	Ghiasi et al.	2020	ORIA: SEMANTIC ADVERSARIAL	No, uses an updated PGD attack method rather than semantical perturbations.
Semantic Adversarial Attacks: Parametric Transformations That Fool Deep Classifiers	Joshi et al.	2019	ORIA: SEMANTIC ADVERSARIAL	Yes
Semantics Preserving Adversarial Attacks	Dia et al.	2019	ORIA: SEMANTIC ADVERSARIAL	No, despite name, it does not use semantic adversarial attack method
Model Robustness with Text Classification: Semantic-preserving adversarial attacks	Singh et al.	2020	ORIA: SEMANTIC ADVERSARIAL	No, made for text classification
Defending Adversarial Attacks via Semantic Feature Manipulation	Wang et al.	2020	ORIA: SEMANTIC ADVERSARIAL	No, does not propose attack method.
UNRESTRICTED ADVERSARIAL EXAMPLES VIA SEMANTIC MANIPULATION	Bhattad et al.	2020	ORIA: SEMANTIC ADVERSARIAL	Yes
TOWARDS FEATURE SPACE ADVERSARIAL ATTACK	Xu et al.	2020	ORIA: FEATURE ADVERSARIAL ATTACK	Yes
D2B: Deep Distribution Bound for Natural-looking Adversarial Attack	Xu et al.	2020	ORIA: FEATURE ADVERSARIAL ATTACK	No, not semantic based attack
Towards Transferable Adversarial Attack against Deep Face Recognition	Zhong et al.	2020	ORIA: FEATURE ADVERSARIAL ATTACK	No, uses pixel-space attacks
GENERATING NATURAL ADVERSARIAL EXAMPLES	Zhao et al.	2017	ORIA: NATURAL ADVERSARIAL	Yes
Natural Adversarial Examples	Hendrycks et al.	2019	ORIA: NATURAL ADVERSARIAL	No, provides adversarial dataset, no attack method.

Mimic and Fool: A Task-Agnostic Adversarial Attack	Chaturvedi et al.	2020	ORIA: NATURAL ADVERSARIAL	No, does not provide semantically meaningful adversarial examples
Analyzing and Improving Neural Networks by Generating Semantic Counterexamples through Differentiable Rendering	Chandrasekaran et al.	2020	GOOGLE SCHOLAR: SEMANTIC ADVERSARIAL	Yes
SemanticAdv: Generating Adversarial Examples via Attribute-conditioned Image Editing	Qui et al.	2020	GOOGLE SCHOLAR: SEMANTIC ADVERSARIAL	Yes
Evaluating Robustness to Context-Sensitive Feature Perturbations of Different Granularities	Dunn et al.	2020	GOOGLE SCHOLAR: SEMANTIC ADVERSARIAL	Yes
Constructing Unrestricted Adversarial Examples with Generative Models	Song et al.	2018	GOOGLE SCHOLAR: NATURAL ADVERSARIAL	No, not semantic based attack
Feature Space Perturbations Yield More Transferable Adversarial Examples	Inkawhich et al.	2019	GOOGLE SCHOLAR: FEATURE ADVERSARIAL ATTACK	Yes
Feature Denoising for Improving Adversarial Robustness	Xie et al.	2019	GOOGLE SCHOLAR: FEATURE ADVERSARIAL ATTACK	No, does not propose attacks method
SPATIALLY TRANSFORMED ADVERSARIAL EXAMPLES	Xiao et al.	2018	SNOWBALLING (BACKWARDS)	Yes
Exploring the Landscape of Spatial Robustness	Engstrom et al.	2019	SNOWBALLING (BACKWARDS)	Yes
Towards Large yet Imperceptible Adversarial Image Perturbations with Perceptual Color Distance	Zhao et al.	2020	SNOWBALLING (FORWARDS)	No, does not perturb semantically
Adversarial Color Enhancement: Generating Unrestricted Adversarial Images by Optimizing a Color Filter	Zhao et al.	2020	SNOWBALLING (FORWARDS)	Yes
Functional Adversarial Attacks	Laidlaw et al.	2019	SNOWBALLING (BACKWARDS)	Yes
ShapeAdv: Generating Shape-Aware Adversarial 3D Point Clouds	Lee et al.	2020	SNOWBALLING (FORWARDS)	No, not implemented for image classification
EDGEFOOL: AN ADVERSARIAL IMAGE ENHANCEMENT FILTER	Shamsabadi et al.	2020	SNOWBALLING (BACKWARDS)	Yes
Semantically Adversarial Learnable Filters	Shamsabadi et al.	2020	SNOWBALLING (FORWARDS)	Yes
ADef: an iterative algorithm to construct adversarial deformations	Alaifari et al.	2018	SNOWBALLING (FORWARDS)	Yes

Geometric robustness of deep networks: analysis and improvement	Kanbak et al.	2018	SNOWBALLING (FORWARDS)	Yes

Appendix C

Extra Adversarial Training Results

The results listed in this section are complimentary to the adversarial training results described in section 5.3. Table C.1 shows model accuracy for random perturbations. EdgeFool was omitted from these results as it is a whitebox attack that inherently does not generate random perturbations, unlike the HSV and RT attacks.

	CIFAR10		CIFAR100	
	HSV	RT	HSV	RT
Standard Trained	46.04%	29.12%	24.50%	37.48%
HSV AdvTrained	**90.47%**	24.77%	**66.19%**	35.21%
RT AdvTrained	45.92%	**87.34%**	27.93%	59.63%
EdgeFool AdvTrained	43.07%	52.68%	26.22%	42.15%
Integration AdvTrained	**91.28%**	**86.70%**	64.84%	**62.10%**

Table C.1: Adversarial training results for random perturbations.

From these results we see that although the different models are trained on adversarial samples of minimal perturbation, they perform quite well on random perturbations as well. In all but one instance does the integrated model achieve optimal accuracy compared to the single-attack trained models.

Table C.2 shows the accuracy for all generated perturbations, including non-adversarial ones. In these experiments too the multi-perturbation model achieves optimal results in most instances.

	CIFAR10			CIFAR100		
	HSV	RT	EdgeFool	HSV	RT	EdgeFool
Standard Trained	87.38%	69.62%	49.82%	49.11%	67.27%	34.39%
HSV AdvTrained	**93.07%**	65.72%	46.56%	**68.29%**	65.01%	31.53%
RT AdvTrained	86.65%	**91.02%**	54.26%	47.69%	70.71%	33.51%
EdgeFool AdvTrained	87.78%	78.71%	**84.45%**	47.93%	67.16%	**61.88%**
Integration AdvTrained	**94.07%**	**90.99%**	81.67%	**67.53%**	**72.09%**	**61.81%**

Table C.2: Adversarial training results for all perturbations.