# Adversarial Training with Layerwise Origin-Target Synthesis

Bachelor's Thesis

**Michael Pablo Hodel**
17-714-619

**Submitted on**
16.04.2021

**Thesis Supervisor**
Prof. Dr. Manuel Günther

University of Zurich

Department of Informatics

**Bachelor's Thesis**

**Author:** Michael Pablo Hodel, michael.hodel@uzh.ch

**Project period:** 19.10.2020 - 19.04.2021

Artificial Intelligence and Machine Learning Group
Department of Informatics, University of Zurich

# Abstract

Deep neural networks are capable of solving various tasks very well and in certain cases even outperform humans. However, they are vulnerable to inputs designed to cause the network to give faulty output, so-called adversarial examples. There does not yet exist a solution for the problem that they pose with respect to the robustness of neural networks. However, there are various techniques to alleviate the vulnerability of neural networks towards adversarial examples. Adversarial training, that is, incorporating adversarial examples into the training data, is amongst the most effective currently existing defense techniques. Layerwise Origin-Target Synthesis (LOTS) is a method to create adversarial examples. This work inspects how well adversarial training increases the robustness of neural networks against adversarial examples created using LOTS.

# Zusammenfassung

Mehrschichtige neuronale Netze sind in der Lage, verschiedene Probleme sehr gut zu lösen und in bestimmten Fällen sogar den Menschen zu übertreffen. Sie sind jedoch anfällig für Eingaben, die darauf ausgelegt sind, das Netzwerk zu einer fehlerhaften Ausgabe zu veranlassen, so genannte Adversarial Examples. Es gibt noch keine Lösung für das Problem, das sie bezüglich der Robustheit von neuronalen Netzen darstellen. Es gibt jedoch verschiedene Techniken, um die Anfälligkeit neuronaler Netze gegenüber solchen Beispielen zu vermindern. Adversarial Training, das heisst das Einbringen von Adversarial Examples in die Trainingsdaten, gehört zu den effektivsten derzeit existierenden Abwehrtechniken. Layerwise Origin-Target Synthesis (LOTS) ist eine Methode zur Erzeugung von Adversarial Examples. In dieser Arbeit wird untersucht, wie gut Adversarial Training die Robustheit von neuronalen Netzen gegen LOTS-Beispiele erhöht.

# Contents

Chapter 1

# Introduction

## 1.1 Adversarial Examples

Despite their impressive performance in a wide range of well-defined problems, deep neural networks have shown to be vulnerable to adversarial examples (Szegedy et al., 2014). An adversarial example with respect to a certain model is an input created by applying slight perturbations to a benign input — from which the model correctly infers — such that the model produces an incorrect output.

An incorrect output is well-defined in case of classification tasks, as for classification tasks, an output is incorrect if the predicted class does not correspond to the correct class and is correct if it does. However, adversarial examples also exist in the realm of regression tasks, where an incorrect output does not exist per se, but would have to be defined, for example by determining whether or not an output significantly deviates from the correct output with respect to a threshold of relative differences. Adversarial examples are also not restricted to neural networks. Due to their prominence however, adversarial examples have been most exhaustively studied in the context of neural networks and computer vision. This work focuses on deep neural networks for image classification.

Techniques for creating high-quality adversarial examples typically rely on slightly perturbing a regular example. Usually, at least in contexts where the model input corresponds to data also explicitly meaningful for humans — as is for example the case for image, text or speech data — the definition is further constrained by requiring the representation depicted by the adversarial example to be recognizable as the representation depicted by its corresponding regular example, thus giving the impression of a valid input. Nevertheless, the definition of adversarial examples is not straight-forward. Whether the perturbations have to be imperceptible to a human for an example to be considered adversarial, or whether it is sufficient for the perturbed example to be recognizable as the unperturbed example, is not well-defined.

## 1.2 Motivation

The obvious reason researchers and practitioners of machine learning may want to care about adversarial examples is computer security, meaning defending models against adversarial examples. Especially for security-critical applications such as face recognition and self-driving cars, models vulnerable to adversarial examples may cause large costs, not necessarily financial — given the existence of parties with malicious intent and access to the model at hand. But computer security is not the only reason why one may want to care about adversarial examples. The internals of deep neural networks, more specifically the features that the models learn and use to distinguish between examples of different classes, are still largely a blackbox. There have been various attempts at brightening this box, e.g. highlighting features of an input in favour for or against a specific class (Zintgraf et al., 2017) or visualizing deep feature representations. However, those visualizations do not necessarily correspond to something meaningful to the human eye. The existence of adversarial examples implies that the learned features do not match the ones that humans learn. If this were not the case, neural networks would not be vulnerable to adversarial examples, since humans by definition are not susceptible to adversarial examples. Thus, research in adversarial machine learning — not to be confused with generative adversarial networks (Goodfellow et al., 2014) — is potentially also very promising with respect to a better understanding and improvement of deep neural networks in general. It is to be assumed that moving towards models better learning features utilized by humans implies moving towards models more robust towards adversarial examples.

## 1.3 Origin

A natural question to ask is why adversarial examples exist in the first place. Despite great efforts and various hypotheses, there is no consensus on what exactly makes room for their existence. What certainly is the case is that the features learned by neural networks are less robust towards adversarial examples as opposed to the features learned by humans (Ilyas et al., 2019). For example, it was shown that convolutional neural networks can be biased towards texture and that biasing the models more towards shape — assumably corresponding more to human-learned features — improves accuracy and robustness (Geirhos et al., 2019). While the feature-robustness hypothesis is sound, it is not an explanation in and of itself, meaning, it just gives a higher-level overview and still leaves room for what in particular causes the learning of non-robust features in the first place.

There are various not always disjoint hypotheses on why adversarial examples exist. That is, there may be more than one peculiar feature of models and their treatments that in combination or each separately give rise to the existence of adversarial examples. For example, it has been argued that the vulnerability of the models towards adversarial examples is related to the high-dimensionality of the input space (Xu et al., 2018) or the linearity of the models (Goodfellow et al., 2015).

## 1.4 Attacks

An adversarial attack refers to inputting a set of adversarial examples to a model. In light of the attacker party's intentions and available options, there are two major axes along which one can classify adversarial attacks in the realm of classification tasks. The first one concerns the attackers intention and is the distinction between an untargeted and a targeted attack. While an untargeted attack uses adversarial examples created with the intent of them being classified incorrectly, a targeted attack uses inputs intended to be misclassified as a specific target class or as one of a set of target classes. In the case of LOTS however, as will be elaborated in section 2.1.2, the notion of a target does not directly correspond to a class.

The second axis concerns the degree of insight the attacker has into the model and distinguishes between whitebox- and blackbox-attacks. In the case of a whitebox-attack, the attacker has full access to the model, and potentially to hyperparameters, the fitting procedure, etc., whereas in the case of a blackbox-attack, the attacker only has access to the network output, potentially even without full knowledge of the class domain. Due to these details, one might be better off thinking about a spectrum of graybox scenarios with the two extreme ends being the attacker having no information and all information, as opposed to a binary blackbox- and whitebox-setting (S. et al., 2018). The vast majority of whitebox-attacks rely on computing the gradients of an adversarial loss with respect to the input values and perturbing the input values based on those gradients (Goodfellow et al., 2015). Thus, essentially, backpropagation on the input is performed — similar to how backpropagation on the network weights is performed during network training. Given this reliance of whitebox-attacks on the gradients, high-quality adversarial examples can typically only be computed in an iterative manner. This introduces a trade-off between using few iterations and lower adversarial quality and many iterations, which can be computationally costly but leads to higher adversarial qualities. Adversarial quality refers to the ratio of attack success rate to perturbations. The attack success rate — when looking at multiple perturbed examples — is simply the share of examples which reach the goal of the attack, e.g. reaching the target classes. The perturbation denotes the difference between an example and its corresponding adversarial example. Adversarial losses are typically defined on the output probabilities. Therefore, targeted attacks require more perturbation than untargeted ones, as it is easier to decrease the probability of the true class until it is not the highest anymore than it is to increase the probability of a target class until it is the highest — given that the target class is not the easiest false class to reach. While real-world whitebox-attacks are less realistic, as getting access to the internals of a deployed model, both with respect to its state and its fitting procedure, typically is not straight-forward for an attacker, they are the worst-case scenario and therefore of interest. This work focuses on targeted whitebox-attacks.

An additional distinction of settings that is relevant with respect to security-critical applications is the input acquisition process. When the acquired input is digital, an adversarial example may experience no transformation between the point of acquisition and input to the modeling pipeline, thus the adversarial perturbation can take full effect. However, if the acquisition process transforms the input, for example by compression or scaling, carefully designed perturbations may become ineffective. This is especially the case for non-digital input, such as input from cameras or scanners, where the input inevitably suffers some small transformations. It has been shown that in case of non-digital input, these transformations already alleviate a lot of the security threat posed by adversarial examples (Graese et al., 2016).

## 1.5 Defenses

Defending against adversarial attacks is difficult. Many different defense techniques have been studied and suggested to be very effective or sometimes even to be entirely solving the issue of adversarial examples, but almost all of them have been shown to be largely useless. One example of such a defense is Defensive Distillation (Papernot et al., 2016). This defense method was shown to not be secure in a paper shortly afterwards (Carlini and Wagner, 2016), in which the authors point out that it is insufficient for a proposed defense method to be secure against existing attacks. Much like in other areas of computer security, improving adversarial attacks and defenses behaves a lot like an arms race.

On a high level, adversarial defenses may be divided into two categories, namely detection and robustness (Wiyatno et al., 2019). Detection means that the goal of the model, on top of correctly doing inference from real examples, is to detect adversarial examples as such, and typically to reject them. One possible approach for detecting adversarial examples in a classification setting is to include an extra class, which a model would predict whenever it perceives an example as adversarial. This relates to Open Set Recognition (Scheirer et al., 2013), where, as opposed to simple standard use cases, the class domain is not assumed to be a fixed set, but is such to allow for an input to be classified as not belonging to any of the known and well-defined classes, thus effectively to not classify it. There are other approaches that can be used to try to detect adversarial examples. For example, one can train two separate networks that are applied in succession: The first network classifies the examples as adversarial or benign and the second network does regular inference and is fed only the benign examples. Another approach is to empirically look at differences between adversarial and benign examples and then use the gained knowledge explicitly.

The other defense strategy is robustness, where the aim — in a classification setting — is to classify the adversarial example as the true class of its corresponding original counterpart. Robustness is defined as the invulnerability of the model to perturbations to benign examples, including adversarial perturbations. Thus, the robustness defense strategy is aiming for models that are more robust towards adversarial examples, as opposed to simply trying to detect and reject them. Within this robustness defense realm, there are various approaches, most predominantly input manipulation and data augmentation. Input manipulation refers to transforming the inputs at evaluation time, before feeding them into the model, with the intent to alleviate the strength of the features causing the network to make a false prediction. Such manipulations may be deterministic, in the case of images for example smoothing or enhancing shape features, but also random, such as adding some form of noise or randomly scaling, shifting or rotating the images (Xie et al., 2018).

Data augmentation refers to using perturbed versions of real examples at training time, where the perturbations may be of the types mentioned just above or adversarial. Arguably the most effective data augmentation defense technique with respect to robustness is adversarial training (Goodfellow et al., 2015). Adversarial training means incorporating adversarial examples — typically created from perturbing some or all of the initial training data — into the learning process, either supplementary to the training data or exclusively. Often, the loss is adjusted to be a weighted sum of the loss on the original and the loss on the adversarial examples. Adversarial training in the case of an untargeted attack may thus be a min-max optimization problem, where the inner maximization problem consists of creating adversarial examples that maximize e.g. the loss of the model's prediction with respect to the true target class, given the adversarial example. In the case of a targeted attack, adversarial training may correspond to a min-min optimization problem. This work focuses on adversarial training as a defense method. There are various improvements to regular adversarial training, such as Curriculum Adversarial Training (Cai et al., 2018) as elaborated in section 2.2.1 or Dynamic Adversarial Training (Wang et al., 2019) as elaborated in section 2.2.2. Nevertheless, despite the improvement in robustness that adversarial training enables, it is not a bulletproof defense, let alone solves the problem of adversarial examples.

Since there seems to be a trade-off between robustness and accuracy (Tsipras et al., 2019), probably the greatest drawback of increasing the robustness of a model towards adversarial attacks is that it typically comes at the expense of a decreased accuracy on regular examples. In use cases where the vast majority of the examples are not adversarial or the cost of not being robust against adversarial examples is negligible, it thus may simply not be worth it to sacrifice this accuracy for a marginal robustness increase. Even in other use cases it may not be desirable to defend against adversarial attacks, as truly robust networks cannot be formed as of now anyway, thus there will be vulnerability to a sophisticated attack either way. One way to try to prevent attacks reliant on gradients is gradient masking, namely trying to prevent the attacker from getting access to gradients. Such methods however too have been shown to not be very effective (Athalye et al., 2018).

In the context of the discrepancy in the learned features between humans and neural networks causing a lacking robustness, it is worth noting that applying tweaks to the model architectures, training processes or the like of currently widely-used neural networks will not necessarily enable models capable of learning sufficiently human-like features. There are two major reasons why this is the case. First, the architectures of neural networks are significantly different from brain architectures. Second, the learning process is different: When humans are confronted with e.g. visual object classification tasks, they utilize a vast range of prior knowledge, not all of which emerged from previously having performed visual object classification. In contrast, most modern deep neural networks utilize only a very small number of priors — as for example translational invariance enabled by convolutional neural networks — and almost all the model's knowledge used for inference therefore comes from the actual task at hand, meaning the training data. Thus, there might simply be crucial information missing. From that perspective, moving neural networks closer towards a reverse-engineered human brain, including both architecture and learning, may be a promising way to go with respect to arriving at a model's learned feature set closer to the one of humans and therefore one better enabling robustness towards adversarial examples.

## 1.6 Evaluation

Crucial to the whole field of adversarial machine learning is how adversarial attacks and defenses are evaluated. This is a difficult problem. Typically, when evaluating an attack, the attack success rate is of interest to the attacker. In the case of classification tasks, the attack success rate is defined as the share of adversarial examples where the target class is reached for targeted attacks, and the share of adversarial examples where the predicted class is not equal to the true class for untargeted attacks. From the perspective of the defender however, the accuracy is of primary interest.

Generally, evaluating the robustness of a model is not straight-forward and improper evaluation methods may give a false sense of robustness and even lead to claims of bulletproof defense methods where there are none. One of the big issues with evaluating attacks and defenses is that fooling a model via an adversarial example implies a correct classification of the corresponding unperturbed example by a human and, depending on the definition, also an only weakly perceptible perturbation. Adversarial examples are typically generated from existing genuine examples by slightly perturbing them. From the attacker's perspective, the perturbation is ideally barely perceptible or even entirely imperceptible to a human. Asking humans to judge examples based on whether they are reasonably similar to or indistinguishable from genuine examples is rather infeasible on a large scale and also infeasible to be consistent and reliable. Thus, the need for a computable notion of perceptual similarity or dissimilarity between an adversarial example and its genuine counterpart arises.

When evaluating adversarial examples in a classification setting, there are two main approaches one can take. The first is to perturb the input until it is misclassified or reaches its target class and then look at the amount of perturbation that was required. The second approach is to limit or fix the perturbation and then check whether or not the generated adversarial example is misclassified or classified as the given target class. Perturbation here denotes the difference between the original input and its adversarial counterpart, in the case of images thus corresponding to pixel differences. Therefore, defining suitable metrics to measure the perturbation or (dis-)similarity between genuine images and their corresponding adversarial images is essential. Defining reasonable thresholds for dissimilarity or similarity measures that e.g. correspond to the boundary where the human is indecisive if the example is genuine or adversarial may, depending on the use case, be important, but leads back to the issue of having to rely on human evaluation that is intended to be solved in the first place. However, an evaluation may not need to depend on absolute statements such as whether or not a perturbation surpasses a specified level of tolerance: Often, different defense and attack settings are compared with each other, and similarity metrics allow making statements about the performance of one setting relative to other settings.

Obvious and widely used candidates for dissimilarity metrics for working with images are the $L_p$ norms of the perturbations. Mainly, $L_2$ and $L_\infty$ are used. However, the $L_p$ norms have

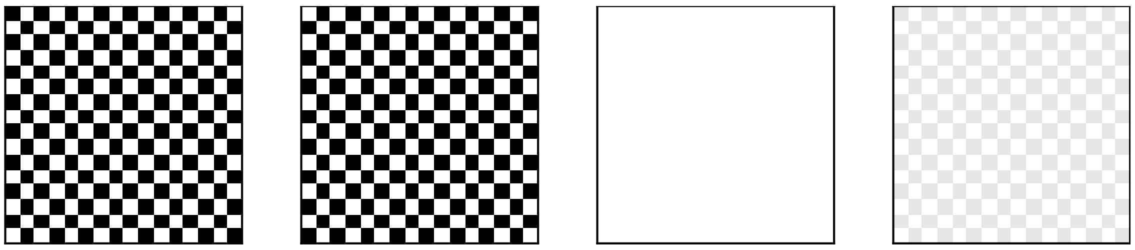


Figure 1.1: Image Dissimilarities Example

the downside that they are not guaranteed to well correspond to a human notion of dissimilarity. Figure 1.1 displays four 16 by 16 grayscale images to illustrate this. A human would presumably value the dissimilarity between the two left images as very small, but both the mentioned $L_p$ norms — as these two images have flipped pixel intensities — yield the maximum possible dissimilarity values of $L_2 = 16$ and $L_\infty = 1$. Conversely, the two images on the right are visibly different, but exhibit only small dissimilarity values of $L_2 = 1.1314$ and $L_\infty = 0.1$. These examples are rather unnatural, but show that indeed the $L_p$ norms are flawed.

There are other, more sophisticated metrics that try to capture human-perceived similarity more accurately, such as the SSIM, short for structural similarity index measure (Wang et al., 2004), which is presented in section 3.1. There exist even more sophisticated metrics trying to take into account invariance of perception towards transformations such as shifting. One such metric is the PASS, short for perceptual adversarial similarity score (Rozsa et al., 2016b), which combines SSIM with the photometric-invariant homography transform alignment.

## 1.7 Transferability

It has been shown that sometimes, an adversarial example successfully attacking one model is also capable of fooling a model different from the one that was used to create it. Thus there exists a transferability, which enables attackers to attack the target model via applying whitebox-attacks on a different model. It has been shown that attacks transfer better between networks with similar architectures (Rozsa et al., 2016a). After a certain degree of perturbation, stronger attacks — that is, attacks that lead to a greater perturbation — are less transferable due to overfitting (Kurakin et al., 2017b), similar to how models can overfit to yield very high accuracies on the training data but poor accuracies on the test data. Also, targeted attacks are typically less transferable than untargeted attacks (Liu et al., 2017). Further, it has been shown that with a sufficient number of queries to the target model, an attacker can reverse engineer the target model to a reasonable degree of accuracy (Oh et al., 2018) and thus is able to run attacks that are highly transferable. However, this attack is assumably infeasible in practice, as queries to deployed models are typically limited.

In chapter 2, attack and defense methods relevant for this work are discussed. Chapter 3 explains how some of these methodologies are adapted for this work and elaborates on the chosen approaches. Chapter 4 presents the experiments and their results and concludes the work.

Chapter 2

# Related Work

## 2.1 Related Attacks

There is a vast range of methods to generate adversarial examples. The work "Adversarial Examples in Modern Machine Learning: A Review" (Wiyatno et al., 2019) for example covers about 30 different attacks, most of which are whitebox-attacks. FGSM, short for Fast Gradient Sign Method (Goodfellow et al., 2015) for example — in the untargeted setting — simply perturbs the pixel values of an image by a small amount in the direction of the signs of the gradients with respect to the classification loss. A more sophisticated well-known attack is the Carlini & Wagner attack (Carlini and Wagner, 2017). The Carlini & Wagner attack creates adversarial examples with the minimal perturbation required for misclassification, but is computationally significantly more costly.

### 2.1.1 Projected Gradient Descent

PGD, short for Projected Gradient Descent (Kurakin et al., 2017a) is a whitebox-attack that can be either targeted or untargeted. Here, targeted PGD is presented. Let $\epsilon$ be the maximum perturbation, in $L_\infty$, and let $\alpha$ denote a step size. Also, let $p()$ be the projection of the suggested adversarial image back onto the error-ball $\chi$, which is the set of all adversarial images $x_p$ which fulfill the perturbation constraint given by $\epsilon$, so $\chi = \{x_p : \|x - x_p\|_\infty \leq \epsilon\}$. Let $c()$ be clipping the pixel values to be in the valid range and let $r()$ be rounding the pixel values to be compliant with the standard image encoding that only allows 256 different intensities per channel. Let $L()$ be the adversarial loss function, e.g. Cross-Entropy-Loss, let $f(\theta, x)$ be the forward-pass of an image $x$ through a network $f$ with parameters $\theta$, let $sgn()$ be the sign function, and let $y$ be the target classes. Then, the PGD adversarial example $x_p$ based on its corresponding original example $x$, is calculated as in equation (2.1).

$$x_p = r(c(p(x - \alpha \cdot sgn(\nabla_x L(f(\theta, x), y))))) \tag{2.1}$$

PGD can be used iteratively by applying the procedure (2.1) multiple times successively and after each iteration resetting $x$ to equal $x_p$ from the previous iteration. The projection function $p()$ ensures that the adversarial example has a perturbation not exceeding $\epsilon$ and effectively performs element-wise clipping of each pixel value of the suggested adversarial image $x_p$ at location $i, j$, denoted by $x_{p,i,j}$, to be in the range $[x_{i,j} - \epsilon, x_{i,j} + \epsilon]$. This is done for each color channel. Thus $p()$ essentially updates $x_{p,i,j}$ to be $max(min(x_{p,i,j}, x_{i,j} + \epsilon), x_{i,j} - \epsilon)$. The projection function can be adjusted for other $L_p$-norms. The function $c()$ clips all pixel values of the adversarial image $x_p$ resulting by subtracting the scaled perturbation from the original image to be in the range $[0, 1]$, so $c(x_{p,i,j}) = min(max(0, x_{p,i,j}), 1)$. The function $r()$ rounds the pixel values to their closest multiples of $1/255$.

### 2.1.2 Layerwise Origin-Target Synthesis

LOTS, short for Layerwise Origin-Target Synthesis (Rozsa et al., 2017) is a targeted whitebox-attack method that, like PGD and most sophisticated attacks, also relies on gradients. Let $f^{(l)}(\theta, x)$ be the activation of the $l$-th layer of the neural network $f$ with parameters $\theta$, given an input image $x$, and let $L(t^{(l)}, f^{(l)}(\theta, x))$ be the adversarial loss between a target activation $t^{(l)}$ of the $l$-th layer of the network and the layer activation $f^{(l)}(\theta, x)$ resulting from the input $x$. LOTS allows for this target activation to be any tensor of matching shape, be this a tensor filled with random values or the activation resulting from feeding the network an image. The adversarial loss is defined to be the mean squared error. Then the gradients of the loss between the activation of layer $l$ of model with parameters $\theta$, given input $x$, and the target activation $t^{(l)}$, with respect to the image $x$, are defined as $d = \nabla_x L(f^{(l)}(\theta, x),\ t^{(l)})$ and indicate the direction in which changing the pixel values of $x$ yields a larger loss, larger values indicating greater influence. Let $g(d)$ denote scaling the gradients to be in the range $[-1, 1]$, namely $g(d) = \frac{d}{\|d\|}$. The LOTS adversarial image $x_p$ in the case of a single iteration is then be defined as in equation (2.2), where the notations from section 2.1.1 for $r()$ and $c()$ also apply.

$$x_p = r(c(x - g(\nabla_x L(f^{(l)}(\theta, x),\ t^{(l)})))) \tag{2.2}$$

Again, this procedure can be applied iteratively. LOTS differs from attacks like targeted PGD in that it does not target a specific class, but a target activation at a specific layer. LOTS thus allows attacking networks that are not end-to-end, meaning, rely on using deep features or logits as opposed to the output probabilities or classes. Nevertheless, LOTS can also be used to perform end-to-end attacks, for example by taking the mean layer activations of a set of examples from a specific target class. The closer the target layer to the output layer, the greater the chance of this approach to produce output probabilities that predict the target class. One of the main advantages of LOTS compared to attacks like targeted PGD, despite the two being fairly similar, is that, for a fixed model, input and target class, it can produce a vastly greater number of adversarial examples. This is due to two additional degrees of freedom, one being the target layer of choice and the other being the specific target activation. Another advantage of LOTS over targeted PGD is that its gradient max-scaling weighs the perturbation strength of each pixel by the influence of its change in value on the adversarial loss, thus not unnecessarily perturbing non-influential pixels too strongly.

## 2.2 Related Defenses

As the case with attacks, there is also a vast range of defense methods that try to alleviate a model's vulnerability towards adversarial attacks, some of which are briefly discussed in section 1.5. Further, there is for example Ensemble Adversarial Training (Tramèr et al., 2018), which extends adversarial training by augmenting the training data with adversarial examples formed by other models. This defense method won the 2017 *NIPS competition on Defenses against Adversarial Attacks*. Another sophisticated defense technique is Defensive Distillation (Papernot et al., 2016), which is a two-stage training technique that essentially trains an additional model using the output probabilities from a previous model instead of the vectors one-hot-encoding the classes. Both Ensemble Adversarial Training and Defensive Distillation however have shown to not be bullet-proof defenses.

### 2.2.1 Curriculum Adversarial Training

The concept of curriculum learning in the context of machine learning dates back to almost three decades ago (Elman, 1993) and has explicitly been named such in (Bengio et al., 2009). The underlying idea is to present the examples to the model at training time in a meaningful order, that is with increasing difficulty. This allows for better generalizations of the trained models. It can be viewed as approximately analogous to a human being presented with increasingly difficult problems when learning a task as opposed to learning from either a series of examples with fixed or random difficulty. Such a curriculum can also be defined for adversarial examples in a straightforward manner, namely by gradually increasing the perturbation of the adversarial examples at training time, for example by increasing the number of iterations or step size. This has been done and is termed Curriculum Adversarial Training correspondingly (Cai et al., 2018).

### 2.2.2 Dynamic Adversarial Training

In the work "On the Convergence and Robustness of Adversarial Training" (Wang et al., 2019), the authors tackle the issue of how to properly evaluate adversarial attacks. They propose a criterion — called the First-Order Stationary Condition (FOSC) — that measures the quality of generated adversarial examples and use that criterion in a new training method called Dynamic Adversarial Training, which can be seen as a form of Curriculum Adversarial Training. As outlined in section 1.6, assessing the quality of adversarial examples is crucial. Solely looking at the perturbation defeats the purpose, as it entirely neglects the question of how well the model at hand can handle it with respect to predicting the true class. Similarly, it is also not sufficient to only look at the attack success rate, as the adversarial examples may have little perceptual similarity to their original counterparts. FOSC combines the two notions of perturbation constraint and adversarial loss into one metric, the FOSC, measuring the convergence quality of adversarial examples. Let $\chi$ again be the error ball. Let $f()$ be the forward pass, let $L()$ be the loss function and let $y$ be the true class of the image $x$ from which $x_p$ is formed. FOSC of the adversarial example $x_p$, denoted with $c(x_p)$, in the untargeted setting, is then defined as in equation (2.3). The authors show that FOSC is equal to the closed-form expression depicted in equation (2.4).

$$c(x_p) = max_{x' \in \chi} \langle x - x', \nabla_x L(f(\theta, x'), y) \rangle \tag{2.3}$$

$$c(x_p) = \epsilon \|\nabla_{x_p} L(f(\theta, x_p), y)\|_1 - \langle x_p - x, \nabla_{x_p} L(f(\theta, x_p), y) \rangle \tag{2.4}$$

FOSC is inversely correlated with adversarial strength, that is, the perturbation, taking the loss into account. Lower FOSC values thus indicate stronger attacks. Dynamic Adversarial Training then works by — at each training epoch $t$ and for each training example image — iteratively perturbing the image as long as it exhibits a FOSC value exceeding a defined lower threshold $c_t$. The value $c_t$ is defined to be $c_t = max(c_{max} - t \cdot c_{max}/T', 0)$, where $c_{max}$ denotes the maximum FOSC value and $T'$ denotes a control epoch, both specified as hyperparameters for training. Over the epochs, $c_t$ thus decreases from $c_{max}$ at the first epoch — indexed with 0 — to zero at the control epoch $T'$. The control epoch $T'$ is chosen to be slightly smaller than $T$, which enables the network to be trained on adversarial images with $FOSC = 0$ during the last epochs $t \geq T'$. However, Dynamic Adversarial Training also specifies a cap of maximum $k$ iterations, thus reaching the $c_t$ threshold is not guaranteed.

# Chapter 3

# Methodology

This work investigates adversarial examples created using LOTS and PGD within the scope of image classification tasks using deep neural networks. Specifically, whitebox-scenarios as well as some graybox scenarios where the attacker is unaware of the attack method used during adversarial training are considered and evaluated against adversarially trained networks using regular adversarial training, Curriculum Adversarial Training and Dynamic Adversarial Training. As LOTS is a targeted attack and this work makes comparisons between LOTS and PGD, this work considers targeted PGD. The experiments are implemented in Python, version 3.8.5, mainly making use of the open source machine learning library PyTorch. This chapter introduces the SSIM, explains how LOTS and Dynamic Adversarial Training are adapted, how the adversarial images are generated and how the models are trained.

## 3.1 Structural Similarity Index Measure

This work focuses on $L_\infty$ but also makes use of the SSIM (Wang et al., 2004). The SSIM tries to measure the similarity between two images. This work uses the SSIM to measure the similarity between the generated adversarial examples and their original counterparts in a setting of not using a fixed number of iterations for attacking. Let $x$ and $x_p$ be two images, in the case of this work a regular image $x$ and its adversarial counterpart $x_p$. The SSIM is a weighted sum of measures capturing the differences in luminance $l(x, x_p)$, contrast $c(x, x_p)$ and structure $s(x, x_p)$. These three measures are defined in equation (3.1).

$$l(x, x_p) = \frac{2\mu_x \mu_{x_p} + c_1}{\mu_x^2 \mu_{x_p}^2 + c_1} \qquad c(x, x_p) = \frac{2\sigma_x \sigma_{x_p} + c_2}{\sigma_x^2 + \sigma_{x_p}^2 + c_2} \qquad s(x, x_p) = \frac{\sigma_{x x_p} + c_3}{\sigma_x \sigma_{x_p} + c_3} \tag{3.1}$$

Here, $\mu_x$ and $\mu_{x_p}$ be the mean of $x$ and $x_p$ respectively, $\sigma_x$ and $\sigma_{x_p}$ be the variance of $x$ and $x_p$ respectively and $\sigma_{x,x_p}$ is the covariance of $x$ and $x_p$. The terms $c_1$, $c_2$ and $c_3$ are stabilization constants. Let $\alpha, \beta$ and $\gamma$ be weighing powers each proportional to the relevance of its corresponding term $l(x, x_p), c(x, x_p)$ and $s(x, x_p)$. Then, the SSIM is given by equation (3.2).

$$SSIM(x, x_p) = l(x, x_p)^\alpha \cdot c(x, x_p)^\beta \cdot s(x, x_p)^\gamma \tag{3.2}$$

The SSIM does not exist for images with multiple color-channels per se. This work makes use of the Python image processing library scikit-image implementation of SSIM, which uses averaged independent similarity calculations of each channel to compute the SSIM for multi-channel images.

## 3.2 Adaption of Dynamic Training

As the Dynamic Adversarial Training strategy utilizes the FOSC metric, which assumes an untargeted attack, but this work investigates targeted attacks, the FOSC metric needed to be adjusted correspondingly. This is achieved by using the negative loss with respect to the target class $y_t$ as opposed to using the loss with respect to the true class $y$, which results in the equation 3.3.

$$c(x_p) = \epsilon \|\nabla_{x_p} - L(f(\theta, x_p), y_t)\|_1 - \langle x_p - x, \nabla_{x_p} - L(f(\theta, x_p), y_t)\rangle \tag{3.3}$$

## 3.3 Adaption of LOTS

To enable a scenario of limited perturbations and make LOTS more comparable to PGD, the projection function discussed in section 2.1.1 is also applied for LOTS, as well multiplying the suggested perturbation by a step size $\alpha$. This results in the adjusted algorithm for LOTS as depicted in equation 3.4.

$$x_p = r(c(p(x - \alpha \cdot g(\nabla_x L(f^{(l)}(\theta, x),\ t^{(l)})))))) \tag{3.4}$$

For these experiments, during adversarial training, the rounding function $r()$ as discussed in section 2.1.1 is deliberately not applied, both for LOTS and PGD. It is only applied at evaluation time.

To make LOTS comparable to the end-to-end-attack PGD, this work uses LOTS to indirectly target classes. To do so, as the targets in the case of LOTS are not classes but layer activations, a layer for which the targets are created has to be determined. This work uses the last network layer, which, as the conversion to probabilities is performed outside of the networks, corresponds to the logits. Since the aim is to create targets leading to a classification of the adversarial example as a specific target class, the experiments use the mean logits of a small set of images from the chosen target class as targets. The averaging is done element-wise. During adversarial training using LOTS, the last 50 logits for each class — formed by feeding the original training examples through the network — are kept as templates and are averaged to create the targets. For evaluation, the targets for creating adversarial images on the test set are fixed and formed for each target class by averaging the logits of all training examples that are correctly predicted by the unsecured, regularly trained model.

This approach introduces the undesired effects that for some of the first images, there are no templates available, in which case a vector of random values serves as the target, and that for some of the first images for which templates are available, the number of templates is smaller than 50. The effect of having rubbish adversarial examples due to using random targets for training is negligible, as the expected number of corresponding rubbish adversarial examples created is very small in comparison to the size of the training set. The former design choice — as opposed to not creating adversarial examples when no templates are available — is made for practical purposes, specifically to have an adversarial example for each benign example, which facilitates implementation and allows for an equal number of adversarial examples for all settings. The effect of having adversarial examples created using targets consisting of fewer than 50 templates, despite affecting more images, is also negligible. Firstly it is also only a small fraction of examples that is affected by this and secondly, targets formed from fewer than 50 templates also constitute valid targets. The procedures for handling templates and generating targets for LOTS attacks are depicted in algorithms 1 and 2.

**Algorithm 1:** Computing LOTS Targets

```
targetClasses: a list containing a target class for each image in the batch
templates    : a list containing a list of logits for each class in the class domain
nExamples ← length of targetClasses;
nClasses ← length of templates;
targets ← initialize an empty list;
for i ← 0 to nExamples - 1 do
    targetClass ← element of targetClasses at index i;
    classTemplates ← element of templates at index targetClass;
    if length of classTemplates = 0 then
        target ← vector with random values in [0, 1] of length nClasses;
    else
        nTemplates ← length of classTemplates;
        target ← $\frac{1}{nTemplates} \sum_{j=0}^{nTemplates-1}$ element of classTemplates at index j;
    end
    append target to targets;
end
return targets;
```

**Algorithm 2:** Computing LOTS Templates

```
classes          : a list of image classes
adversarialLogits: a list containing the logits of the adversarial images in the batch
templates        : a list containing a list of logits for each class in the class domain
nExamples ← length of classes;
for i ← 0 to nExamples - 1 do
    logits ← element of adversarialLogits at index i;
    class ← element of classes at index i;
    append logits to element of templates at index class;
    nTemplates ← length of element of templates at index class;
    if nTemplates > 50 then
        remove first element of element of templates at index class;
    end
end
return templates;
```

## 3.4 Creating Adversarial Images

The adversarial image generation process is depicted in algorithms 3, 4 and 5. Algorithm 3 implements the iterative LOTS and PGD attacks outlined in sections 2.1.2, 2.1.1 and 3.3. The attacking algorithm uses algorithm 4, which implements one iteration of perturbation. Algorithm 5 updates a control vector indicating for each image whether or not to perturb it further, which is required for the Dynamic Adversarial Training setting and the attack setting with non-fixed iterations. How these algorithms are used for the experiments is explained in section 4.1.2 in more detail.

**Algorithm 3:** Computing Adversarial Images

```
f                  : a neural network
images             : a list of images
classes            : a list of true classes corresponding to the images
targetClasses      : a list containing a target class for each image
LOTSTargets        : a list containing a target for each image
templates          : a list containing a list of logits for each class in the class domain
classDomain        : the set of possible classes
attackType         : an identifier
extraStopCriterion : additional criterion that stops perturbation if met
epochFOSCThreshold: lower threshold for FOSC
k                  : the number of iterations
$\epsilon$                  : the maximum perturbation in $L_\infty$
$\alpha$                  : the step size
nExamples ← length of classes;
adversarialImages ← copy of images;
if targetClasses is null then
    targetClasses ← initialize an empty list;
    for class in classes do
        availableTargetClasses ← classDomain - class;
        targetClass ← random element from availableTargetClasses;
        append targetClass to targetClasses;
    end
end
adversarialLogits ← forward-passing adversarialImages through f;
if attackType = PGD then
    targets ← targetClasses;
end
if attackType = LOTS then
    if LOTSTargets is null then
        targets ← apply algorithm 1 on (targetClasses, templates);
        templates ← apply algorithm 2 on (classes, adversarialLogits, templates);
    else
        targets ← one-hot encoded LOTSTargets;
    end
end
controlVector ← a list of length nExamples filled with ones;
controlVector ← apply algorithm 5 on (extraStopCriterion, adversarialLogits);
iterations ← 0;
while $\sum controlVector \geq 1$ and iterations < k do
    adversarialLogits ← forward-passing adversarialImages through f;
    adversarialImages ← apply algorithm 4 on (attackType, adversarialLogits,
      adversarialImages targets, classDomain, controlVector, $\alpha$, $\epsilon$, k);
    controlVector ← apply algorithm 5 on (extraStopCriterion, adversarialLogits,
      epochFOSCThreshold, controlVector);
    iterations ← iterations + 1;
end
return adversarialImages;
```

**Algorithm 4:** Perturbing Images

```
attackType         : an identifier
adversarialLogits  : a list containing the logits of the adversarial images
adversarialImages: a list containing the adversarial images
targets            : a list containing a target activation for each adversarial image
classDomain        : the set of possible classes
controlVector      : vector with values indicating whether or not to perturb each image
k                  : the number of iterations
$\epsilon$                  : the maximum perturbation in $L_\infty$
$\alpha$                  : the step size
nExamples ← length of targets;
if attackType = LOTS then
    adversarialLoss ← Mean Squared Error loss between adversarialLogits and targets;
end
if attackType = PGD then
    adversarialLoss ← Cross Entropy loss between adversarialLogits and targets;
end
gradients ← gradients of adversarialLoss with respect to adversarialImages;
perturbations ← a list of length nExamples filled with images with pixel values all zero;
for i ← 0 to nExamples - 1 do
    indicator ← element of controlVector at index i;
    if indicator = 1 then
        if attackType = LOTS then
            scalar ← max(abs(element of gradients at index i)) # element-wise abs;
            perturbation ← (element of gradients at index i) / scalar;
        end
        if attackType = PGD then
            perturbation ← sign(element of gradients at index i) # element-wise sign;
        end
        element of perturbations at index i ← perturbation;
    end
end
adversarialImages ← adversarialImages - $\alpha \cdot$ perturbations;
if $\epsilon$ is not null and $\alpha \cdot k > \epsilon$ then
    adversarialImages ← min(adversarialImages, adversarialImages + $\epsilon$) # pixel-wise;
    adversarialImages ← max(adversarialImages, adversarialImages - $\epsilon$) # pixel-wise;
end
adversarialImages ← clip(adversarialImages, 0, 1) # pixel-wise clipping;
return adversarialImages;
```

**Algorithm 5:** Control Vector Updating

```
extraStopCriterion    : additional criterion that stops perturbation if met
adversarialLogits     : a list with the logits of the adversarial images
epochFOSCThreshold: epoch-specific FOSC threshold
controlVector         : vector with values indicating whether or not to perturb the
                        images
classes               : a list of image classes
nExamples ← length of classes;
if extraStopCriterion = success then
    for i ← 0 to nExamples - 1 do
        output ← element of adversarialLogits at index i;
        predictedClass ← index of maximum value in output;
        trueClass ← element of classes at index i;
        if predictedClass = trueClass then
            set element of controlVector at index i to zero;
        end
    end
end
if extraStopCriterion = FOSCThreshold then
    for i ← 0 to nExamples - 1 do
        FOSCValue ← FOSC value of element of adversarialImages at index i;
        if FOSCValue < epochFOSCThreshold then
            set element of controlVector at index i to zero;
        end
    end
end
return controlVector;
```

## 3.5 Training

The training procedures used for the experiments are described in algorithm 6. This algorithm, on top of regular training and regular adversarial training, also implements the two additional adversarial training methods Curriculum Adversarial Training and Dynamic Adversarial Training discussed in sections 2.2.1 and 2.2.2. How this algorithm is used for adversarial training specifically is explained in more detail in section 4.1.2.

**Algorithm 6:** Model Training

```
M                    : neural network class
nEpochs              : the number of epochs
attackType           : an identifier
trainingType         : an identifier
maxFOSC              : maximum FOSC value for dynamic adversarial training
maxEpsilon           : maximum $\epsilon$ value for dynamic adversarial training
batchSize            : number of training data per batch
images               : a list of images
classes              : a list of true classes corresponding to the images
classDomain          : the set of possible classes
extraStopCriterion: additional criterion that stops perturbation if met
$k, \alpha, \epsilon$              : number of iterations, maximum perturbation, attack step size
f ← initialization of model class; initialize model parameters with random values;
if attackType = LOTS then
    templates ← initialize list containing an empty list for each class;
else
    templates ← null;
end
nExamples ← length of classes;
nBatches ← ⌈ nExamples / batchSize ⌉;
for epoch ← 0 to nEpochs - 1 do
    if attackType is not null then
        if trainingType = dynamic then
            controlEpoch ← ⌊ 0.8 · nEpochs ⌋;
            epochFOSCThreshold ← max(maxFOSC - epoch · maxFOSC/controlEpoch, 0);
        end
        if trainingType = curriculum then
            $\epsilon$ ← maxEpsilon · epoch / (nEpochs - 1);
            k ← epoch;
        end
    end
    for startIndex ← 0 to batchSize · nBatches by batchSize do
        endIndex ← min(startIndex + batchSize, nExamples - startIndex);
        batchImages ← images from index startIndex to startIndex + endIndex;
        batchClasses ← classes from index startIndex to startIndex + endIndex;
        trainOn ← initialize empty list;
        if attackType ∉ {dynamic, curriculum} then
            append batchImages to trainOn;
        end
        if attackType is not null then
            adversarialImages ← apply algorithm 3 on (f, batchImages, batchClasses,
              templates, classDomain, attackType, extraStopCriterion,
              epochFOSCThreshold, k, $\alpha$, $\epsilon$);
            append adversarialImages to trainOn;
        end
        for miniBatch in trainOn do
            logits ← forward-passing miniBatch through f;
            loss ← Cross Entropy loss between logits and classes;
            gradients ← gradients of loss with respect parameters of f;
            f ← f with parameter values updated using Stochastic Gradient Descent;
        end
    end
end
```

Chapter 4

# Experiments

The experiments investigate the effectiveness of adversarial training with respect to gains in robustness against adversarial examples created using LOTS. Primarily, the effects of varying defense and attack settings involving LOTS, as well as PGD, are studied. The central question is the following: *To what extent does adversarial training, including Curriculum Adversarial Training and Dynamic Adversarial Training particularly, increase the robustness of neural networks against adversarial images created using LOTS?*

## 4.1 Experiment Settings

### 4.1.1 Data and Models

To tackle the central question, the two standard datasets MNIST and CIFAR-10 alongside their standard classification problem are considered. That is, the use case is simple image classification, thus learning network parameters that lead to a large fraction of images from the test correctly classified by the network. The MNIST dataset consists of 60’000 grayscale images of 28 by 28 pixels, each depicting a handwritten digit between 0 and 9. The MNIST classes along with each one sample image are shown in figure 4.1. The CIFAR-10 dataset consists of 60’000 colored images of 32 by 32 pixels, each depicting an object of one of 10 classes. The MNIST classes along with each one sample image are shown in figure 4.2. The data is obtained via the torchvision datasets API. For each of those two datasets, 50’000 images are used as the training set for training the neural networks and 10’000 images are used as the test set for evaluation.

The neural networks are convolutional neural networks, which are still widely regarded as the state-of-the art methodology for image classification tasks. Large parts of the network architectures as well as certain hyperparameters used for the models and their fitting procedures are the same as for the experiments conducted in "On the Convergence and Robustness of Adversarial Training" (Wang et al., 2019). The neural network used to classify MNIST-images consists of two convolutional layers — each followed by batch normalization (Ioffe and Szegedy, 2015) and max-pooling — and two linear layers. Each convolutional layer increases the number of channels by a factor of three and each max-pooling layer uses a kernel size of two, thus decreases the feature map height and with each by a factor of two. The CIFAR-10 model consists of three convolutional blocks followed by two linear layers. Each convolutional blocks consists of two convolutional layers followed by batch normalization and max-pooling with a kernel size of two. Each convolution increases the number of channels by the constant three. In both cases, the rectified linear unit (ReLU) activation function is used. Also in both cases, the number of output features of the first linear layer, corresponding to the number of input features of the second linear layer, is cho-

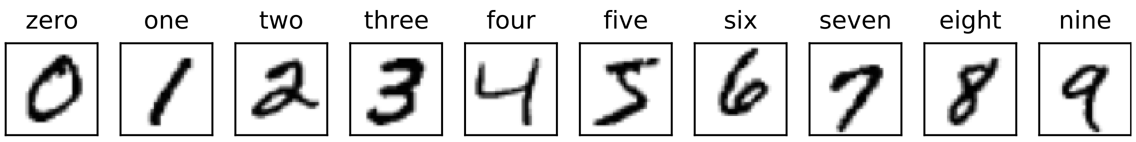


Figure 4.1: MNIST Classes and Images

Figure 4.2: CIFAR-10 Classes and Images

sen to be the closest integer to the geometric mean between the number of features in the flattened last feature map and the number of classes. Both networks are fitted using the stochastic gradient descent optimization technique with a learning rate of 0.01, momentum of 0.9, weight decay of $10^{-4}$ and a batch size of 32. As state-of-the-art accuracies of the networks is not the goal of this work and as both Curriculum Adversarial Training and Dynamic Adversarial Training require a fixed number of epochs, a fixed number of epochs is used for all training settings. The number of epochs for training the MNIST model is set to 8, the number of epochs for training the CIFAR-10 model is set to 16. These hyperparameters yield a layer structure corresponding to the output tensor shapes in table 4.1. For three-dimensional tensors, the first dimension denotes the number of channels and the last two denote the width and height of the images or feature maps and for one-dimensional tensors, the single dimension corresponds to the number of features. The batch normalization and activation operations as well as the Softmax layer at the end of the network are omitted in the representations as they do not change the tensor shapes. The first dimensions of the tensors, which denote the batch size, are also omitted.

### 4.1.2 Adversarial Image Generation

This work investigates a mainly whitebox-scenario. To tackle the main question, multiple attack methods are used for adversarial training and creating images to evaluate them. The two attack techniques considered are LOTS and PGD. Both are applied as a single-step and 10-step attack. For adversarial training, the two additional scenarios of Curriculum and Dynamic Adversarial Training are considered. The models described in table 4.1, each trained regularly with the hyperparameters as stated in section 4.1.1, serve as a benchmark. For both datasets, both attack types LOTS and PGD and for each setting 1-step attack, 10-step attack, Curriculum Adversarial Training and Dynamic Adversarial Training, an additional model with the same architecture and hyperparameters is trained adversarially. For all eight resulting adversarial training settings, the adversarial examples are created for each batch for each training epoch. For the 1-step and 10-step attacks, the adversarial examples are incorporating by an additional backward-pass, thus are essentially doubling the number of training data. The step sizes for LOTS-attacks are chosen to be twice as high as for PGD-attacks due to the applied max-scaling in LOTS which ceteris paribus would lead to a smaller perturbation. The 1-step attacks use a maximum perturbation of $\epsilon = 0.3$ and a step size $\alpha$ equal to 0.5 for LOTS and 0.25 for PGD. The 10-step attacks also use a maximum perturbation of $\epsilon = 0.3$, but a step size equal to 0.15 for LOTS and 0.075 for PGD. For Curriculum Adversarial Training and Dynamic Adversarial Training, the generated adversarial examples are not used supplementary to the original images during training, but instead of, thus leaving

**MNIST**

| Operation | Output Shape |
|---|---|
| *input* | $1 \times 28 \times 28$ |
| Conv. | $3 \times 28 \times 28$ |
| Max Pool. | $3 \times 14 \times 14$ |
| Conv. | $9 \times 14 \times 14$ |
| Max Pool. | $9 \times 7 \times 7$ |
| Flattening | $441$ |
| Linear | $66$ |
| Linear | $10$ |

**CIFAR-10**

| Operation | Output Shape |
|---|---|
| *input* | $3 \times 32 \times 32$ |
| Conv. | $6 \times 28 \times 28$ |
| Conv. | $9 \times 28 \times 28$ |
| Max Pool. | $9 \times 14 \times 14$ |
| Conv. | $12 \times 14 \times 14$ |
| Conv. | $15 \times 14 \times 14$ |
| Max Pool. | $15 \times 7 \times 7$ |
| Conv. | $18 \times 7 \times 7$ |
| Conv. | $21 \times 7 \times 7$ |
| Max Pool. | $21 \times 4 \times 4$ |
| Flattening | $336$ |
| Linear | $58$ |
| Linear | $10$ |

Table 4.1: Neural Network Architectures

the number of training data unchanged. All networks' parameters are initialized using the same manually set seed to ensure both an even starting point and reproducibility. For all settings, the target for each presented image is chosen randomly among the nine incorrect classes.

For Curriculum Adversarial Training, for both datasets MNIST and CIFAR-10, the step size $\alpha$ is chosen to be 0.05 for LOTS and 0.025 for PGD and the overall maximum perturbation $\epsilon_{max}$ is set to 0.3 for both attacks. Then the epoch-specific maximum perturbation $\epsilon$ at each epoch $t$ out of a total number of epochs $T$ is set to be $\epsilon = \epsilon_{max} \cdot t/(T-1)$ and the number of steps $k$ are determined to be equal to the index of the current epoch. Thus, the adversarial examples generated exhibit increasing perturbation over the epochs, due to an increasing number of steps and maximum perturbation.

For Dynamic Adversarial Training, for both datasets MNIST and CIFAR-10, $\epsilon$ is set to 0.3, the step size to 0.15 for LOTS and 0.075 for PGD and the maximum FOSC value $c_{max}$ is set to 0.5 in both cases. The control epoch $T'$ out of a total of $T$ epochs is chosen to be roughly the epoch at 80% of training, specifically $\lfloor 0.8 \cdot T \rfloor$, that is, the 6-th epoch for MNIST and the 12-th epoch for CIFAR-10. Then, the FOSC threshold $c_t$ that the adversarial examples are supposed to reach at each epoch $t$ — restricted to a maximum of 10 iterations — is given by $c_t = max(c_{max} - t \cdot c_{max}/T')$.

## 4.2 Results

### 4.2.1 Attacking Unsecured Models

The accuracies on the original test set are displayed as percentages in table 4.2. The columns indicate the adversarial training method of the model, base denoting the unsecured, regularly trained model, 1 denoting the single-step attack, 10 denoting the 10-step attack, CAT denoting Curriculum Adversarial Training and DAT denoting Dynamic Adversarial Training. These notations are also valid for tables 4.3, 4.4, 4.5 and 4.6. The MNIST models are relatively stable in their accuracies on the test set, varying no more than 1% and all being around 98%. For CIFAR-10 however, the test set accuracies vary greatly, while being as high as 71% for the base model, also dropping as low as 20% on the model adversarially trained using PGD-10.

| | | LOTS | | | | PGD | | | |
|---|---|---|---|---|---|---|---|---|---|
| | **base** | **1** | **10** | **CAT** | **DAT** | **1** | **10** | **CAT** | **DAT** |
| **MNIST** | 98.7 | 98.7 | 98.1 | 98.4 | 98.1 | 98.3 | 98 | 98.4 | 98.6 |
| **CIFAR-10** | 71.4 | 64.5 | 44.6 | 38.7 | 42.7 | 61.5 | 20.4 | 36.1 | 54.7 |

Table 4.2: Test Set Accuracies

| | | LOTS | | | PGD | | |
|---|---|---|---|---|---|---|---|
| | | **1** | **10** | **success** | **1** | **10** | **success** |
| **MNIST** | **Accuracy** | 59.8 | 4.2 | 2.7 | 11.3 | 5.1 | 0.4 |
| | **Attack Success Rate** | 15 | 75.6 | 76 | 23.3 | 82.7 | 94.6 |
| **CIFAR-10** | **Accuracy** | 16 | 2.6 | 0 | 10.7 | 0.6 | 0 |
| | **Attack Success Rate** | 15.1 | 79.4 | 99 | 11.5 | 94.9 | 99.7 |

Table 4.3: Accuracies and Attack Success Rates on Base Models

First of all, it is of interest how well the non-adversarially trained base models defend against the different attacks. To answer this question, adversarial examples are created on the base models using the four attacks LOTS-1, LOTS-10, PGD-1 and PGD-10, as well as attacks for both PGD and LOTS without a fixed number of iterations and without a maximum perturbation, but which instead stop perturbation for each image once it reaches its target class. The latter two use a maximum of 500 iterations and a step size of 0.01 for PGD and 0.02 for LOTS. For these two scenarios, only the examples from the test set are considered for which the unsecured model predicts the correct class. The two methods Curriculum Adversarial Training and Dynamic Adversarial training are not applied for attacking, as they are adversarial training methods. For each attack setting, the same target classes are used. This yielded the accuracies and attack success rates as listed in table 4.3. The column success denotes the scenario of perturbation until either the target class or the maximum of 500 iterations is reached.

As can be seen in table 4.3, for all attacks, the attack success rate significantly increases over the settings (single step, 10-step, perturbation-until-success). PGD under the constrained perturbation-until-success-setting manages to reach an attack success rate of above 94% in the case of MNIST and above 99% in the case of CIFAR-10. LOTS under the same setting manages to reach a 76% attack success rate on MNIST, but an attack success rate of 99% on CIFAR-10, matching PGD. For both datasets, four randomly chosen images from the test set alongside their corresponding adversarial examples are displayed in figures 4.3 and 4.4. The column denotes the attack, the row denotes the image, where the true and chosen target classes are displayed on the left. For each image, the predicted probabilities for the true and target classes are displayed, denoted with $p(C)$ and $p(T)$ respectively. Generally, for both datasets, the perturbations are clearly visible for both fixed-step settings, however much stronger so for PGD attacks as for LOTS attacks. Also, the perturbations of the adversarial examples generated by the single-step attacks are not always significantly less perceptible than the 10-step attacks, as the single-step attacks use much larger step sizes. For the original images, the displayed probabilities for the true classes are fairly large, reaching up to 1, and the probabilities for the target class are zero for all displayed examples, except for the cat image. These probabilities change significantly in the case of the adversarial examples, reaching as low as 0 for the true classes and as high as 1 for the target classes, even for fixed-step attacks.

For the until-success setting the perceptual similarities between the generated adversarial examples and their original counterparts are measured via the SSIM metric presented in section 3.1. The plots in figure 4.5 show, for each SSIM value, the share of generated adversarial examples ex-

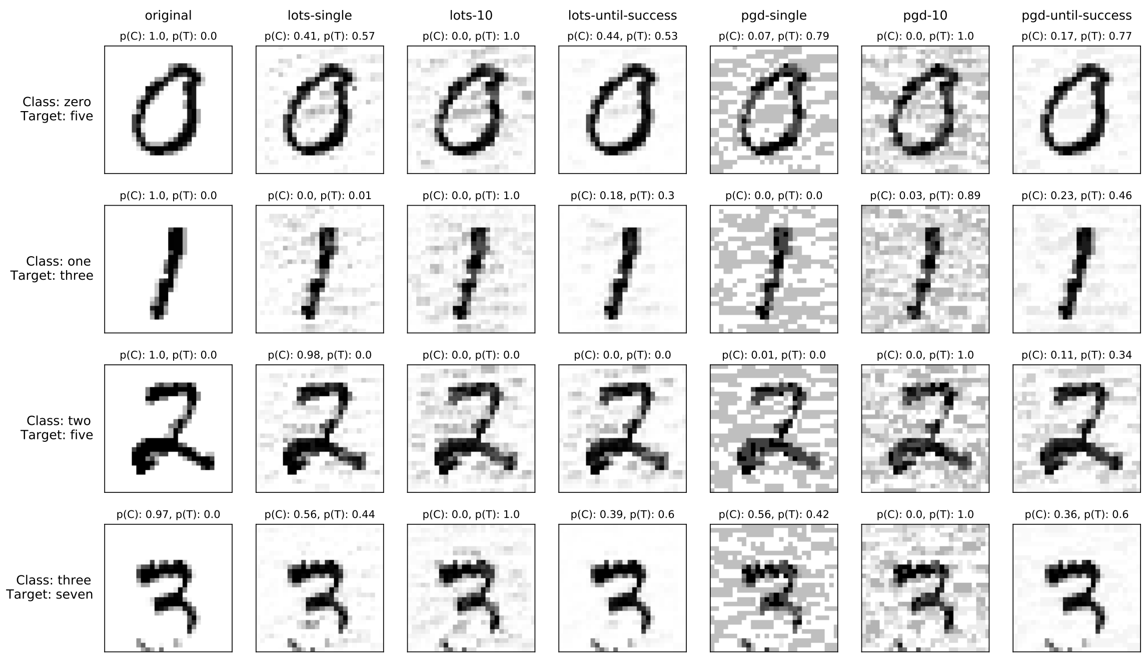


Figure 4.3: MNIST Adversarial Examples

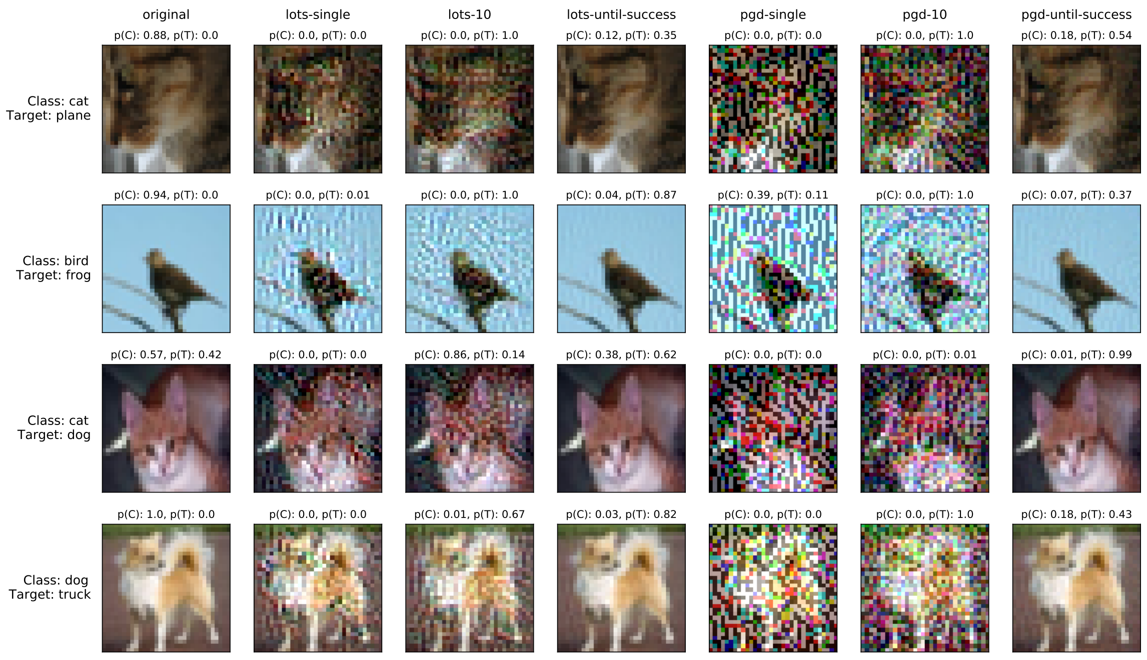
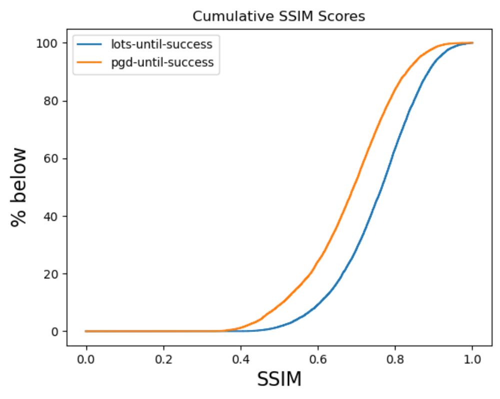
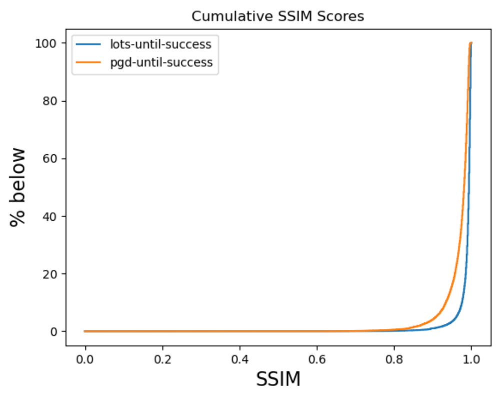


Figure 4.4: CIFAR-10 Adversarial Examples

hibiting a SSIM score not greater than that value. The created adversarial examples for CIFAR-10 yield much greater similarity scores than for MNIST, with about 90% of the generated adversarial examples exhibiting a SSIM value of 0.9 or greater. However, for MNIST, only about 5-10%

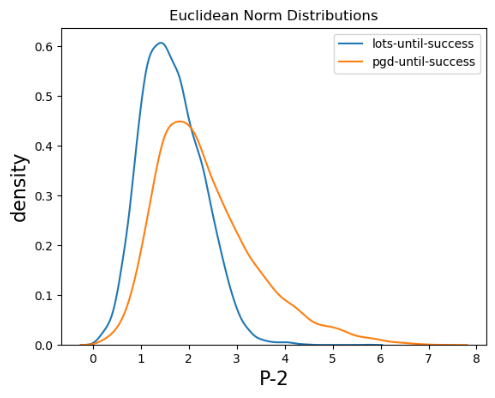


(b) MNIST

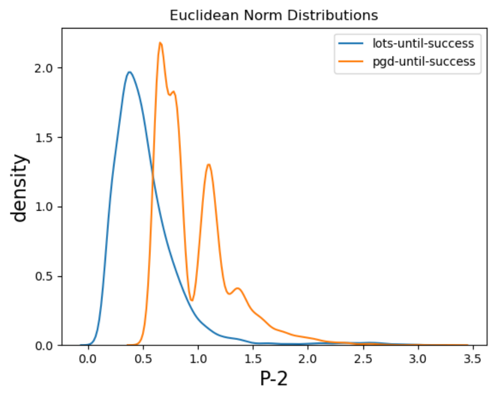


(c) CIFAR-10

Figure 4.5: Cumulative SSIM Scores



(b) MNIST



(c) CIFAR-10

Figure 4.6: Euclidean Norms Densities

of adversarial examples exhibit a SSIM of 0.9 or greater, and about 50% of them have a SSIM lower than 0.8. Surprisingly, PGD and LOTS yielded rather similar curves, even though LOTS in both cases exhibit slightly higher cumulative SSIM values. The kernel density estimate plots for the euclidean norms between the created adversarial examples with the until-success setting and their original counterparts are displayed in figure 4.6 as a reference. Again, adversarial examples created using LOTS in the grand scheme of things exhibit greater similarities when interpreting the inverse of the $L_2$ norms — which can be considered dissimilarity measures — as similarity measures.

### 4.2.2 Cross-Evaluation

Once the adversarially trained networks are fitted, they are cross-evaluated against each attack, that is, for both datasets MNIST and CIFAR-10, each attack setting LOTS-1, LOTS-10, PGD-1 and PGD-10 is applied to each of the nine models. To do so, only a subset of the test set is considered, specifically the set of images for which all models that are cross-evaluated against each other

| | | Accuracies | | | | Attack Success Rates | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | LOTS | | PGD | | LOTS | | PGD | |
| | | **1** | **10** | **1** | **10** | **1** | **10** | **1** | **10** |
| | **base** | 61.3 | 3.8 | 11.4 | 5.4 | 14.1 | 75 | 23.2 | 82.1 |
| **LOTS** | **1** | 96.4 | 55.2 | 79.3 | 8.7 | 2.2 | 33.6 | 7.3 | 83 |
| | **10** | 98.3 | 91.4 | 88.9 | 36.1 | 1.1 | 7.2 | 3.8 | 59.4 |
| | **CAT** | 94.8 | 54.1 | 76.7 | 13 | 2.7 | 32.3 | 7.3 | 74.8 |
| | **DAT** | 98.1 | 91.2 | 87.6 | 37 | 1.1 | 6.8 | 4.1 | 58.7 |
| **PGD** | **1** | 96.3 | 49.9 | 90.1 | 6.5 | 1.9 | 35.9 | 2 | 82.5 |
| | **10** | 98.8 | 94.8 | 98.2 | 96.5 | 0.5 | 3.5 | 0.4 | 2 |
| | **CAT** | 98.5 | 92.7 | 90 | 63 | 0.8 | 5.3 | 2.7 | 28.6 |
| | **DAT** | 94.5 | 60.2 | 73.5 | 16.1 | 2.7 | 27.8 | 7.9 | 68.8 |

Table 4.4: MNIST Cross-Evaluation Accuracies and Attack Success Rates

| | | Accuracies | | | | Attack Success Rates | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | LOTS | | PGD | | LOTS | | PGD | |
| | | **1** | **10** | **1** | **10** | **1** | **10** | **1** | **10** |
| | **base** | 21.2 | 5 | 2.3 | 0.9 | 13.8 | 76.3 | 13.3 | 92.8 |
| **LOTS** | **1** | 76.8 | 0.5 | 51 | 0 | 9 | 94.6 | 13.1 | 99.8 |
| | **10** | 94.4 | 56.4 | 79 | 0.2 | 1.1 | 34.5 | 7.5 | 99.6 |
| | **CAT** | 83.8 | 37.5 | 38.2 | 4.5 | 9.3 | 54.9 | 43.1 | 95.5 |
| | **DAT** | 80.1 | 29.1 | 31.8 | 5.2 | 9.7 | 57.8 | 42.2 | 94.4 |
| **PGD** | **1** | 77.9 | 0.7 | 89.6 | 0.7 | 7.5 | 91.4 | 0.2 | 96.6 |
| | **10** | 92.6 | 78.8 | 80.1 | 55.3 | 0.7 | 9 | 2.7 | 35.4 |
| | **CAT** | 95.7 | 89.8 | 82.8 | 77.4 | 2.5 | 4.3 | 7 | 18.5 |
| | **DAT** | 86.9 | 7.9 | 71.8 | 1.4 | 7.2 | 79.2 | 7 | 98.2 |

Table 4.5: CIFAR-10 Cross-Evaluation Accuracies and Attack Success Rates

predict the correct labels on the test set. This is done as generating adversarial examples from images for which the classes are not correctly predicted in the first place is not meaningful. For MNIST, there are 95.65% of the images for which all models predict the true class correctly. For CIFAR-10, this corresponded to only 4.43%, which makes the statistics less significant, as the 4.43% corresponds to less than 500 images. The models are evaluated with respect to accuracy, meaning the fraction of correctly classified adversarial examples. As the accuracies are mainly relevant for the defender and a significant accuracy drop does not imply an increase in the attack success rate, the metric of interest to the attacker, the attack success rates are also computed. Tables 4.4 and 4.5 display these results. The rows indicate the training method, the columns indicate the attack method.

There are several noteworthy observations to be made from the above results. For both MNIST and CIFAR-10, adversarial training yields higher accuracies compared to non-adversarial training in almost all cases. PGD attacks tend to lead to higher attack success rates and lower accuracies than LOTS attacks. Similarly, 10-step attacks also tend to lead to higher attack success rates and lower accuracies than single-step attacks. On MNIST, PGD attacks reach up to 82% attack success rates, LOTS attacks only up to 74%. On CIFAR-10, PGD attacks reach up to 99% attack success rates, LOTS attacks up to 94%. Models trained with PGD, with respect to accuracy, manage to defend better against LOTS adversarial examples than models trained with LOTS.

| | | MNIST | | | | CIFAR-10 | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | LOTS | | PGD | | LOTS | | PGD | |
| | | **1** | **10** | **1** | **10** | **1** | **10** | **1** | **10** |
| | **base** | 39.9 | 41.3 | 30.5 | 11.5 | 28.9 | 22 | 22.7 | 20.7 |
| **LOTS** | **1** | 100 | 62.1 | 19.7 | 6.5 | 100 | 32.3 | 47.2 | 20.5 |
| | **10** | 36.4 | 100 | 14.8 | 1.4 | 20.7 | 100 | 25.8 | 27.9 |
| | **CAT** | 45.6 | 54.5 | 26.9 | 15.2 | 18.6 | 27.9 | 20.9 | 21.1 |
| | **DAT** | 36.8 | 58.5 | 17.9 | 3.2 | 19.1 | 24.7 | 25.2 | 24.6 |
| **PGD** | **1** | 30.4 | 33.8 | 100 | 27.7 | 29.5 | 32.1 | 100 | 21.1 |
| | **10** | 23.7 | 22.6 | 13.9 | 100 | 20 | 19.1 | 21.5 | 100 |
| | **CAT** | 37.8 | 42.8 | 17.9 | 7.8 | 20 | 24.6 | 20.3 | 26.1 |
| | **DAT** | 53 | 55.7 | 28.7 | 9.7 | 23.5 | 28.8 | 27 | 17.2 |

Table 4.6: Transferability Rates

### 4.2.3 Transferability

Also of great interest is the transferability of an attack, meaning using each attack setting only once to create adversarial examples and then checking how those adversarial exampples transfer to other models. The transferability rate is defined as the share of successful adversarial examples that are also successfully fooling a different model. In this setting, the sets of adversarial examples are each created with the same attack setting that is also used to create the adversarial examples that are used to train the corresponding model. Only the examples from the test set for which the model at hand predicts the correct classes are considered. Then, only the adversarial examples for which the target classes are reached are considered to evaluate the transferability. This yielded the transferability rates displayed in table 4.6.

A great fraction of the attacks transfer better to adversarially trained models as opposed to the base models. Generally, 10-step attacks transfer better than single-step attacks. For MNIST, the highest transferability of 62% is from LOTS-10 adversarial examples on the model trained with LOTS-1, and the lowest transferability of 1% is from PGD-10 adversarial examples on the model trained with LOTS-10. For CIFAR-10, the highest transferability of 29% is from LOTS-1 adversarial examples on the model dynamically trained with LOTS, and the worst transferability of 17% is from PGD-10 adversarial examples on the model trained dynamically with PGD.

## 4.3 Discussion

### 4.3.1 PGD vs. LOTS

It is worth stressing the main differences of the two attack types, LOTS and PGD, namely the target and the gradient treatment at perturbation time. If LOTS in case of using the last layer as the target layer were to apply softmax on the logits first and use one-hot vectors representing the target classes (as opposed to templates formed from the logits of examples of the target class), then the two attack methods are equivalent with respect to the target handling. Further, PGD uses the sign of the gradients, effectively values in the set {-1, 1} as the quotient of perturbation and step size, where LOTS uses max-scaled gradients, effectively values in the range [-1, 1]. Thus the resulting adversarial examples created using PGD typically exhibit significantly stronger perturbations, compared to LOTS. This was also the motivation to use step sizes for LOTS attacks that are twice as high as for PGD attacks. Therefore, PGD typically leads to stronger attacks, at least with respect to low accuracy and high attack success rates; not however with respect to low perceptibility of the perturbations, as was shown in figures 4.3 and 4.4. The latter difference is likely responsible for a large effect on the differences in the results. This raises the question of whether or not running the experiments for the two examined attack methods, PGD and LOTS — with, apart from the attack methods and step sizes, equal attack parameters, such as number of steps — allowed for reasonable comparisons between them in the first place.

### 4.3.2 Shortcomings

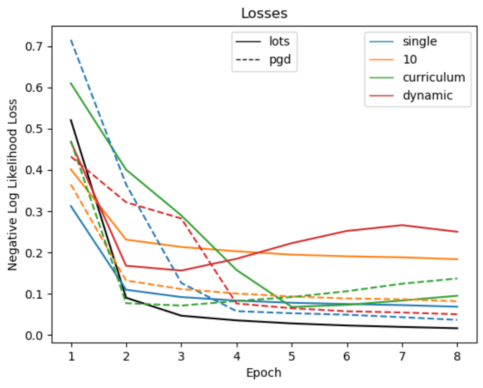
There are some shortcomings of this work. No particular emphasis was put on ensuring that the networks converge well during training, in particular, no regularization — as for example using a validation set during training — was performed. This may have the effect that findings are not transferable to settings of well-converged models, as under- or overfitted models may exhibit a different attacking difficulty. The training losses are displayed in figure 4.7. The displayed losses are defined to be the mean over all the batches for each epoch. For the fixed-step adversarial training settings, the displayed loss corresponds to the mean of the loss from the original and adversarial training images. The black line corresponds to the loss of the regularly trained base model. The comparison of the training losses may not be inherently meaningful, as the performance on the test set, not on the training set, is primarily of interest and further, the number of training data is not fixed over the different training methods. The increasing training losses for the models trained with Curriculum Adversarial Training may be an indicator for an inadequate setup, even though a greater loss would be expected for stronger adversarial examples. A setting for Curriculum Adversarial Training where the original and adversarial training losses are weighted with some coefficients — as opposed to only training on the adversarial images — may would have been a better choice.

As the FOSC metric was developed for untargeted attacks and thus had to be adjusted for targeted attacks, however without explicitly derived it, it may not have been entirely adequate. Especially, the FOSC values did not significantly decrease over the epochs as expected, exhibiting only a slow decrease in the case of MNIST and LOTS, stagnation in the case of CIFAR-10 and LOTS and even an increase in the case of PGD Dynamic Adversarial Training. The plots of the FOSC values of the adversarial images at each epoch during Dynamic Adversarial Training are displayed in figure 4.8.

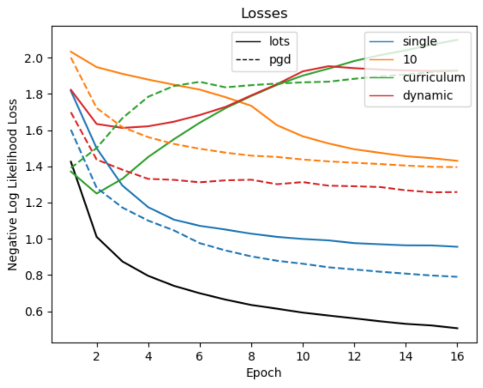
There may be improvements that could have been made with respect to the LOTS target generation. Using the logits from all examples as templates to form targets as opposed to using only the logits from examples for which the model predicted correctly may not have been optimal. That is the case since such templates clearly do a worse job in forming a target that enables gradients indicating good perturbations to reach the target class. An improvement thus may would have been to only use the logits from correctly predicted examples for generating targets during training. An additional improvement that could have been made in conjunction with the one just mentioned is to only start adversarial training after a certain number of epochs, as adversarial examples early on in the fitting procedure may hinder generalization. Further, using the training data to generate the LOTS targets used for evaluation leaks information from the training data, which can be considered a questionable approach.

### 4.3.3 ImageNet

As the MNIST and CIFAR-10 images are of very low resolutions, results obtained from those datasets may not necessarily transfer to scenarios involving larger images. Therefore, the code was extended to enable also running the experiments on a subset of the ImageNet dataset, specifically of subset ILSVRC from the year 2012, which contains images of greater resolutions and

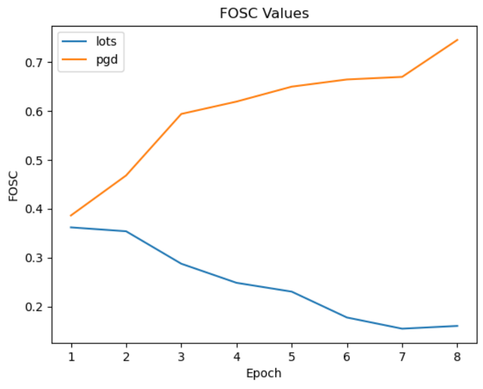


(b) MNIST

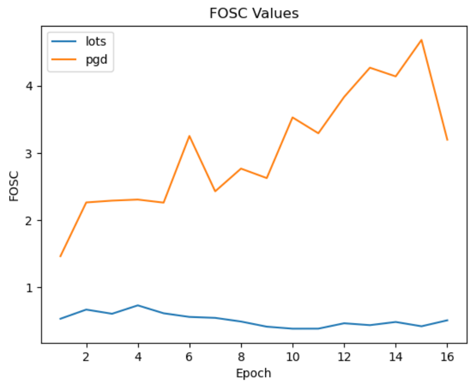


(c) CIFAR-10

Figure 4.7: Training Losses



(b) MNIST



(c) CIFAR-10

Figure 4.8: FOSC Values

quality. The ILSVRC training dataset consists of roughly one million images in 1’000 classes. As these images are not uniform with respect to their shapes and number of color channels, they were filtered such that only RBG images were considered, meaning, grayscale images were filtered out, as well as, at preprocessing time, the images were first resized such that the smaller dimension equals 224 pixels and then center cropped, yielding images of 224 by 224 pixels. This was done mainly for practicality reasons, as firstly, this on average reduces the image size and secondly, it allows for neater tensor operations, specifically batch operations, in both cases leading to reduced computational cost. Further, as the class domain is very large, only a subset of classes was considered. The optimizer used for training on the ImageNet subset used the same hyperparameters as the case for MNIST and CIFAR-10, however a larger batch size of 64 and a larger number of epochs of 32. The model was chosen to be the ResNet-18 (He et al., 2016). The last fully-connected linear layer of the original network architecture was replaced by a linear layer with a number of output features matching the number of selected classes. Unfortunately, however, as the ImageNet dataset was only tackled later on in the work and the runtimes for training the models were unexpectedly high, the ImageNet experiments are not included in chapter 4.2.

### 4.3.4 Conclusion

The goal of this work was to investigate to what extent adversarial training increases the robustness of neural networks against adversarial examples created using LOTS. It was shown that, compared to unsecured, regularly trained networks, adversarially trained networks are clearly more robust towards LOTS adversarial examples. It was also shown that in a graybox-setting of unknown defense method but known network architecture, adversarial examples do transfer reasonably well. However, since the studied settings are limited with respect to variability in the defenses and the used datasets are far from typical images occurring in real-world scenarios, it is hard to make a valid inference from the obtained results on the general capabilities of adversarial training to defend against adversarial images created using LOTS.

There are various directions in which this work can be extended in future research. This work was limited to constraining perturbations with respect to $L_\infty$. It is not clear what effects using other $L_p$ norms such as $L_2$ or even measures such SSIM instead might have. Further, this work only looked at the last network layer as the target layer for LOTS. However, LOTS allows the target layer to be any layer. What is the influence of the target layer on the attack success rate? The hypothesis that using earlier layers as target layers are worse suited for reaching a certain attack success rate under otherwise equal conditions may sound plausible, but would have to be tested. Also, are there more sophisticated ways to form target activations for LOTS as opposed to simply using a moving average of templates as was done in this work? Further, it could be investigated how well other attacks that are significantly more different to LOTS than PGD are able to attack models adversarially trained using LOTS or how well LOTS is able to attack models adversarially trained on them. However, with respect to the grand scheme of robustness towards adversarial examples, studying various specific and attack and defense settings is likely less promising than more fundamental research.

# List of Figures

# List of Tables

# List of Algorithms

# Bibliography

Athalye, A., Carlini, N., and Wagner, D. (2018). Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*, ICML '09.

Cai, Q.-Z., Liu, C., and Song, D. (2018). Curriculum adversarial training. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI)*.

Carlini, N. and Wagner, D. A. (2016). Defensive distillation is not robust to adversarial examples. *arXiv*, abs/1607.04311.

Carlini, N. and Wagner, D. A. (2017). Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy (SP)*.

Elman, J. L. (1993). Learning and development in neural networks: the importance of starting small. In *Cognition*, volume 48.

Geirhos, R., Rubisch, P., Michaelis, C., Bethge, M., Wichmann, F. A., and Brendel, W. (2019). Imagenet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. In *International Conference on Learning Representations (ICLR)*.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 27.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2015). Explaining and harnessing adversarial examples. In *International Conference on Learning Representations (ICLR)*.

Graese, A., Rozsa, A., and Boult, T. E. (2016). Assessing threat of adversarial examples on deep neural networks. In *15th IEEE International Conference on Machine Learning and Applications (ICMLA)*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ilyas, A., Santurkar, S., Tsipras, D., Engstrom, L., Tran, B., and Madry, A. (2019). Adversarial examples are not bugs, they are features. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, volume 37 of *Proceedings of Machine Learning Research*.

Kurakin, A., Goodfellow, I., and Bengio, S. (2017a). Adversarial examples in the physical world. In *ICLR Workshop*.

Kurakin, A., Goodfellow, I., and Bengio, S. (2017b). Adversarial machine learning at scale. In *International Conference on Learning Representations (ICLR)*.

Liu, Y., Chen, X., Liu, C., and Song, D. (2017). Delving into transferable adversarial examples and black-box attacks. In *International Conference on Learning Representations (ICLR)*.

Oh, S. J., Augustin, M., Fritz, M., and Schiele, B. (2018). Towards reverse-engineering black-box neural networks. In *International Conference on Learning Representations (ICLR)*.

Papernot, N., McDaniel, P. D., Wu, X., Jha, S., and Swami, A. (2016). Distillation as a defense to adversarial perturbations against deep neural networks. In *IEEE Symposium on Security and Privacy (SP)*.

Rozsa, A., Günther, M., and Boult, T. E. (2016a). Are accuracy and robustness correlated? In *15th IEEE International Conference on Machine Learning and Applications (ICMLA)*.

Rozsa, A., Günther, M., and Boult, T. E. (2017). LOTS about attacking deep features. In *IEEE International Joint Conference on Biometrics (IJCB)*.

Rozsa, A., Rudd, E. M., and Boult, T. E. (2016b). Adversarial diversity and hard positive generation. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*.

S., V. B., Mopuri, K. R., and Babu, R. V. (2018). Gray-box adversarial training. *ArXiv*, abs/1808.01753.

Scheirer, W., Rocha, A., Sapkota, A., and Boult, T. (2013). Toward open set recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 35(7).

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2014). Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR)*.

Tramèr, F., Kurakin, A., Papernot, N., Goodfellow, I., Boneh, D., and McDaniel, P. (2018). Ensemble adversarial training: Attacks and defenses. In *International Conference on Learning Representations (ICLR)*.

Tsipras, D., Santurkar, S., Engstrom, L., Turner, A., and Madry, A. (2019). Robustness may be at odds with accuracy. In *International Conference on Learning Representations (ICLR)*.

Wang, Y., Ma, X., Bailey, J., Yi, J., Zhou, B., and Gu, Q. (2019). On the convergence and robustness of adversarial training. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*.

Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. (2004). Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4).

Wiyatno, R. R., Xu, A., Dia, O., and de Berker, A. (2019). Adversarial examples in modern machine learning: A review. *ArXiv*, abs/1911.05268.

Xie, C., Wang, J., Zhang, Z., Ren, Z., and Yuille, A. (2018). Mitigating adversarial effects through randomization. In *International Conference on Learning Representations (ICLR)*.

Xu, W., Evans, D., and Qi, Y. (2018). Feature squeezing: Detecting adversarial examples in deep neural networks. In *25th Annual Network and Distributed System Security Symposium (NDSS)*.

Zintgraf, L. M., Cohen, T. S., and Welling, M. (2017). A new method to visualize deep neural networks. In *International Conference on Learning Representations (ICLR)*.