Adversarial Examples and Robust Training of Deep Neural Networks for Image Classification

Bachelorarbeit

für die Prüfung zum

Bachelor of Science

des Studiengangs Informationstechnik
an der Dualen Hochschule Baden-Württemberg Mannheim
von

Fabrice von der Lehr

7. September 2021

Bearbeitungszeitraum:	14.06.2021 – 06.09.2021
Matrikelnummer, Kurs:	6208253, TINF18IT1
Ausbildungsfirma:	Deutsches Zentrum für Luft- und Raumfahrt e.V. in der Helmholtz-Gemeinschaft
Abteilung:	Institut für Softwaretechnologie – HPC
Betreuer der Ausbildungsfirma:	Dr. Martin Siggel
Gutachter der Dualen Hochschule:	Prof. Dr. Karl Stroetmann

EIGENSTÄNDIGKEITSERKLÄRUNG

Gemäß §5 (4) der Studien- und Prüfungsordnung DHBW Technik vom 27. Juli 2020.
Ich versichere hiermit, dass ich meine Bachelorarbeit zu dem Thema

Adversarial Examples and Robust Training of Deep Neural Networks for Image Classification

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.*

Köln, den 7. September 2021

*falls beide Fassungen gefordert sind

ABSTRACT

Deep neural networks (DNNs) have become a powerful tool for image classification tasks in recent years, being nowadays also relevant for safety-critical applications like autonomous driving. Despite being highly accurate even for unknown images, the existence of so-called "adversarial examples" nevertheless calls the robustness of DNNs into question: These are slightly, but purposefully perturbed versions of natural images, being only barely distinguishable from their unperturbed originals, but causing the DNN to misclassify them.

In the scope of this work, two white-box attacks (Fast Gradient Sign Method, Projected Gradient Descent) and a black-box attack (Boundary Attack) were implemented to create the adversarial examples on the basis of images from the CIFAR-10 and the GTSRB datasets. The trained DNNs, being based on the PreAct-ResNet-50 architecture, were subsequently evaluated concerning their robustness against both adversarial and random perturbations. Furthermore, two variants of adversarial training (using the Fast Gradient Sign Method and the Stable Single Step algorithm, respectively) were implemented to analyze, in how far such an adaption of the training process influences the robustness and general accuracy of DNNs. Last, the loss landscapes of the differently trained DNNs were investigated qualitatively.

The results show that the susceptibility to adversarial examples is highly data dependent, with images from CIFAR-10 generally exhibiting a higher risk than those from the GTSRB dataset. By contrast, random perturbations comparatively rarely led to misclassifications, regardless of the dataset considered. Moreover, Stable Single Step-based adversarial training has proven to increase the robustness against adversarial examples to a limited extent, but also slightly lower the accuracy for natural images. In general, however, adversarial training led to insufficient robustness enhancements, for which substantial overfitting of the trained DNNs was identified as the main reason.

ZUSAMMENFASSUNG

Tiefe neuronale Netze haben sich in den letzten Jahren zu einem mächtigen Werkzeug für die Klassifizierung von Bildern entwickelt, sodass sie heutzutage auch für sicherheitskritische Anwendungen wie das autonome Fahren infrage kommen. Obwohl sie auch bei unbekannten Bildern eine hohe Genauigkeit aufweisen, führt die Existenz sogenannter „Adversarial Examples“ doch zu Zweifeln an der Robustheit der neuronalen Netze. Denn solche Adversarial Examples sind nur leicht, auf spezielle Art und Weise verrauschte Versionen von tatsächlichen Bildern, die praktisch nicht von den Originalen zu unterscheiden sind, aber zu einer Fehlklassifizierung durch das DNN führen.

Im Rahmen dieser Arbeit wurden zwei White-Box-Attacken (Fast Gradient Sign Method, Projected Gradient Descent) und eine Black-Box-Attacke (Boundary Attack) implementiert, um Adversarial Examples auf der Grundlage von Bildern aus den Datensätzen CIFAR-10 und GTSRB zu erzeugen. Die trainierten neuronale Netze, die auf der PreAct-ResNet-50 Architektur basieren, wurden anschließend hinsichtlich ihrer Robustheit sowohl gegenüber zufälligem Rauschen als auch dem speziellen Rauschen von Adversarial Examples untersucht. Darüber hinaus wurden zwei Varianten des „Adversarial Trainings“ (auf Basis der Fast Gradient Sign Method bzw. des Stable Single Step Algorithmus’) implementiert, um zu analysieren, inwieweit eine derartige Anpassung des Trainingsprozesses die Robustheit und die allgemeine Genauigkeit der neuronalen Netze beeinflusst. Zuletzt wurde die Form der Kostenfunktion für die unterschiedlich trainierten neuronalen Netze qualitativ untersucht.

Die Ergebnisse zeigen, dass die Anfälligkeit für Adversarial Examples stark von den zugrundeliegenden Daten abhängt; Bilder aus dem CIFAR-10 Datensatz wiesen im Allgemeinen ein höheres Risiko auf als solche aus dem GTSRB Datensatz. Dagegen führte zufälliges Rauschen vergleichsweise selten zu Fehlklassifizierungen, unabhängig von dem betrachteten Datensatz. Ferner hat sich herausgestellt, dass Adversarial Training unter Verwendung des Stable Single Step Algorithmus’ die Robustheit gegenüber Adversarial Examples in einem begrenzen Rahmen erhöhen kann, aber zugleich in einem leichten Genauigkeitsverlust gegenüber natürlichen Bildern resultiert. Im Allgemeinen hatte Adversarial Training jedoch nur eine unzureichende Steigerung der Robustheit zur Folge. Als Hauptursache hierfür wurde eine erhebliche Überanpassung der trainierten neuronalen Netze identifiziert.

CONTENTS

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS

Abbreviation	Expansion	First appearance
ABC	Automated border control	p. 1
Adam	Adaptive moment estimation	p. 24
AdamW	Adaptive moment estimation with decoupled weight decay	p. 24
AI	Artificial intelligence	p. 1
CIFAR	Canadian Institute for Advanced Research	p. 17
DLR	German Aerospace Center	p. 46
DNN	Deep neural network	p. 1
FGM	Fast Gradient Method	p. 32
FGSM	Fast Gradient Sign Method	p. 28
GTSRB	German Traffic Sign Recognition Benchmark	p. 17
iid	Independent and identically distributed	p. 33
LeNet	LeCun Network	p. 9
MNIST	Modified National Institute of Standards and Technology	p. 2
PGD	Projected Gradient Descent	p. 29
PreAct	Pre-activation	p. 18
ReLU	Rectified linear unit	p. 8
ResNet	Residual network	p. 18
SGD	Stochastic gradient descent	p. 12
SSS	Stable Single Step	p. 41
StGB	Strafgesetzbuch	p. 89

Abbreviation	Expansion	First appearance
TanH	Hyperbolic tangens	p. 9

LIST OF SYMBOLS

Symbol	Description	First appearance
$\hat{f}$	Neural network model function (classifier, yields classification certainties)	p. 7
$\mathbf{x}$	Input data	p. 7
$\boldsymbol{\theta}$	Neural network parameters	p. 7
$\hat{\mathbf{y}}$	Neural network prediction (one-hot encoded)	p. 7
$\mathbf{y}$	Ground truth data (one-hot encoded)	p. 7
$\mathcal{Y}$	Set of one-hot encoded output data vectors	p. 7
$\mathcal{C}$	Set of classes	p. 7
c	Ground truth data (class label)	p. 7
$\hat{c}$	Neural network prediction (class label)	p. 7
$\hat{f}_c$	Neural network model function (classifier, yields class affliation)	p. 7
ρ_{data}	Data distribution of images belonging to the considered classification task	p. 9
J	Loss function (cross entropy loss)	p. 10
$\hat{\rho}_{\text{data}}$	Training dataset	p. 10
α	L_2 regularization strength	p. 10
η	Learning rate	p. 11
d	Dissimilarity metric	p. 12
$\overline{\mathcal{B}}$	Closed ball in a metric space	p. 12
$\mathcal{B}$	Open ball in a metric space	p. 12

Symbol	Description	First appearance
$\mathcal{S}$	Sphere in a metric space	p. 12
Π	Euclidean projection function	p. 12
$\ell_p^{(n)}$	n-dimensional ℓ_p space	p. 13
d_p	p-metric	p. 13
$\mathcal{X}_p$	Input space, based on the p-metric	p. 14
$\mathcal{X}$	Input space, based on the ∞-metric	p. 14
ϵ	Maximum magnitude of perturbation (adversarial attack)	p. 14
$\mathbf{x}'$	Perturbed input data	p. 14
s	Integer that determines the count of convolutional kernels for a residual block (ResNet architecture)	p. 21
β_1	Decay rate for exponential moving averages of the first moment of the loss function's gradient w.r.t. the neural network parameters (Adam optimization algorithm)	p. 24
β_2	Decay rate for exponential moving averages of the second raw moment of the loss function's gradient w.r.t. the neural network parameters (Adam optimization algorithm)	p. 24
α^*	Weight decay rate	p. 26
η_{max}	Initial learning rate for cosine annealing	p. 27
η_{min}	Minimum learning rate for cosine annealing	p. 27
T_{max}	Number of batches per cosine annealing cycle, decreased by one	p. 27
$\mathbf{v}$	Adversarial direction in which a natural image is perturbed	p. 29

Symbol	Description	First appearance
$\mathcal{V}^*$	Set of valid perturbation vectors for FGSM/PGD	p. 31
$\mathcal{X}^*$	Set of valid perturbed data for FGSM/PGD	p. 31
η'	PGD step size	p. 33
$\mathbf{x'}^{(\mathbf{i})}$	Perturbed input data after the i-th step of PGD/the i-th step of the Boundary Attack/after using the i-th smalles magnitude of perturbation in SSS-based adversarial training	p. 33
$\mathbf{x'}^{(\mathbf{0})}$	Initialization data for PGD/SSS/the Boundary Attack	p. 33
N	Number of iterations during the Boundary Attack	p. 35
M	Number of computed candidate samples during an iteration of the Boundary Attack	p. 35
$\mathbf{o}^{(\mathbf{i}),(\mathbf{j})}$	j-th orthogonal sample candidate during the i-th iteration of the Boundary Attack	p. 35
$\sigma_1^{(i)}$	Parameter that determines the size of the orthogonal substep during the i-th iteration of the Boundary Attack	p. 35
$\mathbf{o}^{*(\mathbf{i}),(\mathbf{j})}$	j-th orthogonal sample candidate during the i-th iteration of the Boundary Attack (projected onto the input space)	p. 35
$\mathbf{q}^{(\mathbf{i}),(\mathbf{j})}$	j-th minimization sample candidate during the i-th iteration of the Boundary Attack	p. 36

Symbol	Description	First appearance
$\sigma_2^{(i)}$	Parameter that determines the size of the minimization substep during the i-th iteration of the Boundary Attack	p. 36
$\psi_1^{(i)}$	Adversarial ratio w.r.t. the projected orthogonal sample candidates during the i-th iteration of the Boundary Attack	p. 38
$\psi_2^{(i)}$	Adversarial ratio w.r.t. the minimization sample candidates during the i-th iteration of the Boundary Attack	p. 38
ϵ^*	Maximum magnitude of perturbation (adversarial training)	p. 41
ϵ_i^*	i-th smallest magnitude of perturbation (SSS-based adversarial training)	p. 43
L	Level of discretization/number of different magnitudes of perturbation in SSS-based adversarial training	p. 42
$\mathbf{u}$	Random direction in which a natural image is perturbed	p. 42
λ	Parameter that determines the magnitude of random perturbation for loss landscape investigation	p. 50
μ	Parameter that determines the magnitude of adversarial perturbation for loss landscape investigation	p. 50

1 INTRODUCTION

As a motivation to this work, imagine a future world where processes are automated to a high extend. Imagine a future world, where computer systems are able to take over decision-making from humans in certain application fields by the use of artificial intelligence (AI).

In many cases, decision-making is at least partly based on visual information. For example, think of self-driving cars that use cameras to recognize traffic signs and road markings, or automated border control (ABC) systems that use them to verify the identity of travelers at airports. Moreover, optical diagnosing systems are conceivable in the healthcare sector, being able to detect a disease earlier than a doctor.

The applications mentioned have in common that they are all safety-critical, i.e., malfunction can cause severe economical or physical damage. While it would be utopian to expect flawlessness without exception, the used AI systems nevertheless have to be highly reliable. They must be robust against disturbance, both natural and intentional. Otherwise, in particular the missing of the latter opens the way to fraud, blackmail and harm in general.

In this work, deep learning as a promising AI approach for image classification tasks is investigated in view of its robustness against perturbations of the input data.

1.1. BACKGROUND

During recent years, deep neural networks (DNNs) have become one of the most effective and powerful tools in many machine learning use cases. Besides the domains of natural language processing [1, 2] and speech recognition [3, 4], great progress has particularly been made in image recognition [5–8]. DNNs nowadays achieve near-human accuracy in image classification tasks, facilitating the application in aforementioned fields like autonomous driving [9, 10] or medical diagnosing [11, 12] at least theoretically.

However, in 2014, Szegedy et al. [13] have indicated for the first time that the sole performance measured on test datasets and even during inference may be fallacious. Starting at natural images, they were able to create so-called *adversarial examples* – almost imperceptibly perturbed versions of the originals –, which subsequently caused DNNs to misclassify. Since then, the issue has been the subject of extensive research, with over 3000 scientific papers being published so far [14].

Unfortunately, the problem could not be solved yet. Although various methods were invented to increase the robustness against adversarial attacks, stronger attacking schemes reliably broke the defenses [15, 16].

Furthermore, the development of effective defensive approaches is complicated by the fact that adversarial examples tend to transfer across different DNNs with potentially different architectures [13, 17]. While many attacking schemes require comprehensive knowledge about the networks to be fooled, this also enables black-box attacks with only limited knowledge [18]. Moosavi-Dezfooli et al. [19] even showed that there exist so-called universal perturbations, which lead to misclassification when being applied to the majority of natural images sampled from a task-specific distribution.

Goodfellow, Shlens, and Szegedy [20] were the first to propose incorporating adversarial examples into the training dataset of a DNN. Provided a sufficiently strong method to create the adversarial examples, this *adversarial training* turned out to be more effective than other approaches that aim to enhance a network's robustness [16, 21]. The idea is the same as for the well-known data augmentation technique, where, e.g., affine transformations like rotation or scaling are applied to images from the training dataset, aiming to increase the generalization ability of the network [22].

Last, in contrast to adversarial training, which can only provide empirical robustness, there exist some certification techniques than can guarantee robustness under certain conditions [23–25]. However, these are computationally expensive and only barely scale to large state-of-the-art DNN architectures as well as high-dimensional data. In addition, the conditions under which adversarial robustness can be guaranteed are relatively strict, i.e., there are many adversarial examples that do not satisfy the conditions but are still imperceptible to humans [26].

1.2. PROBLEM STATEMENT

As explained at the beginning, the progressive dissemination of DNNs and their potential usage in safety-critical application fields increasingly require solutions for the lack of robustness against adversarial perturbations. The extensive research efforts without any groundbreaking success so far already indicate the high complexity of the problem. Looking at the circumstances under which DNNs are trained in the domain of image classification provides reasons for the apparent difficulty of robust training.

Consider a classification task of $28\,\text{px} \times 28\,\text{px}$ grayscale images that depict handwritten digits from zero to nine. A dataset that matches the described task is known as *Modified National Institute of Standards and Technology (MNIST)* [27] in the literature. Since every pixel is treated as a distinct dimension, this *input space*, which has to be separated into

the ten classes, has already 784 dimensions. The input space that is spanned by higher resoluted images with multiple color channels is even larger.

The training dataset that consists of image-class-pairs represents points in the input space whose class affiliation is known. In practice, the size of the dataset in relation to the dimensionality of the image data becomes problematic, as the volume of the input space grows exponentially to its number of dimensions. In view of the typical high-dimensionality of the input space, the available training data becomes sparse. For example, the aforementioned MNIST dataset contains "only" 60 000 images, which is literally nothing in a 784-dimensional space. Consequently, it is difficult to learn the actual structure of the input space w.r.t. the classification task, i.e., the ideal separation of the input space into classes, from the few given data. This issue is also known as *curse of dimensionality* and was first described by Bellman [28] in 1961.

Nevertheless, DNNs achieve remarkable performance even for new data that was not part of the training dataset, implying that the learned separation of the input space might be partly correct. By contrast, the existence of adversarial examples points out clear inadequacies and suffering from the curse of dimensionality. Adversarial training here seems to be an inherent method to increase the robustness of the DNN against adversarial perturbations, as it extends the training dataset specifically by the problematic samples.

In this work, we aim to work out the characteristics of adversarial examples on the one hand and evaluate the effectivity of adversarial training on the other hand. Therefore, we investigate both methods to create adversarial examples and to perform adversarial training.

We first train a DNN with the *PreAct-ResNet-50* architecture [5, 29] on the *CIFAR-10* dataset [30] and the *GTSRB* dataset [31], respectively. In both cases, we create adversarial examples and evaluate them for various generating algorithms, both qualitatively and quantitatively. In particular, we differentiate between a white-box attack that has full knowledge about the DNN as well as its parameters and a black-box-attack that has no further knowledge, but can solely pass images to the classifier and receive the classification result. Moreover, we study whether random perturbations that are comparable in magnitude to the adversarial ones lead to misclassification as well.

Subsequently, we train the DNNs afresh, this time adversarially, considering different variations of adversarial training. In the end, we compare the resulting DNN instances in terms of adversarial robustness on the one hand and accuracy w.r.t. natural images on the other hand. Last, we make some qualitative investigation of the loss landscape of natural images in the input space, both for standardly and adversarially trained DNNs.

1.3. MATHEMATICAL NOTATION

Unless otherwise indicated, mathematical notation in this work is defined as follows:

- *Scalars* are written in italics, either upper or lower case (e.g., α or N).
- *Vectors* are written in bold face and lower case (e.g., $\mathbf{a}$). In general, they are defined as *column vectors.*
- *Matrices* and *higher-rank tensors* are written in bold face and upper case (e.g., $\mathbf{A}$).
- For a vector $\mathbf{a} \in \mathbb{R}^n$, the notation a_i with $i \in \{1, 2, \ldots, n\}$ represents the *i-th component* of $\mathbf{a}$. Analogously, $A_{i,j}$ with $i \in \{1, 2, \ldots, n\}$, $j \in \{1, 2, \ldots, m\}$ denotes the component in the i-th row and the j-th column of the matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$. The same holds for higher-rank tensors.
- For a vector $\mathbf{a} \in \mathbb{R}^n$, $\sum \mathbf{a}$ is a short notation of $\sum_{i=1}^{n} a_i$.
- $\mathbf{0} \in \mathbb{R}^n$ and $\mathbf{1} \in \mathbb{R}^n$ denote the n-dimensional vectors that contain only zeros and ones, respectively.
- All functions applied to a vector/matrix/tensor are generally applied element-wise, unless otherwise specified.
- The notation $\delta_{i,j} = \begin{cases} 1, & \text{for } i = j \\ 0, & \text{for } i \neq j \end{cases}$ represents the *Kronecker delta.*
- Let $f : \mathbb{R}^n \to \mathbb{R}$ be a multivariate, scalar-valued function. Moreover, let such $f(x_1^{(1)}, x_2^{(1)}, \ldots, x_{n_1}^{(1)}, x_1^{(2)}, x_2^{(2)}, \ldots, x_{n_2}^{(2)}, \ldots, x_{n_m}^{(m)})$ with $\sum_{i=1}^{m} n_i = n$ be represented by $f^*(\mathbf{x^{(1)}}, \mathbf{x^{(2)}}, \ldots, \mathbf{x^{(m)}})$, where $\forall i \in \{1, 2, \ldots, m\} : \mathbf{x^{(i)}} = \left(x_1^{(i)}, x_2^{(i)}, \ldots, x_{n_i}^{(i)}\right)^\intercal$. Then, the *gradient of* f^* *w.r.t.* $\mathbf{x^{(i)}}$ is defined as $\nabla_{\mathbf{x}^{(i)}} f^* = \left(\frac{\partial f}{\partial x_1^{(i)}}, \frac{\partial f}{\partial x_2^{(i)}}, \ldots, \frac{\partial f}{\partial x_{n_i}^{(i)}}\right)$.
- Let X be a random variable with probability distribution ρ. Then, $\mathbb{E}_{X \sim \rho}(X)$ denotes the expected value of X and $\mathbb{V}_{X \sim \rho}(X)$ denotes the variance of X. In addition, given an n-dimensional vector $\mathbf{x} \in \mathbb{R}^n$, $\mathbb{E}(\mathbf{x}) = \frac{1}{n} \sum \mathbf{x}$ and $\mathbb{V}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mathbb{E}(\mathbf{x}))^2$ denote the mean and the variance of this vector's components, respectively.
- The uniform distribution over the closed interval $[a, b]$ with $a, b \in \mathbb{R}$ is denoted by $\mathcal{U}(a, b)$. Moreover, the Gaussian distribution with mean $\mu \in \mathbb{R}$ and variance $\sigma^2 \in \mathbb{R}$ is denoted by $\mathcal{N}(\mu, \sigma^2)$.

1.4. OUTLINE

The remainder of this work is structured as follows:

Chapter 2 – Preliminaries: Here we give a basic background to the topics that are covered in this thesis. That includes, e.g., how DNNs and their training work. Moreover, we provide definitions and introduce notations being important for the rest of this work.

Chapter 3 – Materials and Methods: In this chapter, we list the concrete materials and methods used in this work. This comprises the used datasets and DNN architecture as well as special algorithms for the training of DNNs, the creation of adversarial examples and the increase of DNNs' robustness. Furthermore, we specify the experiments that are conducted based on these materials and methods.

Chapter 4 – Results: This chapter contains the results of the previously specified experiments.

Chapter 5 – Discussion: Here, we evaluate the results presented before and discuss possible reasons why they turned out the way they did. In addition, outlooks on possible future research and developments are given.

Chapter 6 – Conclusion: In this chapter, the main results of this work are summed up.

Appendices A to D: In the appendices, we provide some formal proofs as well as supplementary material such as additional plots.

2 PRELIMINARIES

In the following, the preliminaries for this work are presented. First, in sections 2.1 to 2.2, a short overview on DNNs and how they are trained is given. Subsequently, in section 2.3, we introduce balls and spheres in metric spaces. Next, we introduce ℓ_p spaces and the corresponding p-metrics in section 2.4. In section 2.5, a general definition for adversarial examples is given. Last, we specify different threat models for adversarial attacks in section 2.6.

2.1. DEEP NEURAL NETWORKS

A *neural network* is a highly parameterized mathematical model $\hat{f}(\mathbf{x};\boldsymbol{\theta}) = \hat{\mathbf{y}}$, where $\mathbf{x} \in \mathbb{R}^n$ is an n-dimensional input, $\hat{\mathbf{y}} \in \mathbb{R}^m$ is an m-dimensional output and $\boldsymbol{\theta} \in \mathbb{R}^p$ is a p-dimensional parameter vector. They aim to approximate a certain function $f(\mathbf{x}) = \mathbf{y}$ with identical dimensionality of domain and range [32].

If the output data $\mathbf{y}$ is categorical, a neural network that is to approximate the corresponding function f is called a *classifier*. We assume that $\mathbf{y}$ is one-hot-encoded, i.e., $\mathbf{y} \in \mathcal{Y}$ with $\mathcal{Y} = \left\{\left(\delta_{1,j}, \delta_{2,j}, \ldots, \delta_{m,j}\right)^\intercal \mid j \in \{1, 2, \ldots, m\}\right\}$, where m denotes the number of classes to differentiate. In words, with a set of classes $C = \{1, 2, \ldots, m\}$, a specific class $c \in C$ is represented as the m-dimensional vector $\mathbf{y}$ that is zeroed except for the c-th component, which equals one. Then, the predicted class $\hat{c} \in C$ is the index of the maximum component of the model prediction $\hat{\mathbf{y}}$ [32]:

$$\hat{c} = \arg\max_{i \in C} \hat{y}_i. \tag{2.1}$$

Here, we define $\hat{f}_c : \mathbb{R}^n \to C$ as an extended version of the model function $\hat{f}$, which also performs eq. (2.1) after the result $\hat{\mathbf{y}}$ of $\hat{f}$ has been computed, yielding the predicted class affiliation of the input data $\mathbf{x}$.

Internally, neural networks are built up from layers, where the first one is the input layer, the last one is the output layer and those in between are called hidden layers. If a neural network incorporates two or more hidden layers, it is called *deep*. Moreover, *feedforward neural networks*, also known as *multilayer perceptrons*, are a special type of neural networks whose computational flow is straightforward, i.e. containing no cycles or feedback loops between layers [32].

In 1989, Cybenko [33] stated the *universal approximation theorem*, which says that neural networks with one hidden layer are able to represent any continuous function. One reason for this capability is their high number of parameters $\boldsymbol{\theta}$, which are gradually adjusted to form the best-possible approximation of f. Another one is their non-linearity: A layer typically consists of linear mathematical operations, being followed by a non-linear activation function, which only make neural networks capable of approximating non-linear functions. An exception is the input layer, which simply passes the input data to the following layer. The two mostly used types of layers are fully connected layers and convolutional layers [32].

Fully connected layers on the one hand multiply the vectorial input data $\mathbf{x}$ by a weight matrix $\mathbf{W}$, being followed by the addition of a bias vector $\mathbf{b}$ and resulting in the feature vector $\mathbf{z}$:

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}. \tag{2.2}$$

Convolutional layers on the other hand convolve the input data with a kernel to produce a so-called feature map. They are often used to process two-dimensional image data in the early stages of a DNN. Here, they can be interpreted as a kind of edge detectors. As two-dimensional image data is frequently composed by several channels (e.g. RGB), both the input data and the kernel have to be rank-three tensors. Moreover, a convolutional layer does usually not only incorporate a single kernel, but various kernels that have all the same size, but differ in values, producing various feature maps. In consequence, the set of kernels can be understood as a rank-four tensor [32].

Let the input data be a rank-three tensor $\mathbf{X} \in \mathbb{R}^{c_{in} \times h_{in} \times w_{in}}$ and let the set of kernels be a rank-four tensor $\mathbf{K} \in \mathbb{R}^{c_{out} \times c_{in} \times k_h \times k_w}$, where c_{in} denotes the channel count, h_{in} the height and w_{in} the width of input images. Furthermore, c_{out} here denotes the kernel count, synonymous to the number of output channels, k_h the kernel height and k_w the kernel width. Then, a convolutional layer performs channel-wise convolution between $\mathbf{X}$ and $\mathbf{K}$ and eventually adds a bias tensor $\mathbf{B} \in \mathbb{R}^{c_{out} \times h_{out} \times w_{out}}$, resulting in the rank-three tensor $\mathbf{Z} \in \mathbb{R}^{c_{out} \times h_{out} \times w_{out}}$ with feature map height h_{out} and feature map width w_{out} [32]:

$$Z_{i,j,l} = \sum_{a=1}^{c_{in}} \left(\mathbf{X}_a * \mathbf{K}_{i,a}\right)_{j,l} + B_{i,j,l} \tag{2.3}$$

For both types of layers, a non-linear activation function is applied element-wise to the result $\mathbf{z}$ or $\mathbf{Z}$. A common one is the rectified linear unit (ReLU) function, which clips negative values to zero [32]:

$$\mathrm{ReLU}(\mathbf{z}) = \max\left(\mathbf{0}, \mathbf{z}\right). \tag{2.4}$$

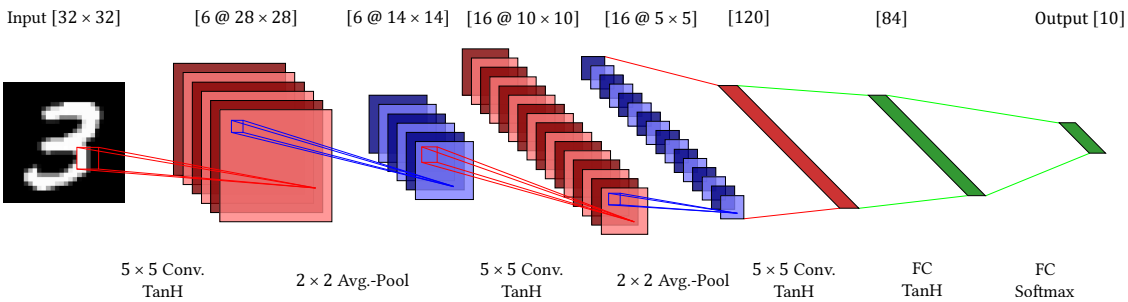


Figure 2.1.: Architecture of LeNet-5 with three convolutional layers, two fully connected layers and hyperbolic tangens (TanH) as activation function of the hidden layers. The input is required to be 32 px × 32 px grayscale images, here a 28 px × 28 px image from the MNIST dataset is upscaled beforehand. Red mappings refer to convolutional layers, green to fully connected layers and blue to average pooling. Moreover, stacked tiles represent feature maps and stripes one-dimensional feature vectors.

On the output layer of a classifier, the Softmax function can be used as an activation function to achieve a categorical probability distribution (also called Multinoulli distribution), meaning values of output layer's units are between zero and one, while all these values sum up to one [32]. Therefore, the output values of the Softmax function can be interpreted as certainties that the input $\mathbf{x}$ belongs to the respective class. It is defined as

$$\text{Softmax}(\mathbf{z}) = \frac{\exp \mathbf{z}}{\sum \exp \mathbf{z}}. \tag{2.5}$$

Opposite to fully connected layers, convolutional layers in many cases perform another operation, called pooling. Pooling basically reduces the size of feature maps by mapping several values onto a single one. Similar to convolution, a fixed-shape window is slid over all regions of the activated feature maps, creating views. But instead of computing a weighted sum, a reducing operation is performed on the view. For example, $n \times n$-average-pooling takes the average value from each $n \times n$ view onto the feature maps [32].

Last, the concrete composition of a DNN's layers, including their types, sizes, activation functions, etc., is called the *architecture* of the DNN [32]. For example, fig. 2.1 visualizes the LeCun Network (LeNet)-5 architecture [34] (named by its originator Yann LeCun) with three convolutional layers and two fully connected layers, being designed e.g. for the above-mentioned MNIST dataset.

2.2. TRAINING OF DEEP NEURAL NETWORKS

Training a DNN means gradually adapting its parameters $\boldsymbol{\theta}$ in such a way that the discrepancy between model's output $\hat{\mathbf{y}}$ and the objectively correct output $\mathbf{y}$, called *ground*

truth, becomes minimal for all inputs $\mathbf{x}$ of the considered data distribution ρ_{data}. The aim of training is that the DNN achieves a high performance, e.g., being measured by the *accuracy* metric, which states the proportion of correctly classified elements over a finite set of samples from ρ_{data} [32].

However, since accuracy is basically a binary metric (either a specific prediction is correct or it is not), it is not suitable to fine tune parameters as detailed as required. By contrast, a *loss function* $J : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ provides an indirect performance measurement, mapping the discrepancy of a model's output $\hat{\mathbf{y}}$ and the corresponding ground truth data $\mathbf{y}$ onto a real number. A loss function hence takes the complete output data into account, rather than just the binary result of being a match or not. The smaller the discrepancy between $\hat{\mathbf{y}}$ and $\mathbf{y}$, the lower the loss function becomes. Thus, optimization of the model parameters $\boldsymbol{\theta}$ can be achieved by minimization of the loss function over the considered data distribution, being modeled by the training dataset $\hat{\rho}_{\text{data}}$ [32]:

$$\arg\min_{\boldsymbol{\theta}} \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\hat{\rho}_{\text{data}}} J(\hat{f}(\mathbf{x}), \mathbf{y}). \tag{2.6}$$

The loss function can be extended by a parameter norm penalty, regularizing the parameter adaption to prevent *overfitting*, i.e., excessive adaption to the training dataset $\hat{\rho}_{\text{data}}$, but insufficient generalization to other samples from ρ_{data} For example, L^2 *regularization* adds the L^2 norm of the current parameter vector $\boldsymbol{\theta}$ to the loss function, penalizing high absolute parameter values that may create highly specialized models. With L^2 regularization, the composite loss function results in

$$J(\hat{\mathbf{y}}, \mathbf{y}) + \alpha \frac{1}{2} \|\mathbf{w}\|_2^2, \tag{2.7}$$

where $\alpha \in \mathbb{R}_{\geq 0}$ is a hyperparameter[1], denoting the regularization strength, and $\mathbf{w}$ represents the parameter vector without any bias parameters. The bias parameters are unregularized to avoid underfitting [32].

For multinomial classification, *cross-entropy* is often used as a loss function. Cross-entropy is a statistical measure that makes a statement about the dissimilarity between two probability distributions [35].[2] Let $\mathbf{p} \in \mathbb{R}^n$ and $\mathbf{q} \in \mathbb{R}^n$ be two discrete probability distributions. Then, the cross entropy of $\mathbf{q}$ w.r.t. $\mathbf{p}$ is defined as

$$\mathbb{H}(\mathbf{p}, \mathbf{q}) = -\sum_{i=1}^{n} p_i \log q_i. \tag{2.8}$$

Since in multinomial classification the output layer returns a discrete probability distri-

[1]Hyperparameters are parameters that are not optimized during training, but set beforehand.

[2]In general, cross-entropy is not symmetrical, i.e., $\mathbb{H}(\mathbf{p}, \mathbf{q}) \neq \mathbb{H}(\mathbf{q}, \mathbf{p})$ [35]. However, in terms of cross-entropy loss, only the first term is needed.

bution $\hat{\mathbf{y}}$ and the ground truth data $\mathbf{y}$ can be interpreted as fixed reference probability distribution, cross entropy loss is analogously defined as

$$J(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{i=1}^{m} y_i \log \hat{y}_i. \tag{2.9}$$

The loss function w.r.t. $\boldsymbol{\theta}$ is typically non-convex [32]. In order to minimize such a non-convex function, one can use *gradient descent*, if the function is differentiable. In gradient descent, the gradient of the loss function w.r.t. the model parameters is gradually subtracted from the current parameter vector $\boldsymbol{\theta}^{(\mathbf{i})}$, beginning at an either randomly or purposely chosen initial parameter vector $\boldsymbol{\theta}^{(\mathbf{0})}$ and eventually converging at a minimum [32]:

$$\boldsymbol{\theta}^{(\mathbf{i+1})} = \boldsymbol{\theta}^{(\mathbf{i})} - \eta \nabla_{\boldsymbol{\theta}} J. \tag{2.10}$$

Here, $\eta \in \mathbb{R}_{>0}$ is a hyperparameter and denotes the so-called learning rate, which scales the gradient. When setting the learning rate, one has to find a good trade-off between fast convergence and the risk of leaping over a minimum. It is usually reduced as the training process progresses, leading to local optimization rather than further exploration of the parameter space. Unfortunately, non-convexity implies that gradient descent does not necessarily converge at the global minimum, but can converge at local minima as well [36]. Moreover, gradient descent can get stuck at saddle points and plateaus of the loss landscape, causing insufficient parameter adaption. There exist, however, several sophisticated variants of gradient descent that can reduce risk for convergence at those bad critical points, being capable of escaping from there. In addition, Choromanska et al. [37] have shown that most local minima are sufficient, while the global minimum often raises overfitting.

Because the numerical computation of the gradient $\nabla_{\boldsymbol{\theta}} J$ through the difference quotient is computationally too expensive[3], it cannot be used. The *backpropagation* algorithm, on the contrary, allows efficient and analytical computation of the gradient, using the multivariable chain rule of calculus. Assuming that every mathematical operation being part of the network's forward pass can be expressed as a differentiable function, the loss function can be seen as a composition of these functions, being differentiable itself. Then, starting from the loss function, partial differentials w.r.t. intermediate results of hidden layers can be computed by the aid of the chain rule. The partial differentials between those intermediate results and the parameters being involved in their computation are known; hence, partial differentials of the loss function w.r.t. these parameters and therefore

[3]The loss function would have to be evaluated once for every single model parameter. Every evaluation includes a separate forward pass through the DNN.

the gradient can ultimately be retrieved from appropriate multiplication of known and previously computed differentials. This is also called backward pass [32].

Classical gradient descent on the one hand incorporates the whole training dataset per optimization step. In view of large datasets, such a forward and backward pass can be computationally very expensive, slowing down the training process. Stochastic gradient descent (SGD) on the other hand only takes a subset of the whole training dataset, called *mini-batch*, per optimization step. Since these subsets are not necessarily representative for the considered data distribution, the computed gradients are only an approximation of the actual gradients and tend to be noisy, especially for small batch sizes. Therefore, the progress of minimization is not monotone. But only due to the noisy gradients, SGD is capable of escaping from local minima and plateaus on the loss landscape [32].

The input $\mathbf{x}$ of a DNN is usually normalized and standardized, leading to faster convergence during training [38]. While sole normalization is sensitive to outliers in the data, image data is naturally bounded by the closed interval $[0..255]$. Hence, mapping pixel intensities linearly onto the closed interval $[0, 1]$ is sufficient.

Last, the training process is organized in terms of *epochs*. An epoch is the utilization of the complete training data set for loss minimization, regardless of the batch size and the resulting number of minimization steps. After an epoch, the loss is usually evaluated for a separate validation dataset as well, which is withheld from actual training. If the validation loss begins to increase and the training loss simultaneously continues to decrease, this indicates overfitting and training should be stopped [32].

2.3. BALLS AND SPHERES IN METRIC SPACES

Let $\mathcal{X} \subseteq \mathbb{R}^n$ be a metric space with a metric $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$. Furthermore, let $r \in \mathbb{R}_{\geq 0}$ and $\mathbf{x}^{(0)} \in \mathcal{X}$. Then, first, the *closed ball* of radius r centered at the point $\mathbf{x}^{(0)}$ is the set

$$\overline{\mathcal{B}}_{\mathcal{X}}(\mathbf{x}^{(0)}, r) = \{\mathbf{x} \in \mathcal{X} \mid d(\mathbf{x}, \mathbf{x}^{(0)}) \leq r\}. \tag{2.11}$$

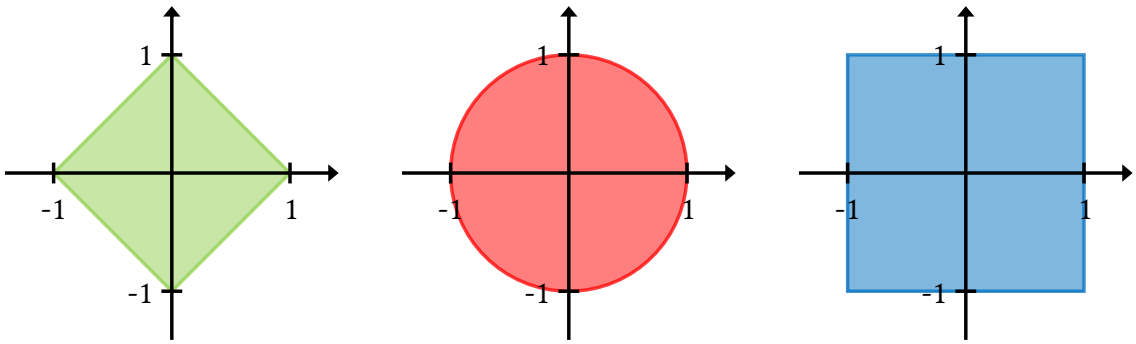
Second, the *open ball* of radius r centered at the point $\mathbf{x}^{(0)}$ is the set

$$\mathcal{B}_{\mathcal{X}}(\mathbf{x}^{(0)}, r) = \{x \in \mathcal{X} \mid d(\mathbf{x}, \mathbf{x}^{(0)}) < r\}. \tag{2.12}$$

Third, the *sphere* of radius r centered at the point $\mathbf{x}^{(0)}$ is the set

$$\mathcal{S}_{\mathcal{X}}(\mathbf{x}^{(0)}, r) = \{\mathbf{x} \in \mathcal{X} \mid d(\mathbf{x}, \mathbf{x}^{(0)}) = r\} = \overline{\mathcal{B}}_{\mathcal{X}}(\mathbf{x}^{(0)}, r) \setminus \mathcal{B}_{\mathcal{X}}(\mathbf{x}^{(0)}, r). \tag{2.13}$$

In addition, for a closed ball $\overline{\mathcal{B}}_{\mathcal{X}}(\mathbf{x}^{(0)}, r)$, the function $\Pi_{\overline{\mathcal{B}}_{\mathcal{X}}(\mathbf{x}^{(0)}, r)} : \mathcal{X} \to \overline{\mathcal{B}}_{\mathcal{X}}(\mathbf{x}^{(0)}, r)$ denotes the Euclidean projection function that projects any point $\mathbf{x}^* \in \mathcal{X}$ onto $\overline{\mathcal{B}}_{\mathcal{X}}(\mathbf{x}^{(0)}, r)$:



(a) $p = 1$ **(b)** $p = 2$ **(c)** $p = \infty$

Figure 2.2.: Closed ℓ_p unit balls for $p = 1, 2, \infty$ in two dimensions. The closed ℓ_1 unit ball $\overline{\mathcal{B}}_{\ell_1^{(n)}}(\mathbf{0}, \mathbf{1})$ (left) corresponds to a 45°-rotated square with diagonal length two, the closed ℓ_2 unit ball $\overline{\mathcal{B}}_{\ell_2^{(n)}}(\mathbf{0}, \mathbf{1})$ (center) to the unit circle and the closed ℓ_∞ unit ball $\overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{0}, \mathbf{1})$ (right) to the unit square.

$$\Pi_{\overline{\mathcal{B}}_{\mathcal{X}}(\mathbf{x}^{(0)}, r)}(\mathbf{x}^*) = \underset{\mathbf{x} \in \overline{\mathcal{B}}_{\mathcal{X}}(\mathbf{x}^{(0)}, r)}{\arg\min} \|\mathbf{x} - \mathbf{x}^*\|_2. \tag{2.14}$$

Last, the *closed/opened unit ball* and the *unit sphere* are defined as the closed/opened ball and sphere with radius $r = 1$, respectively [39]. Geometrically speaking, the sphere represents the surface of the corresponding closed ball, while the opened ball represents the inner volume of the closed ball, excluding its surface.

2.4. ℓ_p SPACES

An ℓ_p *space* is a normed vector space, whose norm is the *p-norm* $\|\cdot\|_p$. For $p \in [1, \infty)$ and an n-dimensional vector $\mathbf{x} \in \mathbb{R}^n$, the p-norm of $\mathbf{x}$ is defined as [39]

$$\|\mathbf{x}\|_p = \begin{cases} \sqrt[p]{\sum_{i=1}^n |x_i|^p}, & \text{for } 1 \leq p < \infty \\ \lim\limits_{p \to \infty} \sqrt[p]{\sum_{i=1}^n |x_i|^p} = \max_{i \in \{1,2,\ldots,n\}} |x_i|, & \text{for } p = \infty \end{cases}. \tag{2.15}$$

Let $\ell_p^{(n)}$ be an n-dimensional ℓ_p space. For $\mathbf{x}^{(1)}, \mathbf{x}^{(2)} \in \ell_p^{(n)}$,

$$d_p(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) = \|\mathbf{x}^{(1)} - \mathbf{x}^{(2)}\|_p \tag{2.16}$$

is the p-norm induced metric inside $\ell_p^{(n)}$, called *p-metric*, and the result itself is called ℓ_p *distance*. Commonly used p-metrics are the 1-metric, also known as *Manhatten distance*, the 2-metric, which is simply the Euclidean distance, as well as the ∞-metric, also known as *Chebyshev distance* [40].

Furthermore, the *closed/opened ℓ_p ball* and the *ℓ_p sphere* are defined as the closed/opened ball and sphere w.r.t. the corresponding p-metric, respectively [39]. In two dimensions, the closed ℓ_1 unit-ball corresponds to a 45°-rotated square with diagonal length two, the closed ℓ_2 unit-ball to the unit circle and the closed ℓ_∞ unit-ball to the unit square, as shown in fig. 2.2. In n dimensions, this can be generalized to the shapes of a cross-polytope (ℓ_1), a hypersphere (ℓ_2) and a hypercube (ℓ_∞).

A closed ℓ_p ball is always a *convex set*, i.e., for two points $\mathbf{x}^{(1)}, \mathbf{x}^{(2)} \in \overline{\mathcal{B}}_{\ell_p^{(n)}}(\mathbf{x}^{(0)}, r)$ with $\mathbf{x}^{(0)} \in \ell_p^{(n)}$ and $r \in \mathbb{R}_{\geq 0}$, any point on the line segment that joins $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$ is part of $\overline{\mathcal{B}}_{\ell_p^{(n)}}(\mathbf{x}^{(0)}, r)$ [39]:

$$\forall \lambda \in [0,1] : \lambda \mathbf{x}^{(1)} + (1-\lambda)\mathbf{x}^{(2)} \in \overline{\mathcal{B}}_{\ell_p^{(n)}}(\mathbf{x}^{(0)}, r). \tag{2.17}$$

Last, for an appropriate n-dimensional metric ℓ_∞ space $\ell_\infty^{(n)} \subseteq \mathbb{R}^n$, center point $\mathbf{x}^{(0)} \in \ell_\infty^{(n)}$ and radius $r \in \mathbb{R}_{\geq 0}$, the Euclidean projection Π of a point $\mathbf{x}^* \in \ell_\infty^{(n)}$ onto the closed ℓ_∞ ball $\overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x}^{(0)}, r)$ is

$$\Pi_{\overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x}^{(0)}, r)}\left(\mathbf{x}^*\right) = \max\left(\mathbf{x}^{(0)} - r \cdot \mathbf{1}, \min\left(\mathbf{x}^*, \mathbf{x}^{(0)} + r \cdot \mathbf{1}\right)\right). \tag{2.18}$$

The corresponding proof is stated in appendix A.1.

2.5. ADVERSARIAL EXAMPLES

Informally, an *adversarial example* is an only slightly perturbed version of a natural image which causes a DNN classifier to misclassify it as opposed to the original image.

Let $\mathcal{X}_p = [0,1]^n$ be the n-dimensional input space of an image classifier $\hat{f}_c : \mathcal{X}_p \to C$ with the m-elements set of classes $C = \{1, 2, \ldots, m\}$. Furthermore, the input space $\mathcal{X}_p$ is a metric space that is based on the p-metric d_p. As a short notation, we write $\mathcal{X}$ for $\mathcal{X}_\infty$. Hence, note that $\mathcal{X}$ is equivalent to the closed ℓ_∞ ball $\overline{\mathcal{B}}_{\ell_p^{(n)}}\left(\frac{1}{2} \cdot \mathbf{1}, \frac{1}{2}\right)$.

Moreover, let d_p measure the dissimilarity of two images, and let $\epsilon \in \mathbb{R}_{\geq 0}$ be a maximum permitted dissimilarity score. Then, we call a perturbed version $\mathbf{x}' \in \mathcal{X}_p$ of a natural image $\mathbf{x} \in \mathcal{X}_p$ with ground truth class c adversarial iff the following holds [16]:

$$\hat{f}_c(\mathbf{x}) = c \quad \wedge \quad \hat{f}_c(\mathbf{x}) \neq \hat{f}_c(\mathbf{x}') \quad \wedge \quad d(\mathbf{x}, \mathbf{x}') \leq \epsilon. \tag{2.19}$$

2.6. THREAT MODELS

A threat model typically "enumerate[s] the goals and capabilities of adversaries in a target domain" [41]. While the "target domain" is clearly defined by image classification

of DNNs, the remaining terms "capabilities" and "goals" have to be specified more closely here.

First, concerning adversarial attacks of DNNs, "capabilities" mostly refers to the kind of access given to the target DNN as well as the knowledge that follows from it. A common differentiation here is distinction between the *white-box setting* and the *black-box setting* [18]. While there are more fine-grained kinds of differentiation [41], we use the following one in the scope of this work:

In the white-box setting, adversaries have unrestricted access to the target DNN. They have knowledge about its architecture as well as its parameter values $\boldsymbol{\theta}$ after training. In consequence, they are able to compute gradients of the model function $\hat{f}(\mathbf{x}; \boldsymbol{\theta})$. Moreover, they have access to a representative set of images sampled from the data distribution ρ_{data} to which the target DNN is adapted, although this does not necessarily have to be the used training dataset $\hat{\rho}_{\text{data}}$.

Opposite to this, in the black-box setting, adversaries have only restricted access to the target DNN. Specifically, they have no further knowledge about its internals, but can only pass an input image to the model and receive the classification result as an output, in the manner of an oracle. Hence, they are not able to compute gradients of the model function $\hat{f}(\mathbf{x}; \boldsymbol{\theta})$ as well. However, they still have access to a representative set of images sampled from the considered data distribution ρ_{data}, as in the white-box setting.

Second, "goals" points, for one thing, to the kind of misclassification an adversary aims to achieve. In the simplest case, we can differentiate here between the *untargeted setting*, where any other class apart from the correct one is accepted, and the *targeted setting*, where a specific wrong class is to be predicted by the classifier. Again, there are further subdivisions of the spectrum [41]. In the scope of this work, we only consider the untargeted setting.

For another thing, an adversary has to specify the dissimilarity metric d on the basis of which the adversarial examples are to be created. Due to the ease of their computation, p-metrics are frequently used in practice, in particular the 2-metric [21, 42] as well as the ∞-metric [20, 21]. In the scope of this work, we focus on the latter.

In total, adversarial attacking becomes more difficult with both increasing complexity of adversarial goals and decreasing scope of adversarial capabilities [41].

3 MATERIALS AND METHODS

In this chapter, we present both the materials and the methods used in this work. First, in section 3.1, we show the two considered image datasets, being followed by the chosen DNN architecture in section 3.2. Next, we explain some algorithms that are utilized for DNN training in section 3.3. In sections 3.4 and 3.5, we first introduce methods to create adversarial examples and then present adaptions of the training process to increase the robustness of DNN against them. Last, section 3.6 specifies the concrete experiments being conducted in this work.

3.1. DATASETS

We use two different public image classification datasets, namely the Canadian Institute for Advanced Research (CIFAR)-10 and the German Traffic Sign Recognition Benchmark (GTSRB) dataset.

3.1.1. *CIFAR-10*

The CIFAR-10 dataset was published in 2009 by Krizhevsky [30]. It contains 60 000 32 px × 32 px RGB images, which are labeled with one of the ten classes airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. The dataset is uniformly distributed over the set of classes, i.e., there are 6000 images per class.

The dataset already provides a split between a training and a validation dataset. The

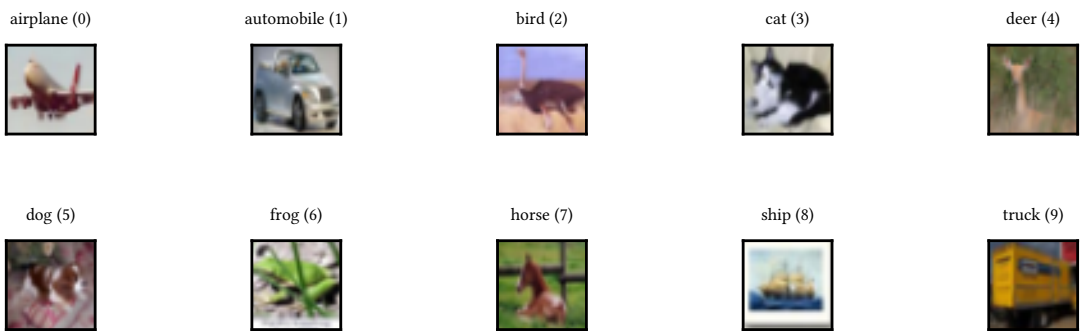


Figure 3.1.: Example images from the CIFAR-10 dataset. Per class, one example images was taken randomly from the training dataset.

training dataset of CIFAR-10 consists of 50 000 images randomly selected from the entire dataset, where all classes are 5000 times each and therefore equally represented. The validation dataset encompasses the remaining 10 000 images, being uniformly distributed over the ten classes as well.

Figure 3.1 shows some example images from the CIFAR-10 dataset.

3.1.2. *GTSRB*

The second considered dataset is the GTSRB dataset, being published in 2011 by Stallkamp et al. [31]. It consists of 51 839 RGB images that vary from 15 px × 15 px to 222 px × 193 px in resolution. Furthermore, the images depict a total of 43 different German traffic signs, which act as classes.

Just as for CIFAR-10, the GTSRB dataset provides a split between a training and a validation dataset. The training dataset here encompasses 39 209 images and the validation dataset contains the remaining 12 630 images. The split is based on random selection with some constraints to preserve the class distribution. Unlike CIFAR-10, the class distribution is not uniform, so some classes are overrepresented (e.g., the class "speed limit 30 km/h" with 5.7 % share and the class "speed limit 50 km/h" with 5.8 % share) and others are underrepresented (e.g., the classes "speed limit 20 km/h" and "left-hand bend" with 0.5 % share each). Note that the ideal uniform distribution had a class share of 2.3 %. The total relative class distribution is visualized as a histogram in fig. 3.2.

Finally, fig. 3.3 shows some example images from the GTSRB dataset as well.

3.2. DEEP NEURAL NETWORK ARCHITECTURE

We use the *pre-activation (PreAct)-residual network (ResNet)-50* architecture [29] for the DNNs trained in this work.

3.2.1. *Residual Blocks*

First of all, the general *ResNet* architecture stands out due to including so-called *residual blocks*, which are a composition of several, typically convolutional layers. Basically, such a residual block does not pass the sole output of its last layer to the following layer, but adds the input of its first layer to the output of its last layer, called *skip connection*. In the original version of the ResNet architecture, this addition is performed just before the activation of the block's last layer. In general, the ReLU function is used as an activation function for the hidden layers in the ResNet architecture.

So let $\mathbf{x}$ be the input to the residual block, let $\mathbf{z}$ be the non-activated output of the block's last layer and let $\mathcal{F}$ with $\mathcal{F}(\mathbf{x}) = \mathbf{z}$ be the composite function of all layers inside

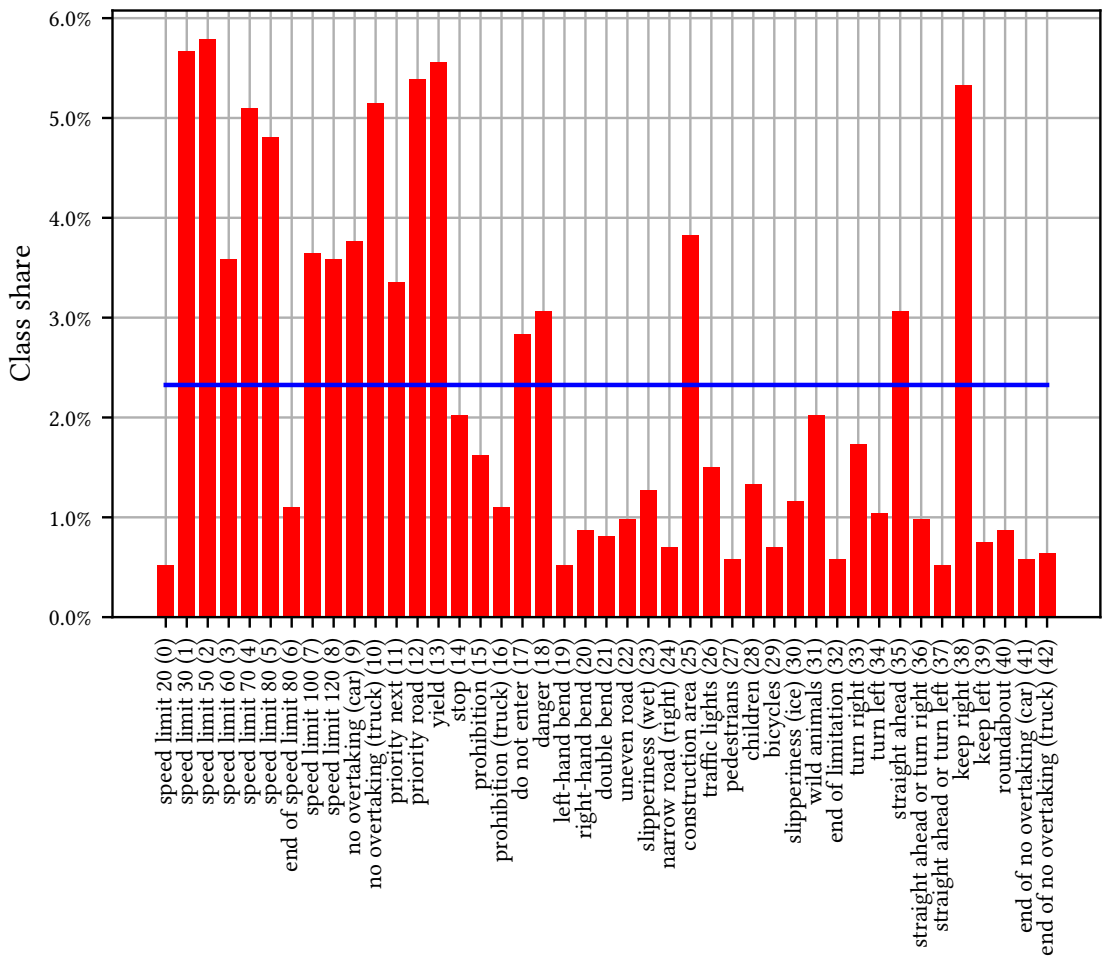


Figure 3.2.: Relative class distribution of the GTSRB dataset. The blue bar marks the ideal uniform distribution.

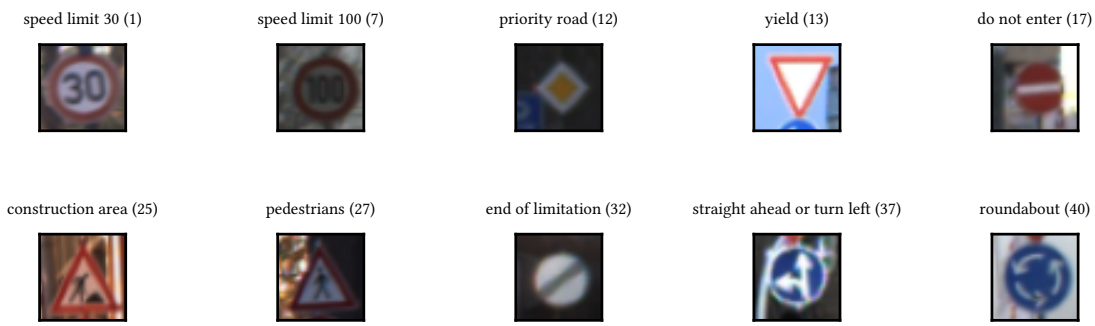


Figure 3.3.: Example images from the GTSRB dataset. Ten example images with distinct classes were taken randomly from the training dataset. They were resized to 32 px × 32 px.

the block, excluding last layer's activation.[1] Then, $\mathcal{F}$ is called the *residual function* and the eventual output of the corresponding residual block is computed by [5]

$$\mathrm{ReLU}(\mathcal{F}(\mathbf{x}) + \mathbf{x}). \tag{3.1}$$

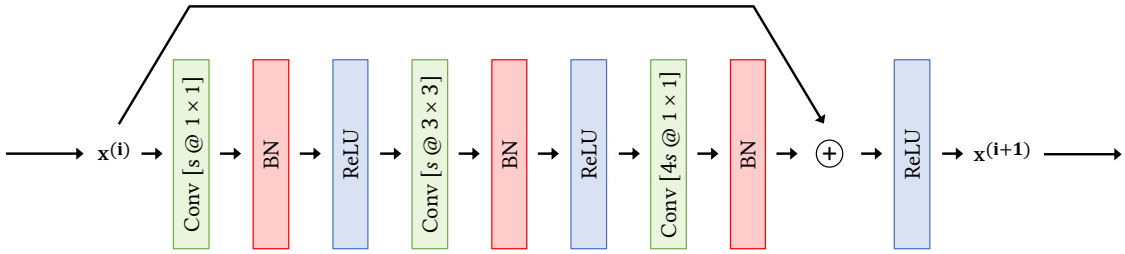
The rationale for such DNN architecture originates from the empirical observation that deeper neural networks show poorer classification performance than shallower ones. This also holds for comparisons where the deeper network is just an extended version of the shallower one, incorporating additional layers, but sharing the same architecture apart from that. In particular the latter contradicts the constructed solution of the deeper network learning layer-wise the same mappings as the shallower one and learning the identity mapping for its additional layers [5].

Consequently, He et al. [5] assumed that the identity mapping might be difficult to learn. So instead of attempting to let a DNN learn the desired underlying mapping $\mathcal{H}(\mathbf{x})$ directly, they tried to let it learn the additive residual $\mathcal{F}(\mathbf{x}) = \mathcal{H}(\mathbf{x}) - \mathbf{x}$. They "hypothesize[d] that it [was] easier to optimize the residual mapping than to optimize the original, unreferenced mapping" and that, "if an identity mapping were optimal, it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers" [5].

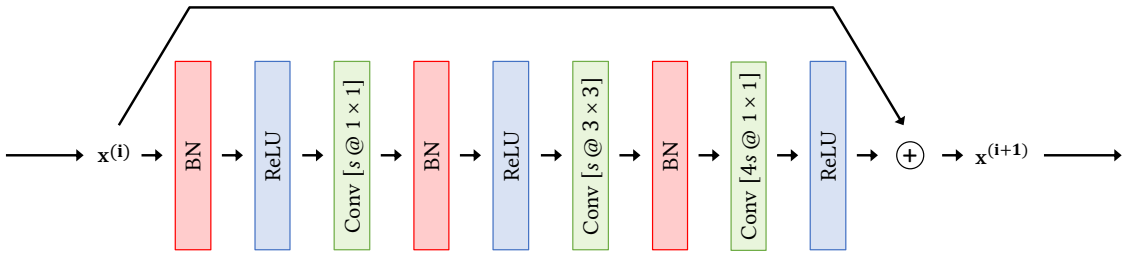
Opposite to the original reasoning, however, the identity mapping being the optimal mapping for several layers is quite unlikely, as the authors ascertained [5]. But the use of residual blocks has another side effect that might explain its success better. In usual "plain" DNNs, differentials are simply multiplied layer-wise with each other to compute the derivatives of the loss function w.r.t. the model parameters. If the absolute value of differentials is less than one, derivatives become steadily smaller for hidden layers being closer to the input layer. This phenomenon is known as *gradient vanishing* [43] and can lead to insufficient parameter adaption, especially at front layers. With the ResNet architecture, on the contrary, skip connections allow derivatives of deeper layers to move to front layers without a loss in signal during backpropagation. This can be shown easily: Omitting the ReLU function for clarity and denoting the input and output of the i-th residual block with residual function $\mathcal{F}^{(i)}$ by $\mathbf{x}^{(\mathbf{i})}$ and $\mathbf{x}^{(\mathbf{i+1})}$, respectively, the derivative of the loss function J w.r.t. $\mathbf{x}^{(\mathbf{i})}$, given the derivative of J w.r.t. $\mathbf{x}^{(\mathbf{i+1})}$ from backpropagation so far, is computed by [29]

$$\frac{\partial J}{\partial \mathbf{x}^{(\mathbf{i})}} = \frac{\partial J}{\partial \mathbf{x}^{(\mathbf{i+1})}} \frac{\partial \mathbf{x}^{(\mathbf{i+1})}}{\partial \mathbf{x}^{(\mathbf{i})}} = \frac{\partial J}{\partial \mathbf{x}^{(\mathbf{i+1})}} \frac{\partial \left(\mathcal{F}(\mathbf{x}^{(\mathbf{i})}) + \mathbf{x}^{(\mathbf{i})}\right)}{\partial \mathbf{x}^{(\mathbf{i})}} = \frac{\partial J}{\partial \mathbf{x}^{(\mathbf{i+1})}} \left(\frac{\partial \mathcal{F}^{(i)}(\mathbf{x}^{(\mathbf{i})})}{\partial \mathbf{x}^{(\mathbf{i})}} + 1 \right). \tag{3.2}$$

[1] In practice, e.g., input data to convolutional layers are rank-three tensors (see section 2.1). To the sake of simplicity, in the following we consider data in their flattened, vectorial representation.



(a) ResNet-50



(b) PreAct-ResNet-50

Figure 3.4.: Structure of a residual block for the ResNet-50 (top) and PreAct-ResNet-50 (bottom) architectures. Convolutional layers are colored green, Batch Normalization red and ReLU activation blue.

One can see here that the derivative of J w.r.t. $\mathbf{x}^{(\mathbf{i+1})}$ is simply added to the derivative of the actual layer transformations included in $\mathcal{F}^{(i)}$.

Internally, in case of the ResNet-50 architecture[2], the residual function of a single residual block consists of three subsequent convolutional layers. More specifically, 1×1, 3×3 and 1×1 convolution are performed in that order. Furthermore, the kernel count of convolutional layers depends on an integer s, which itself depends on the location of the considered residual block in the whole DNN and will be discussed later. In any case, for the first and the second convolutional layer of a residual block, the kernel count equals s; for the third one, it is $4s$ [5].

One might have noticed that the skip connection as shown in eq. (3.1) cannot work, if the input's shape differs from the output's shape. In such cases, a minimal transformation is performed to align shapes, e.g., a 1×1 convolution with as many kernels as output channels, if the channel counts are different [5].

Last, fig. 3.4a visualizes the structure of a residual block for the ResNet-50 architecture.

3.2.2. *Batch Normalization*

As already mentioned, the activation function each convolution is followed by is the ReLU function. It does, however, not follow directly, but another operation is performed in

[2]The suffix "50" here implies that the corresponding DNN has 50 layers in total.

between, called *Batch Normalization* [5]. The fundamental observation from which Batch Normalization is motivated is that parameter changes during training also change the input distribution a subsequent layer is confronted with, since their input is dependent on the parameters of previous layers. Ioffe and Szegedy [44] called this effect the *Internal Covariate Shift*. As it slows down the training, they invented Batch Normalization to standardize[3] the input of a DNN's hidden layer, leading to a fixed input distribution.

Batch Normalization essentially consists of two steps. First, the current data batch $\mathbf{X} \in \mathbb{R}^{b \times n}$ with batch size b is standardized element-wise to mean zero and variance one over the batch dimension:

$$\forall i \in \{1, 2, \ldots, b\} : \forall j \in \{1, 2, \ldots, n\} : Z_{i,j} = \frac{X_{i,j} - \mathbb{E}\left((\mathbf{X}^\intercal)_\mathrm{j}\right)}{\sqrt{\mathbb{V}\left((\mathbf{X}^\intercal)_\mathrm{j}\right)}}. \tag{3.3}$$

As standardization can also reduce the learning capabilities of the DNN, the standardized data $\mathbf{Z} \in \mathbb{R}^{b \times n}$ is then linearly transformed in a second step:

$$\forall i \in \{1, 2, \ldots, b\} : \forall j \in \{1, 2, \ldots, n\} : Z^*_{i,j} = \gamma_j Z_{i,j} + \zeta_j. \tag{3.4}$$

Here, $\gamma, \zeta \in \mathbb{R}^n$ denote further learnable parameters of the DNN [44].

3.2.3. *ResNet-50 and PreAct-ResNet-50 Architectures*

Considering a DNN with ResNet-50 architecture from top view, the very first hidden layer is a 3×3 convolutional layer with 64 kernels. Next, there are four top-level blocks, which contain a different number of residual blocks and also determine the aforementioned parameter s. The first of these top-level blocks contains three residual blocks with $s = 64$, the second four with $s = 128$, the third six with $s = 256$, and the forth another three with $s = 512$. Subsequently, 4×4 average-pooling is performed and the output is flattened, i.e., restructured as a vector per input image. Finally, a fully connected layer with width 2048 maps to the output layer, whose width is equal to the number of classes according to the considered classification task [5].

The eventually used PreAct-ResNet-50 architecture is a modification of the original ResNet-50 architecture, where the order of operations inside a residual block is changed. In detail, instead of being performed after convolutions as usual, both the Batch Normalization and the ReLU activation take place before the convolution, with Batch Normalization preceding ReLU as for the regular ResNet-50 architecture. Moreover, ReLU activation at the block's very end is moved into the residual function. Figure 3.4b visualizes the

[3]Despite the name, Batch Normalization is actually a standardization technique.

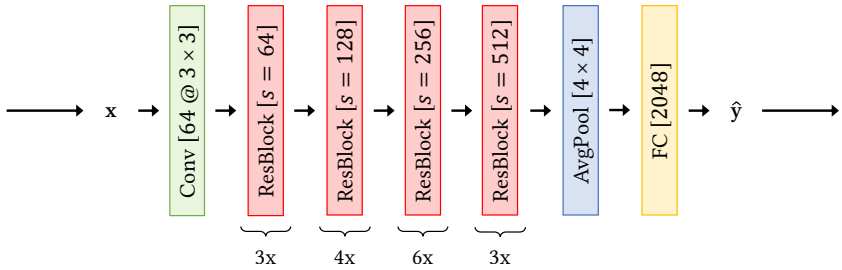


Figure 3.5.: Structure of the ResNet-50 and PreAct-ResNet-50 architecture. Convolutional layers are colored green, residual blocks red, average pooling blue and fully connected layers yellow.

structure of a residual block for the PreAct-ResNet-50 architecture [29].

The motivation for this change arises from the fact that unmodified signal propagation during backpropagation as pointed out in eq. (3.2) does not actually happen: The ReLU activation we omitted for clarity might have less impact on the signal during backpropagation than convolutions, since it is an identity mapping for all positive input, but it still leads to gradient vanishing in case of non-positive input. Therefore, moving the ReLU activation that is performed after addition into the residual function provides a nearly clean path for signal propagation through skip connections. Remaining, but mandatory operations are the 1×1 convolutions needed for shape alignment. But since they occur only four times, at each first residual block of a top-level block, they can be neglected. Besides, experiments have proven that the final PreAct architecture – with both Batch Normalization and ReLU activation preceding convolutional layers – yields the best learning performance, compared to other permutations [29].

Last, fig. 3.5 shows the structure of the entire ResNet-50 and PreAct-ResNet-50 architecture.

3.3. DEEP NEURAL NETWORK TRAINING

3.3.1. *Data Augmentation*

Data augmentation is a technique where training data is manipulated such that the diversity of training data is increased. One can think of reasonable, synthetic data that is created from the existing data. It is particularly used in case of image data, where semantics preserving transformations, e.g., scaling or rotation, can easily be applied.

In this work, data augmentation is dynamically applied to the images of the training datasets, i.e., the training datasets are not statically augmented beforehand, but the original images are manipulated anew each epoch. Most utilized transformations are of stochastic nature, providing different results when being applied several times on the same input image. As a consequence, the total data volume used for training remains

unchanged unlike to static augmentation, but the variability of data is incomparably higher, subject to a sufficiently high number of epochs.

We use the following transformations (in this order) as part of the augmentation process:

- Random horizontal flipping[4],
- random rotation by $\varphi \in [-15°, 15°]$,
- extrapolation to 38 px × 38 px using reflection along the image boundary by 3 px in each direction, and
- random cropping back to 32 px × 32 px.

3.3.2. *AdamW*

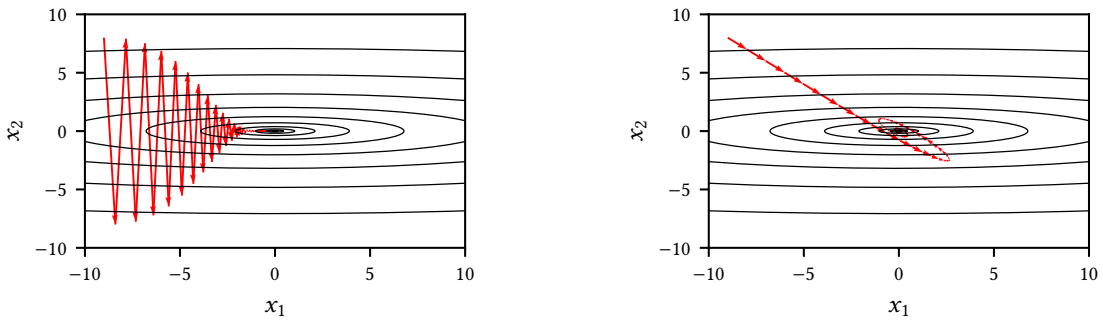
Adaptive moment estimation with decoupled weight decay (AdamW) is an optimization algorithm that represents a slight modification of the original adaptive moment estimation (Adam) algorithm to remove problems with L_2 regularization [45]. In the following, we first show how the basic Adam algorithm works and subsequently point out the difference to AdamW.

Adam is, equal to classical SGD, a first-order optimization method, i.e., it exploits only the gradient of the loss function to minimize the latter. It was developed by Kingma and Ba [46] in 2014, motivated from the observation that strictly following of the current gradient, as SGD does, can lead to oscillating or noisy minimization paths, which make for slow convergence themselves. Figure 3.6a visualizes the issue in a simple way for a two-dimensional, convex loss landscape, but one can imagine that it becomes even more problematic for a higher-dimensional, non-convex one as usual in DNN training – in particular, if the loss landscape changes continuously due to the usage of randomized mini-batches [32].

Informally speaking, Adam mitigates the described problem by "smoothing" the minimization path using exponential moving averages of the gradient (see fig. 3.6b). More specifically, per optimization step, the first moment (mean) and the second raw moment (uncentered variance) of the gradient are estimated by their exponential moving averages. After that, the actual step vector is computed by dividing the first moment estimate by the square root of the second raw moment estimate [46].

So let $\mathbf{g}^{(\mathbf{i})} = \nabla_{\boldsymbol{\theta}} J$ be the gradient of the loss function J w.r.t. the parameter vector $\boldsymbol{\theta}$ at the i-th optimization step. Moreover, let $\beta_1, \beta_2 \in [0, 1)$ be the decay rates for

[4]Horizontal flipping is omitted for the GTSRB dataset, where it actually can change semantics (e.g., by changing a turn left sign to a turn right sign).



(a) Gradient Descent

(b) Adam

Figure 3.6.: Optimization schemes of standard gradient descent and Adam in view of a two-dimensional convex optimization problem. 200 iterations are performed each. The initial learning rate is set $\eta = 1$. To avert leaping the minimum, it is reduced linearly for gradient descent. For Adam, parameters β_1, β_2 are set to their defaults ($\beta_1 = 0.9$, $\beta_2 = 0.999$) [46]. The optimization problem itself is characterized by a narrow valley. Note that while gradient descent exhibits a strongly fluctuating course initially, minimization is comparatively straightforward with Adam, although again not being perfect and spinning spirals at convergence.

exponential moving averages of the first moment and the second raw moment of the gradient, respectively. Then, the first moment estimate $\mathbf{m}^{(\mathbf{i})}$ and the second raw moment estimate $\mathbf{v}^{(\mathbf{i})}$ are computed by [46]

$$\mathbf{m}^{(\mathbf{i})} = \beta_1 \mathbf{m}^{(\mathbf{i}-\mathbf{1})} + (1 - \beta_1)\mathbf{g}^{(\mathbf{i})} \tag{3.5}$$

and

$$\mathbf{v}^{(\mathbf{i})} = \beta_2 \mathbf{v}^{(\mathbf{i}-\mathbf{1})} + (1 - \beta_2)\mathbf{g}^{(\mathbf{i})^2} \tag{3.6}$$

with $\mathbf{m}^{(\mathbf{0})} = 0$ and $\mathbf{v}^{(\mathbf{0})} = 0$. As a consequence of this initialization, however, both estimates are biased towards zero during the first optimization steps. To counter this and achieve meaningful estimates for the optimization step, they are bias-corrected as follows [46]:

$$\hat{\mathbf{m}}^{(\mathbf{i})} = \frac{\mathbf{m}^{(\mathbf{i})}}{1 - {\beta_1}^i}, \tag{3.7}$$

$$\hat{\mathbf{v}}^{(\mathbf{i})} = \frac{\mathbf{v}^{(\mathbf{i})}}{1 - {\beta_2}^i}. \tag{3.8}$$

Last, with η denoting the learning rate, the step vector is computed as described above and the parameter vector $\boldsymbol{\theta}$ is basically updated as for SGD [46]:

$$\boldsymbol{\theta}^{(\mathbf{i+1})} = \boldsymbol{\theta}^{(\mathbf{i})} - \eta \frac{\hat{\mathbf{m}}^{(\mathbf{i})}}{\sqrt{\hat{\mathbf{v}}^{(\mathbf{i})}}}. \tag{3.9}$$

Although the eventual effect of both the first moment estimate and the second raw moment estimate is similar, the cause actually differs: With the first moment estimate replacing the gradient itself in the update rule (see eq. (3.9)), gradients are averaged on an exponentially weighted basis, canceling out fluctuations. On the other hand, division by the square root of the second raw moment estimate can rather be interpreted as a parameter-wise rescaling of the learning rate η, when dividing the latter instead of the first moment estimate. If, e.g., a certain parameter θ_j exhibited frequent fluctuations during previous optimization steps, the corresponding second raw moment ${v^{(i)}}_j$ is high. In consequence, the learning rate for this specific parameter is scaled down, leading to smaller changes of θ_j during the following optimization steps. Analogously, the parameter-wise learning rate remains on a level close to η, if there are only minor fluctuations, and allows greater changes here. With these similar effects, the first moment estimate and the second raw moment estimate synergize well and make Adam an effective and also efficient optimization algorithm for DNN training [32].

However, as mentioned at the beginning, Adam turned out to be less effective when being combined with L^2 regularization, which is itself used to tackle the overfitting problem. The reason for this is that while L^2 regularization is useful and beneficial for classical SGD, it is not for Adam, where the estimation of moments distorts the effects of regularization [45].

Resorting to notations in section 2.2, the gradient of an L^2 regularized loss function J w.r.t. the parameter vector $\boldsymbol{\theta}$ results in $\nabla_{\boldsymbol{\theta}} J + \alpha \mathbf{w}$.[5] Hence, for SGD (see eq. (2.10)), the parameter update is performed by

$$\boldsymbol{\theta}^{(\mathbf{i+1})} = \boldsymbol{\theta}^{(\mathbf{i})} - \eta \left(\nabla_{\boldsymbol{\theta}} J - \alpha \mathbf{w}^{(\mathbf{i})} \right). \tag{3.10}$$

With Adam, on the other hand, regularization as part of the loss function happens before the estimation of moments and therefore $\mathbf{g}^{(\mathbf{i})} = \nabla_{\boldsymbol{\theta}} J + \alpha \mathbf{w}^{(\mathbf{i})}$ holds. As a result, the estimation of moments takes influence on the regulative effect of the gradient $\mathbf{g}^{(\mathbf{i})}$. In detail,

(a) the first moment estimate potentially damps or even cancels out regularization and

(b) the second moment estimate can lead to small parameter-specific learning rates, reducing the effect of regularization as well.

[5] In order to align shapes and enable vector addition, $\mathbf{w}$ is zero-padded for bias components of $\nabla_{\boldsymbol{\theta}} J$.

To solve this issue, Loshchilov and Hutter [45] developed a modified version of Adam, called AdamW. Essentially, it does not perform L_2 regularization as part of the loss function, but *weight decay* as part of the update rule, using the weight decay rate $\alpha^* \in \mathbb{R}$:

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \eta \frac{\hat{\mathbf{m}}^{(i)}}{\sqrt{\hat{\mathbf{v}}^{(i)}}} + \alpha^* \mathbf{w}^{(i)}. \tag{3.11}$$

Thus, estimations of the first moment and the second raw moment do not distort the effects of regularization anymore. Note that when substituting $\eta\alpha$ by α^* in eq. (3.10), the form of AdamW's update rule (eq. (3.11)) is now equivalent to that of SGD, showing the fundamental problem that led to the dedicated development of AdamW: L_2 regularization and weight decay are equivalent only for SGD, but usually, only L_2 regularization and not weight decay is implemented [45].

3.3.3. *Cosine Annealing*

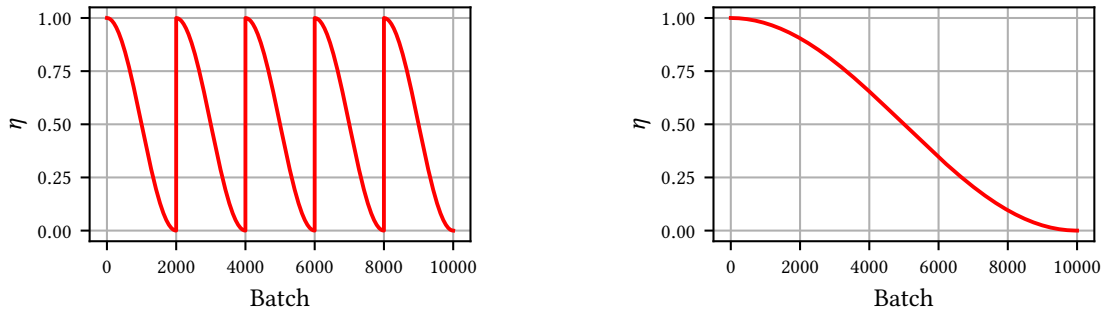
Cosine annealing is a method for learning rate scheduling in the course of DNN training, being proposed by Loshchilov and Hutter [47] as well. Learning rate scheduling here means that the learning rate η is not kept constant as training progresses, but is adjusted, or more precisely, lowered.

Basically, the cosine annealing schedule provides for a learning rate course in accordance with the first quarter period of the cosine function. The reduction can be done either after the completion of an epoch or, more fine-grained, after each batch performed. In this work, we use the latter definition.

The motivation for this particular kind of scheduling comes the following observation: Concerning the decrease in learning rate, a warm-up phase and a cool-down phase in which the change in learning rate is continuously increased and decreased, respectively, is more effective than, e.g., a continuous linear decrease. This follows the classical approach of heuristic optimization, where one seeks to explore the search space widely in the beginning and changes over to exploiting the previously achieved progress in the end. Analogously, cosine annealing allows larger optimization steps for the first batches and smaller ones for the last batches, while remaining on a comparable level for some time.

Loshchilov and Hutter also proposed the use of so-called *warm restarts*, where after reaching the minimum learning rate, it is reset to the initial learning rate again. Afterwards, cosine annealing begins again. One can imagine here that a sharp increase in learning rate provides an additional opportunity to escape from a local minimum and, subsequently, possibly find a deeper one.

Let $\eta^{(i)}$ be the learning rate in the i-th batch and let η_{max}, η_{min} be the initial, maximum and the final, minimum learning rate, respectively. Moreover, let T_{cur} denote the number



(a) With warm restarts

(b) Without warm restarts

Figure 3.7.: Cosine annealing learning rate scheduling. For the left plot, five warm restarts are performed ($T_{max} = 2000$), whereas for the right one, no warm restarts are performed at all ($T_{max} = 10000$). In both cases, 10 000 batches are performed and the initial and final learning rate are set $\eta_{max} = 1$ and $\eta_{min} = 0$.

of batches since the last warm restart and let T_{max} denote the number of batches per cosine annealing cycle, decreased by one. Then, $\eta^{(i)}$ is computed as follows [47]:

$$\eta^{(i)} = \eta_{min} + \frac{1}{2}\left(\eta_{max} - \eta_{min}\right)\left(1 + \cos\left(\frac{T_{cur}}{T_{max}}\pi\right)\right). \tag{3.12}$$

The development of the learning rate under cosine annealing is shown in fig. 3.7a for 10 000 performed batches and parameters $\eta_{max} = 1$, $\eta_{min} = 0$, $T_{max} = 2000$.

A special case comes up, if T_{max} equals the number of totally performed batches. In that case, no warm restarts are performed, being visualized in fig. 3.7b for the same remaining settings as in fig. 3.7a.

Last, despite the intrinsic learning rate adaption capabilities of the latter, cosine annealing has also proven to be effective in combination with Adam as well as AdamW. In comparison with an otherwise fixed learning rate, the learning performance could be further increased [45].

3.4. ADVERSARIAL ATTACKS

An *adversarial attack* is defined as a method to purposefully perturb given natural images with the aim of creating adversarial examples and causing misclassification. Furthermore, a specific attack is based on a threat model, modeling the circumstances under which it is performed [41].

As part of this work, we examine three different adversarial attacks, two of which are white-box attacks and one of which is a black-box attack. These are

(a) the Fast Gradient Sign Method (FGSM),

(b) Projected Gradient Descent (PGD), and

(c) the Boundary Attack.

In the following, we present how the corresponding algorithms work.

3.4.1. *Fast Gradient Sign Method*

The Fast Gradient Sign Method (FGSM), being published by Goodfellow, Shlens, and Szegedy [20] in 2015, was one of the very first techniques to create adversarial examples.

It is a white-box attack, so adversaries must have extensive knowledge on the architecture of the DNN $\hat{f}$ to be attacked as well as its trained parameters $\boldsymbol{\theta}$. Given a natural image and corresponding ground truth data $(\mathbf{x}, \mathbf{y}) \sim \rho_{\text{data}}$ with $\mathbf{x} \in \mathcal{X}$ and $\mathbf{y} \in \mathcal{Y}$, they must particularly be capable of computing the gradient of a suitable loss function J (typically, cross entropy loss) *w.r.t. the input* $\mathbf{x}$, i.e., $\nabla_{\mathbf{x}} J(\hat{f}(\mathbf{x}), \mathbf{y})$. Since the backpropagation algorithm can be used here as well – in comparison to DNN training, only the eventually intended derivatives are different –, this gradient can also be computed quite efficiently.

In order to create a perturbed image $\mathbf{x}'$ that is probably an adversarial example, first, the sign function is applied to the previously computed gradient $\nabla_{\mathbf{x}} J(\hat{f}(\mathbf{x}), \mathbf{y})$. Next, it is multiplied by a magnitude of perturbation $\epsilon \in \mathbb{R}_{\geq 0}$ to be specified in advance. The result then actually represents the perturbation vector, being added to the natural image $\mathbf{x}$.

However, the perturbed "image" obtained in this way is not necessarily part of the considered input space $\mathcal{X}$, and thus, by definition (see section 2.5), not necessarily a valid image anymore. Hence, it is ultimately projected back onto the input space $\mathcal{X}$:

$$\mathbf{x}' = \Pi_{\mathcal{X}} \left(\mathbf{x} + \epsilon \operatorname{sign} \nabla_{\mathbf{x}} J(\hat{f}(\mathbf{x}), \mathbf{y}) \right). \tag{3.13}$$

By applying the sign function and subsequently scaling with ϵ, the perturbed image $\mathbf{x}'$ is guaranteed to be part of the closed ℓ_∞ ball $\overline{\mathcal{B}}_{\mathcal{X}}(\mathbf{x}, \epsilon)$. Therefore, the FGSM implicitly operates in a threat model constrained by the ∞-metric d_∞. The corresponding proof is stated in appendix A.2. Besides, fig. 3.8 visualizes how the FGSM works.

The principle idea behind the FGSM is similar to that of DNN training: Based on the fact that the loss function J makes a statement on the defectiveness of the model function $\hat{f}$ in view of some input data $\mathbf{x}$ and ground truth data $\mathbf{y}$, it can generally be used as an objective function to perform optimization. What differs in the concrete is *the quantity to be optimized*: While DNN training minimizes J w.r.t. the model parameters in order to find the optimal model parameters and thus the best model, the FGSM *aims to maximize the loss w.r.t. the input data in order to find similar/close valid input data, but where the model is in error with the same ground truth data.* By specifying a sufficiently small magnitude of perturbation ϵ, the difference between an original image $\mathbf{x}$ and its

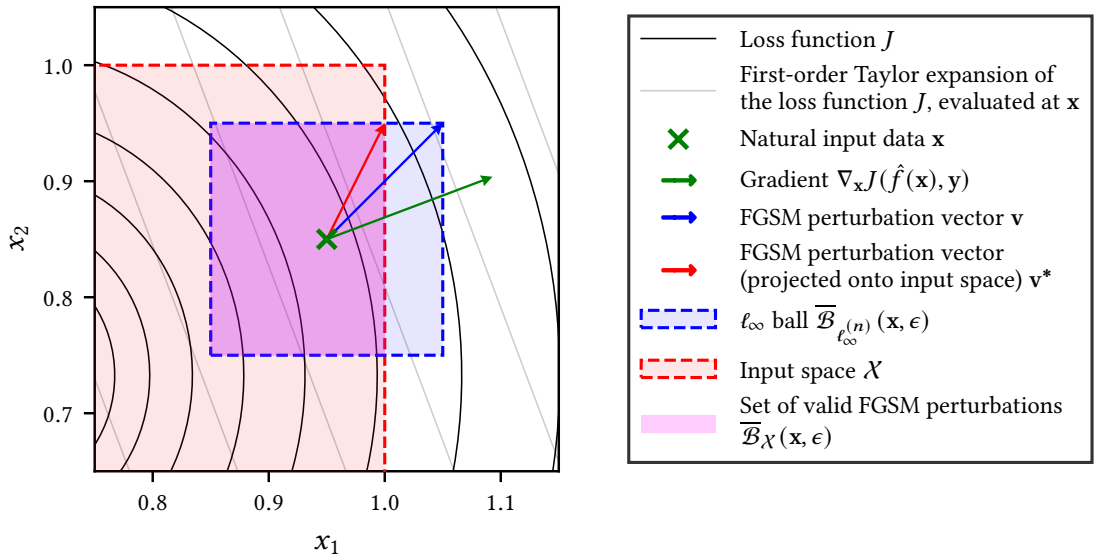


Figure 3.8.: Fast Gradient Sign Method in two dimensions. The loss function J is plotted as black contour lines and the natural input data $\mathbf{x} = (0.95, 0.85)^\intercal$ is marked by a green cross. Moreover, the gradient $\nabla_{\mathbf{x}} J(\hat{f}(\mathbf{x}), \mathbf{y})$ is visualized as a green arrow. The FGSM then maximizes the loss for the linearized loss function (gray contour lines) in the closed ℓ_∞ ball $\overline{\mathcal{B}}_{\mathcal{X}}(\mathbf{x}, \epsilon)$ (blue square) with $\epsilon = 0.1$, the resulting initial perturbation vector $\mathbf{v}$ is shown as a blue arrow. Last, the perturbation is projected back onto the input space $\mathcal{X}$ (red rectangle), leading to the final perturbation vector $\mathbf{v}^*$ (red arrow). One sees that $\mathbf{v}^*$ maximizes the linearized loss function in the set of valid FGSM perturbations $\overline{\mathcal{B}}_{\mathcal{X}}(\mathbf{x}, \epsilon)$ (purple area).

FGSM-perturbed version $\mathbf{x}'$ is constrained to be small enough to prevent a semantic change, which would contradict the ground truth data kept constant. In detail, as the FGSM operates in the closed ℓ_∞ ball $\overline{\mathcal{B}}_{\mathcal{X}}(\mathbf{x}, \epsilon)$, the maximum perturbation per pixel equals ϵ, independent of the total number of perturbed pixels.

While we have shown that applying the sign function as stated in eq. (3.13) leads to the aforementioned boundedness by $\overline{\mathcal{B}}_{\mathcal{X}}(\mathbf{x}, \epsilon)$, the question remains as to why it is used in particular. It should be clear that its purpose is to normalize the gradient $\nabla_{\mathbf{x}} J(\hat{f}(\mathbf{x}), \mathbf{y})$ w.r.t. the ∞-norm. But actually, this can be achieved by dividing by the ∞-norm $\|\nabla_{\mathbf{x}} J(\hat{f}(\mathbf{x}), \mathbf{y})\|_\infty$ as well, while maintaining the original direction of the gradient. The application of the sign function, on the other hand, results in a change of direction in the vast majority of cases.

To understand the reason, we look at the problem from a different perspective: We now consider the previously stated properties of the FGSM (being constrained by the closed ℓ_∞ ball $\overline{\mathcal{B}}_{\mathcal{X}}(\mathbf{x}, \epsilon)$, only one iteration of first-order optimization) as *premises*, under which a fast method to maximize the loss function J in the vicinity of the natural image $\mathbf{x}$ is to be developed. Operating in the vicinity of $\mathbf{x}$ allows us to linearize J: For some $\mathbf{v} \in \overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{0}, \epsilon)$, the first-order Taylor approximation of $J(\hat{f}(\mathbf{x} + \mathbf{v}), \mathbf{y})$ in the closed ℓ_∞

ball $\overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x}, \epsilon)$ is

$$J(\hat{f}(\mathbf{x}+\mathbf{v}), \mathbf{y}) \approx J(\hat{f}(\mathbf{x}), \mathbf{y}) + \mathbf{v}^\intercal \nabla_{\mathbf{x}} J(\hat{f}(\mathbf{x}), \mathbf{y}). \tag{3.14}$$

Here, $\mathbf{v}$ represents the direction from the point $\mathbf{x}$, in which the loss is increased by $\mathbf{v}^\intercal \nabla_{\mathbf{x}} J(\hat{f}(\mathbf{x}), \mathbf{y})$, relative to the loss at $\mathbf{x}$. Therefore, the vector $\mathbf{v}$ that maximizes the linearized loss function in $\overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x}, \epsilon)$ can be found by solving the constrained optimization problem

$$\underset{\mathbf{v} \in \overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{0}, \epsilon)}{\arg\max} \ \mathbf{v}^\intercal \nabla_{\mathbf{x}} J(\hat{f}(\mathbf{x}), \mathbf{y}). \tag{3.15}$$

It turns out that the solution to this problem is exactly $\mathbf{v} = \epsilon \operatorname{sign} \nabla_{\mathbf{x}} J(\hat{f}(\mathbf{x}), \mathbf{y})$. At this point, we omit a formal proof, since this can be verified quite intuitively:

(a) All components-wise pairs of $\mathbf{v}$ and $\nabla_{\mathbf{x}} J(\hat{f}(\mathbf{x}), \mathbf{y})$ should have the same sign, respectively. Indeed, they can, since the constraint $v_i \in [-\epsilon, \epsilon]$ applies to all components v_i with $i \in \{1, 2, \ldots, n\}$, so no component v_i is restricted to have a specific sign that might be different of that of $\frac{\partial J(\hat{f}(\mathbf{x}), \mathbf{y})}{\partial x_i}$.

(b) Second, the absolute value of all components v_i should be maximal. We quickly see that this is ϵ, again independent of the dimension.

Eventually, the synthesis of both demands results in $\mathbf{v} = \epsilon \operatorname{sign} \nabla_{\mathbf{x}} J(\hat{f}(\mathbf{x}), \mathbf{y})$.

However, one might have recognized that we assumed $J(\hat{f}(\mathbf{x}+\mathbf{v}), \mathbf{y}) \in \overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x}, \epsilon)$ in eq. (3.14), whereas we actually would have to consider the constraint $J(\hat{f}(\mathbf{x}+\mathbf{v}), \mathbf{y}) \in \overline{\mathcal{B}}_{\mathcal{X}}(\mathbf{x}, \epsilon)$. The reason for not doing so is that the actually valid set of points for $\mathbf{v}$ does not necessarily have the same properties we exploited above as $\overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{0}, \epsilon)$. To be precise, $\mathbf{v}$ would have to be constrained as follows:

$$\mathbf{v} \in \mathcal{V}^* \quad \text{with} \quad \mathcal{V}^* = \left\{ \mathbf{v}^* \in \overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{0}, \epsilon) \;\middle|\; \mathbf{x} + \mathbf{v}^* \in \mathcal{X} \right\}. \tag{3.16}$$

While there is no immediate closed form solution to eq. (3.15) with the changed constraint $\mathbf{v} \in \mathcal{V}^*$, it can be shown that we can simply project $\mathbf{x}+\mathbf{v}$ back onto the input space $\mathcal{X}$ and therefore maximize the linearized loss function in the set of valid FGSM perturbations

$$\mathcal{X}^* = \overline{\mathcal{B}}_{\mathcal{X}}(\mathbf{x}, \epsilon) = \overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x}, \epsilon) \cap \mathcal{X} = \left\{ \mathbf{x}' \in \mathcal{X} \;\middle|\; \mathbf{x}' - \mathbf{x} \in \overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{0}, \epsilon) \right\}. \tag{3.17}$$

Note here that $\mathcal{X}^*$ is in principle equivalent to $\mathcal{V}^*$, but shifted by $\mathbf{x}$. Geometrically speaking, the sets $\mathcal{V}^*$ and $\mathcal{X}^*$ represent a hyperrectangle, but not necessarily a hypercube as for $\overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{0}, \epsilon)$ and $\overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x}, \epsilon)$ (see fig. 3.8 as well).

The aforementioned maximization through projection is possible because the input space $\mathcal{X}$ is a closed ℓ_∞ ball itself, so in every dimension, the corresponding boundaries of $\mathcal{X}$, $\overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x}, \epsilon)$, $\mathcal{V}^*$ and $\mathcal{X}^*$ are parallel to each other. As a consequence, for all dimensions $i \in \{1, 2, \ldots, n\}$ in which $\overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x}, \epsilon)$ exceeds $\mathcal{X}$, the component v_i^* of $\mathbf{v}^* = \Pi_{\mathcal{X}}\left(\mathbf{x} + \epsilon \operatorname{sign} \nabla_{\mathbf{x}} J(\hat{f}(\mathbf{x}), \mathbf{y})\right) - \mathbf{x}$ is minimal in $\mathcal{V}^*$, if $\frac{\partial J(\hat{f}(\mathbf{x}),\mathbf{y})}{\partial x_i} < 0$, and maximal in $\mathcal{V}^*$, if $\frac{\partial J(\hat{f}(\mathbf{x}),\mathbf{y})}{\partial x_i} > 0$. If $\frac{\partial J(\hat{f}(\mathbf{x}),\mathbf{y})}{\partial x_i} = 0$, the dimension i does not contribute to the maximization in 3.15 anyway, no matter how v_i^* is chosen. In other, non-exceeding dimensions $j \in \{1, 2, \ldots, n\}$ with $j \neq i$, (a) $\operatorname{sign} v_j^* = \operatorname{sign} \frac{\partial J(\hat{f}(\mathbf{x}),\mathbf{y})}{\partial x_j}$ and (b) $|v_j^*| = \epsilon$ still hold. Thus, $\mathbf{v}^*$ is an optimal solution to eq. (3.15) with the corrected constraint stated in eq. (3.16).

To the end of this section, we briefly discuss a generalization of the FGSM that is constrained by an arbitrary p-metric, called the Fast Gradient Method (FGM) [48]. The basic algorithm is the same as for FGSM, what differs is again the constraint in eq. (3.15): Instead of restricting $\mathbf{v}$ to $\mathcal{V}^*$ as stated in eq. (3.16), it is formulated as follows:

$$\mathbf{v} \in \mathcal{V}_p^* \quad \text{with} \quad \mathcal{V}_p^* = \left\{ \mathbf{v}^* \in \overline{\mathcal{B}}_{\ell_p^{(n)}}(\mathbf{0}, \epsilon) \;\middle|\; \mathbf{x} + \mathbf{v}^* \in \mathcal{X} \right\}. \tag{3.18}$$

Then, solving the changed optimization problem in eq. (3.15) provides the formula to compute the perturbation vector w.r.t. the chosen p-metric. As we focus on the ∞-metric in this work, we do not attempt to solve it here for p-metrics apart from the ∞-metric. But when imagining the resulting intersection shapes e.g. for closed ℓ_1 or ℓ_2 balls, which are essentially convex polytopes and spherical segments, the question arises if it is tractable at all. Hence, one may need to go beyond linearization with further approximations and heuristics.

3.4.2. *Projected Gradient Descent*

Projected Gradient Descent (PGD)[6] is another first-order white-box adversarial attack, i.e., it exploits the gradients of the loss function J in the input space $\mathcal{X}$, just like FGSM. It was first used by Kurakin, Goodfellow, and Bengio [49] in 2016 as "basic iterative method". Later, Madry et al. [21] reused it for adversarial training and referred to it with PGD, since it is actually the same as the eponymous technique in constrained optimization [50].

One difference between the previously presented FGSM and PGD is in the motivation of usage: FGSM was primarily designed to create perturbations easily and quickly that are frequently adversarial examples. PGD, on the other hand, is more sophisticated and needs

[6] The naming "Projected Gradient Descent" originates from the fact that in practice, optimization problems are formulated as minimization problems in most cases, and has therefore historical reasons. As we rather maximize the loss function, the correct term would actually be "Projected Gradient Ascent".

more computational resources, but provides adversarial examples in the vast majority of cases. That aside, PGD as we use it in this work[7] exhibits basically three differences to the FGSM, which are listed in the following. Note here that we only consider PGD constrained to the ∞-metric in the scope of this work, while it is in principle also possible to use p-metrics apart from that.

1. PGD is not a single-step, but a multi-step procedure. Broadly speaking, it performs several perturbation steps of FGSM, each of which starts at the result of the previous iteration. Nevertheless, the steps themselves are slightly different, compared with FGSM.

2. PGD perturbations are still constrained by the closed ℓ_∞ ball $\overline{\mathcal{B}}_{\mathcal{X}}(\mathbf{x}, \epsilon)$. A single PGD step, however, is initially not scaled to ϵ as for FGSM, but to a lower *PGD step size* η'. Subsequently, the resulting perturbed image is first projected back onto the closed ℓ_∞ ball $\overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x}, \epsilon)$, then onto the input space $\mathcal{X}$, before the next step is performed.

3. PGD is not initialized with the natural input data $\mathbf{x}$, but with a point that is randomly sampled from a uniform distribution over the closed ℓ_∞ ball $\overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x}, \epsilon)$.

Formalizing the above aspects, we denote the perturbed version of the original image $\mathbf{x}$ after the i-th PGD step by $\mathbf{x'^{(i)}}$. Moreover, let $\mathbf{x'^{(0)}}$ be the random initialization data. Then, the perturbation procedure of PGD can be expressed as follows:

$$\mathbf{x'^{(i)}} = \Pi_{\mathcal{X}} \left(\Pi_{\overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x}, \epsilon)} \left(\mathbf{x'^{(i-1)}} + \eta' \operatorname{sign} \nabla_{\mathbf{x}} J(\hat{f}(\mathbf{x}), \mathbf{y}) \right) \right). \tag{3.19}$$

To sample the point $\mathbf{x'^{(0)}}$ from a uniform distribution over $\overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x}, \epsilon)$, we simply sample every component from an independent and identically distributed (iid) uniform distribution over the interval $[-\epsilon, \epsilon]$ and eventually shift it by the corresponding component of $\mathbf{x}$:

$$\forall j \in \{1, 2, \ldots, n\} : \left(x_j'^{(0)} - x_j \right) \sim \mathcal{U}(-\epsilon, \epsilon). \tag{3.20}$$

To the end of this section, we give reasons and plausibility why PGD works, as we did for FGSM. First, concerning the addition of $\eta' \operatorname{sign} \nabla_{\mathbf{x}} J(\hat{f}(\mathbf{x}), \mathbf{y})$ to $\mathbf{x'^{(i-1)}}$, we refer to section 3.4.1, where we explained the approximation of the loss function J by its first-order Taylor expansion. The result of that addition is necessarily part of the closed ℓ_∞ ball $\overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x'^{(i-1)}}, \eta')$.

[7] There might be variants which differ in some implementation details from our implementation, e.g., when and how often the projection back onto the input space $\mathcal{X}$ is performed.

Next, the ultimate goal of the i-th PGD step's projection part is to project any such point back onto the intersection of the closed ℓ_∞ ball $\overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x}, \epsilon)$ and the input space $\mathcal{X}$, where the latter is a closed ℓ_∞ ball as well. This intersection is not empty, since $\mathbf{x}$ is necessarily part of both $\overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x}, \epsilon)$ and $\mathcal{X}$. Hence, the Euclidean projection onto the intersection $\overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x}, \epsilon) \cap \mathcal{X}$ is equal to any order of successive Euclidean projections onto $\overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x}, \epsilon)$ and $\mathcal{X}$, which has been proven in appendix A.3.

Furthermore, it is to be clarified if $\mathbf{x'^{(i)}}$ is an optimal, loss-maximized solution in the set of valid PGD perturbed data

$$\mathcal{X}^* = \overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x'^{(i-1)}}, \eta') \cap \overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x}, \epsilon) \cap \mathcal{X} \tag{3.21}$$

after the i-th PGD step, i.e., if the following holds:

$$\mathbf{x'^{(i)}} = \underset{\mathbf{x}^* \in \mathcal{X}^*}{\arg\max}\, J(\hat{f}(\mathbf{x}^*), \mathbf{y}). \tag{3.22}$$

Unfortunately, due to the observed non-concavity of a DNN loss function, this cannot be guaranteed. But what can actually be guaranteed is that $\mathbf{x'^{(i)}}$ is an optimal solution in $\mathcal{X}^*$ for maximizing the loss function J that is linearized in the vicinity of $\mathbf{x'^{(i-1)}}$. As for FGSM, this is because $\overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x'^{(i-1)}}, \eta')$, $\overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x}, \epsilon)$ and $\mathcal{X}$ are all closed ℓ_∞ balls, having boundaries that are parallel to those of each other as well as their intersection $\mathcal{X}^*$. The rest follows the argumentation in section 3.4.1. Viewed over all iterations, PGD is thus able to find a local maximum of the loss function J in the closed ℓ_∞ ball $\overline{\mathcal{B}}_{\mathcal{X}}(\mathbf{x}, \epsilon)$.

Last, we discuss the decision to use random initialization. We assume that the further away a point is from the natural image $\mathbf{x}$, the more likely it is that the image that corresponds to this point is misclassified. At the same time, we have to consider that PGD is constrained by the closed ℓ_∞ ball $\overline{\mathcal{B}}_{\mathcal{X}}(\mathbf{x}, \epsilon)$. So because of the general non-concavity of the loss function J, the initialization point $\mathbf{x'^{(0)}}$ should be close to the feasible area $\overline{\mathcal{B}}_{\mathcal{X}}(\mathbf{x}, \epsilon)$ in order to ensure that the first PGD step is not pointless after projection. Assuming that ϵ is relatively small, we can initialize in $\overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x}, \epsilon)$ without taking this risk. Not initializing in $\overline{\mathcal{B}}_{\mathcal{X}}(\mathbf{x}, \epsilon)$ in order to reduce the risk further has mostly practical reasons, since dimensions would have to be sampled independently, while the benefit would probably be marginal.

Now it depends on which metric is used to measure the distance between a possible initialization point $\mathbf{x'^{(0)}}$ and the natural image $\mathbf{x}$. The Euclidean metric, e.g., would imply to initialize at one of the 2^n corners of $\overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x}, \epsilon)$, geometrically speaking. However, in the scope of this work, we resort to the ∞-metric as well, yet admiring that it would be interesting to investigate the influence of this more restrictive initialization scheme on the efficiency of PGD. In consequence, when sampling from a uniform distribution

over $\overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x}, \epsilon)$, the distance is almost guaranteed to be maximized for large n, as the probability raises that at least one dimension is close to either $-\epsilon$ or ϵ. This procedure also coincides with the original formulation by Madry et al. [21].

3.4.3. *Boundary Attack*

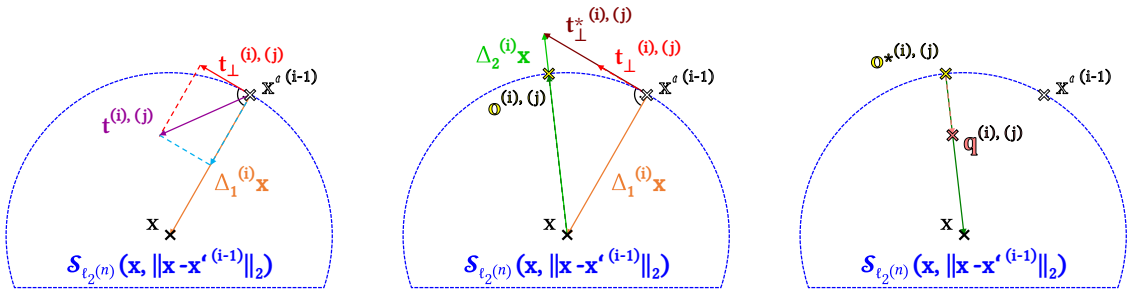
The Boundary Attack was published by Brendel, Rauber, and Bethge [51] in 2018. Opposite to the previously presented adversarial attacks, it operates in a black-box setting. Therefore, it does neither require any knowledge on the architecture, nor on the trained parameters $\boldsymbol{\theta}$ of the DNN to be attacked. However, what it does still need is access to the model $\hat{f}_c$ as a black-box $\hat{f}_c : \mathcal{X} \to \mathcal{C}$, i.e., feeding input data $\mathbf{x} \in \mathcal{X}$ into it and receiving the classification result $\hat{c} \in \mathcal{C}$. Moreover, it requires the availability of so-called *target data*. This is natural images $\mathbf{x}$ and corresponding ground truth data c being sampled from the data distribution ρ_{data} of the classification task the DNN was trained on. It does not necessarily have to be classified correctly by the model, although the question arises, whether performing the Boundary Attack is necessary at all in the case of misclassification.

In simplified terms, the Boundary Attack performs a random walk in the direction of the target image $\mathbf{x}$, being initialized at a random, but adversarial point $\mathbf{x}'^{(\mathbf{0})} \in \mathcal{X}$ and staying adversarial while reducing the distance to $\mathbf{x}$. By a point "being adversarial", we mean that the DNN classifies the corresponding image wrongly. Here, the ground truth data c of the target image $\mathbf{x}$ is used throughout as a reference.

Another major difference to FGSM and PGD is that the Boundary Attack operates w.r.t. the Euclidean metric, not the ∞-metric. In consequence, the ℓ_2 distance and not (necessarily) the ℓ_∞ distance to the target image $\mathbf{x}$ is reduced, as the attack progresses.

While the Boundary Attack is of iterative nature, a single step can be further subdivided into an *orthogonal substep* and a *minimization substep*. Let $\mathbf{x}'^{(\mathbf{i})}$ denote the perturbed data after the i-th of N iterations of the Boundary Attack. Then, in the i-th iteration and starting from the point $\mathbf{x}'^{(\mathbf{i-1})}$, M random orthogonal substeps are performed on the ℓ_2 sphere $\mathcal{S}_{\ell_2^{(n)}}(\mathbf{x}, \|\mathbf{x} - \mathbf{x}'^{(\mathbf{i-1})}\|_2)$, resulting in the *orthogonal sample candidates* $\mathbf{o}^{(\mathbf{i}),(\mathbf{j})}$ with $j \in \{1, 2, \ldots, M\}$. Here, $M \in \mathbb{N}_{\geq 1}$ is a parameter of the algorithm that is to be set beforehand, just as for the number of iterations $N \in \mathbb{N}_{\geq 1}$. Last, these random orthogonal substeps are also limited in their size by some parameter $\sigma_1^{(i)} \in \mathbb{R}_{>0}$, whose concrete function and computation will be explained later on.

By choosing the orthogonal sample candidates as described, the ℓ_2 distance between the target image $\mathbf{x}$ and all orthogonal sample candidates $\mathbf{o}^{(\mathbf{i}),(\mathbf{j})}$ is initially kept constant, equalling the ℓ_2 distance between $\mathbf{x}$ and $\mathbf{x}'^{(\mathbf{i-1})}$. However, as an orthogonal sample candidate $\mathbf{o}^{(\mathbf{i}),(\mathbf{j})}$ is not necessarily part of the input space $\mathcal{X}$, it has to be projected back onto the latter:



(a) Orthogonal substep (I) **(b)** Orthogonal substep (II) **(c)** Minimization substep

Figure 3.9.: Orthogonal substep and minimization substep of the Boundary Attack in two dimensions. For simplification, we assume here that the orthogonal sample candidate $\mathbf{o}^{(\mathrm{i}),(\mathrm{j})}$ is part of the input space $\mathcal{X}$ and therefore equal to its projected version $\mathbf{o}^{*(\mathrm{i}),(\mathrm{j})}$.

$$\forall j \in \{1, 2, \ldots, M\} : \mathbf{o}^{*(\mathrm{i}),(\mathrm{j})} = \Pi_{\mathcal{X}}\left(\mathbf{o}^{(\mathrm{i}),(\mathrm{j})}\right). \tag{3.23}$$

Doing so, its ℓ_2 distance to the target image $\mathbf{x}$ can change, but should still be relatively close to the previous one.

Next, in the minimization substep of the i-iteration, a step in the direction of the target image $\mathbf{x}$ is performed from each projected orthogonal sample candidate $\mathbf{o}^{*(\mathrm{i}),(\mathrm{j})}$, resulting in the *minimization sample candidates* $\mathbf{q}^{(\mathrm{i}),(\mathrm{j})}$. These taken steps have all a Euclidean norm of $\sigma_2^{(i)} \in \mathbb{R}_{>0}$, which is another parameter that is to be seen similar to $\sigma_1^{(i)}$ for now. Thus, the minimization sample candidates $\mathbf{q}^{(\mathrm{i}),(\mathrm{j})}$ are computed as follows:

$$\forall j \in \{1, 2, \ldots, M\} : \mathbf{q}^{(\mathrm{i}),(\mathrm{j})} = \mathbf{o}^{*(\mathrm{i}),(\mathrm{j})} + \frac{\sigma_2^{(i-1)}}{\|\mathbf{x} - \mathbf{o}^{*(\mathrm{i}),(\mathrm{j})}\|_2}\left(\mathbf{x} - \mathbf{o}^{*(\mathrm{i}),(\mathrm{j})}\right). \tag{3.24}$$

Since both all projected orthogonal sample candidates $\mathbf{o}^{*(\mathrm{i}),(\mathrm{j})}$ and the target image $\mathbf{x}$ are part of the input space $\mathcal{X}$ and the latter – as an ℓ_∞ ball – is a convex set, we do not have to project a minimization sample candidate $\mathbf{q}^{(\mathrm{i}),(\mathrm{j})}$ back onto the input space $\mathcal{X}$. Figure 3.9 visualizes the orthogonal substep and the minimization substep for a random sample direction j in the i-th iteration.

Eventually, the images that correspond to the minimization sample candidates $\mathbf{q}^{(\mathrm{i}),(\mathrm{j})}$ are all passed through the DNN. Those being classified correctly are discarded; if there are any remaining, an arbitrary minimization sample candidate $\mathbf{r}^{(\mathrm{i}),(\mathrm{j}^*)}$ is chosen as $\mathbf{x}'^{(\mathrm{i})}$, being the starting point of the next iteration. In this case, $\mathbf{q}^{*(\mathrm{i}),(\mathrm{j}^*)} - \mathbf{x}'^{(\mathrm{i}-1)}$ and $\mathbf{r}^{(\mathrm{i}),(\mathrm{j}^*)} - \mathbf{q}^{*(\mathrm{i}),(\mathrm{j}^*)}$ can be considered as the final orthogonal and minimization substeps taken in the i-th iteration, respectively. If there are not any misclassified samples, in turn, $\mathbf{x}'^{(\mathrm{i})}$ is set to $\mathbf{x}'^{(\mathrm{i}-1)}$.

In the following, we show how the orthogonal sample candidates $\mathbf{o}^{(\mathrm{i}),(\mathrm{j})}$ are computed in detail, starting from $\mathbf{x}'^{(\mathrm{i}-1)}$. First of all, we define

$$\Delta_1^{(\mathbf{i})}\mathbf{x} = \mathbf{x} - \mathbf{x}'^{(\mathbf{i}-\mathbf{1})} \tag{3.25}$$

as the difference vector between the target image $\mathbf{x}$ and the perturbed data as well as starting point in the i-th iteration $\mathbf{x}'^{(\mathbf{i}-\mathbf{1})}$. Then, we sample $M \times n$ times from an iid Gaussian distribution $\mathcal{N}(0,1)$:

$$\forall j \in \{1, 2, \ldots, M\} : t_1^{(i),(j)}, t_2^{(i),(j)}, \ldots, t_n^{(i),(j)} \sim \mathcal{N}(0,1). \tag{3.26}$$

Subsequently, for all $\mathbf{t}^{(\mathbf{i}),(\mathbf{j})}$, we compute the vector rejection $\mathbf{t}_\perp^{(\mathbf{i}),(\mathbf{j})}$ of $\mathbf{t}^{(\mathbf{i}),(\mathbf{j})}$ w.r.t. the difference vector $\Delta_1^{(\mathbf{i})}\mathbf{x}$:

$$\forall j \in \{1, 2, \ldots, M\} : \mathbf{t}_\perp^{(\mathbf{i}),(\mathbf{j})} = \mathbf{t}^{(\mathbf{i}),(\mathbf{j})} - \frac{\mathbf{t}^{(\mathbf{i}),(\mathbf{j})\,\mathsf{T}}\Delta_1^{(\mathbf{i})}\mathbf{x}}{\left(\Delta_1^{(\mathbf{i})}\mathbf{x}\right)^{\mathsf{T}}\Delta_1^{(\mathbf{i})}\mathbf{x}}\Delta_1^{(\mathbf{i})}\mathbf{x}. \tag{3.27}$$

Geometrically speaking, $\mathbf{t}_\perp^{(\mathbf{i}),(\mathbf{j})}$ is the vector that is orthogonal to $\Delta_1^{(\mathbf{i})}\mathbf{x}$ and that combines with the projection of $\mathbf{t}^{(\mathbf{i}),(\mathbf{j})}$ onto $\Delta_1^{(\mathbf{i})}\mathbf{x}$ to $\Delta_1^{(\mathbf{i})}\mathbf{x}$ itself. Actually, however, we only need the first property, since in the next step, we rescale $\mathbf{t}_\perp^{(\mathbf{i}),(\mathbf{j})}$ by $\sigma_1^{(i-1)}$ times the Euclidean norm of $\Delta_1^{(\mathbf{i})}\mathbf{x}$:

$$\forall j \in \{1, 2, \ldots, M\} : \mathbf{t}^{*}{}_\perp^{(\mathbf{i}),(\mathbf{j})} = \frac{\sigma_1^{(i-1)}\|\Delta_1^{(\mathbf{i})}\mathbf{x}\|_2}{\|\mathbf{t}_\perp^{(\mathbf{i}),(\mathbf{j})}\|_2}\mathbf{t}_\perp^{(\mathbf{i}),(\mathbf{j})}. \tag{3.28}$$

After that, we define

$$\forall j \in \{1, 2, \ldots, M\} : \Delta_2^{(\mathbf{i}),(\mathbf{j})}\mathbf{x} = \mathbf{t}^{*}{}_\perp^{(\mathbf{i}),(\mathbf{j})} - \mathbf{x} \tag{3.29}$$

as the difference vector between $\mathbf{t}^{*}{}_\perp^{(\mathbf{i}),(\mathbf{j})}$ and the target image $\mathbf{x}$.

Now, applying the Pythagorean theorem, we can compute the ratio between the Euclidean norms of $\Delta_1^{(\mathbf{i})}\mathbf{x}$ and $\Delta_2^{(\mathbf{i}),(\mathbf{j})}\mathbf{x}$:

$$\begin{aligned}
\forall j \in \{1, 2, \ldots, M\} : \|\Delta_2^{(\mathbf{i}),(\mathbf{j})}\mathbf{x}\|_2 &= \sqrt{\|\Delta_1^{(\mathbf{i})}\mathbf{x}\|_2^2 + \|\mathbf{t}^{*}{}_\perp^{(\mathbf{i}),(\mathbf{j})}\|_2^2} \\
&= \sqrt{\|\Delta_1^{(\mathbf{i})}\mathbf{x}\|_2^2 + {\sigma_1^{(i-1)}}^2\|\Delta_1^{(\mathbf{i})}\mathbf{x}\|_2^2} \\
&= \sqrt{\left({\sigma_1^{(i-1)}}^2 + 1\right)\|\Delta_1^{(\mathbf{i})}\mathbf{x}\|_2^2} \\
&= \sqrt{{\sigma_1^{(i-1)}}^2 + 1}\;\|\Delta_1^{(\mathbf{i})}\mathbf{x}\|_2 \\
&\Updownarrow \\
\forall j \in \{1, 2, \ldots, M\} : \frac{\|\Delta_1^{(\mathbf{i})}\mathbf{x}\|_2}{\|\Delta_2^{(\mathbf{i}),(\mathbf{j})}\mathbf{x}\|_2} &= \frac{1}{\sqrt{{\sigma_1^{(i-1)}}^2 + 1}}.
\end{aligned} \tag{3.30}$$

Algorithm 1 Boundary Attack

Parameters

$\mathbf{x}$	Target image
N	Number of iterations
M	Number of computed candidate samples per iteration

function BoundaryAttack($\mathbf{x}$, N, M)
  $\sigma_1^{(0)} \leftarrow 0.01$ ▹ Initialization
  $\sigma_2^{(0)} \leftarrow 0.01$

  repeat
    $x_1'^{(0)}, \ldots, x_n'^{(0)} \sim \mathcal{U}(0, 1)$
  until $\hat{f}_c(\mathbf{x'^{(0)}}) = c$

  for $i = 1, \ldots, N$ **do**
    $\mathbf{x'^{(i)}} \leftarrow \mathbf{x'^{(i-1)}}$
    $\boldsymbol{\Delta}_1^{(\mathbf{i})}\mathbf{x} = \mathbf{x} - \mathbf{x'^{(i-1)}}$

    for $j = 1, \ldots, M$ **do**
      $t_1^{(i),(j)}, t_2^{(i),(j)}, \ldots, t_n^{(i),(j)} \sim \mathcal{N}(0, 1)$ ▹ Orthogonal substep
      $\mathbf{t}_\perp^{(\mathbf{i}),(\mathbf{j})} \leftarrow \mathbf{t}^{(\mathbf{i}),(\mathbf{j})} - \frac{\mathbf{t}^{(\mathbf{i}),(\mathbf{j})\,\intercal}\boldsymbol{\Delta}_1^{(\mathbf{i})}\mathbf{x}}{\left(\boldsymbol{\Delta}_1^{(\mathbf{i})}\mathbf{x}\right)^\intercal \boldsymbol{\Delta}_1^{(\mathbf{i})}\mathbf{x}}\boldsymbol{\Delta}_1^{(\mathbf{i})}\mathbf{x}$
      $\mathbf{t}^{*(\mathbf{i}),(\mathbf{j})}_\perp \leftarrow \frac{\sigma_1^{(i-1)}\|\boldsymbol{\Delta}_1^{(\mathbf{i})}\mathbf{x}\|_2}{\|\mathbf{t}_\perp^{(\mathbf{i}),(\mathbf{j})}\|_2}\mathbf{t}_\perp^{(\mathbf{i}),(\mathbf{j})}$
      $\boldsymbol{\Delta}_2^{(\mathbf{i}),(\mathbf{j})}\mathbf{x} \leftarrow \mathbf{t}^{*(\mathbf{i}),(\mathbf{j})}_\perp - \mathbf{x}$
      $\mathbf{o}^{(\mathbf{i}),(\mathbf{j})} \leftarrow \mathbf{x} + \frac{1}{\sqrt{\sigma_1^{(i-1)\,2}+1}}\boldsymbol{\Delta}_2^{(\mathbf{i}),(\mathbf{j})}\mathbf{x}$
      $\mathbf{o}^{*(\mathbf{i}),(\mathbf{j})} \leftarrow \Pi_{\mathcal{X}}\left(\mathbf{o}^{(\mathbf{i}),(\mathbf{j})}\right)$
      $\mathbf{q}^{(\mathbf{i}),(\mathbf{j})} \leftarrow \mathbf{o}^{*(\mathbf{i}),(\mathbf{j})} + \frac{\sigma_2^{(i-1)}}{\|\mathbf{x}-\mathbf{o}^{*(\mathbf{i}),(\mathbf{j})}\|_2}\left(\mathbf{x} - \mathbf{o}^{*(\mathbf{i}),(\mathbf{j})}\right)$ ▹ Minimization substep
    end for

    for $j = 1, \ldots, M$ **do** ▹ Receiving $\mathbf{x'^{(i)}}$
      if $\hat{f}_c(\mathbf{q}^{(\mathbf{i}),(\mathbf{j})}) \neq c$ **then**
        $\mathbf{x'^{(i)}} \leftarrow \mathbf{q}^{(\mathbf{i}),(\mathbf{j})}$
        break
      end if
    end for

    $\psi_1^{(i)} \leftarrow \#\{j \in \{1, \ldots, M\} \mid \hat{f}_c(\mathbf{o}^{*(\mathbf{i}),(\mathbf{j})}) \neq c\}/M$ ▹ Computing $\psi_1^{(i)}, \psi_2^{(i)}$
    $\psi_2^{(i)} \leftarrow \#\{j \in \{1, \ldots, M\} \mid \hat{f}_c(\mathbf{q}^{(\mathbf{i}),(\mathbf{j})}) \neq c\}/M$

```
        if ψ_1^(i) > 0.5 then                                  ▷ Adapting σ_1^(i), σ_2^(i)
            σ_1^(i) ← 3/2 σ_1^(i−1)
        else if 0.2 ≤ ψ_1^(i) ≤ 0.5 then
            σ_1^(i) ← σ_1^(i−1)
        elseψ_1^(i) < 0.2
            σ_1^(i) ← 2/3 σ_1^(i−1)
        end if

        if ψ_2^(i) > 0.25 then
            σ_2^(i) ← 3/2 σ_2^(i−1)
        else if 0.1 ≤ ψ_2^(i) ≤ 0.25 then
            σ_2^(i) ← σ_2^(i−1)
        elseψ_2^(i) < 0.1
            σ_2^(i) ← 2/3 σ_2^(i−1)
        end if
    end for

    x′ ← x′^(N)                                                ▷ Final adversarial example
    return x′
end function
```

Eventually, we simply scale $\mathbf{\Delta}_2^{(\mathbf{i}),(\mathbf{j})}\mathbf{x}$ by this ratio and add the result to $\mathbf{x}$. Thus, we obtain the orthogonal sample candidate $\mathbf{o}^{(\mathbf{i}),(\mathbf{j})}$ that has the same ℓ_2 distance to $\mathbf{x}$ as $\mathbf{x'}^{(\mathbf{i-1})}$:

$$\forall j \in \{1, 2, \ldots, M\} : \mathbf{o}^{(\mathbf{i}),(\mathbf{j})} = \mathbf{x} + \frac{1}{\sqrt{\sigma_1^{(i-1)^2} + 1}} \mathbf{\Delta}_2^{(\mathbf{i}),(\mathbf{j})}\mathbf{x}. \tag{3.31}$$

To the end of this section, we discuss how the parameters $\sigma_1^{(i)}$ and $\sigma_2^{(i)}$ are computed. Actually, they are not computed by some complex function, but changed relatively to their values $\sigma_1^{(i-1)}$ and $\sigma_2^{(i-1)}$ during the previous iteration. This is done at the end of each iteration.

According to Brendel, Rauber, and Bethge, "the adjustment is inspired by Trust Region methods" [51]. Essentially, the changing depends on the *adversarial ratio* w.r.t. the projected orthogonal sample candidates $\mathbf{o}^{*(\mathbf{i}),(\mathbf{j})}$ and the minimization sample candidates $\mathbf{q}^{(\mathbf{i}),(\mathbf{j})}$, respectively. By "adversarial ratio", we mean the ratio of misclassified projected orthogonal sample candidates $\mathbf{o}^{*(\mathbf{i}),(\mathbf{j})}$/misclassified minimization sample candidates $\mathbf{q}^{(\mathbf{i}),(\mathbf{j})}$ to the total number of samples M. Denoting the first ratio by $\psi_1^{(i)}$ and the second one by $\psi_2^{(i)}$, $\sigma_1^{(i)}$ and $\sigma_2^{(i)}$ are computed by

$$\sigma_1^{(i)} = \begin{cases} \frac{3}{2}\sigma_1^{(i-1)}, & \text{for } 0.5 < \psi_1^{(i)} \leq 1 \\ \sigma_1^{(i-1)}, & \text{for } 0.2 \leq \psi_1^{(i)} \leq 0.5 \\ \frac{2}{3}\sigma_1^{(i-1)}, & \text{for } 0 \leq \psi_1^{(i)} < 0.2 \end{cases}, \tag{3.32}$$

$$\sigma_2^{(i)} = \begin{cases} \frac{3}{2}\sigma_2^{(i-1)}, & \text{for } 0.25 < \psi_1^{(i)} \leq 1 \\ \sigma_2^{(i-1)}, & \text{for } 0.1 \leq \psi_1^{(i)} \leq 0.25 \\ \frac{2}{3}\sigma_2^{(i-1)}, & \text{for } 0 \leq \psi_1^{(i)} < 0.1 \end{cases}. \tag{3.33}$$

Both the thresholds and the change rates coincide with the reference implementation of the Boundary Attack. The same applies to the initial values, which are set to $\sigma_1^{(0)} = 0.01, \sigma_2^{(0)} = 0.01$ [52].

The above expected ranges can be interpreted as follows [51]:

1. **Orthogonal substep**: Ideally, the orthogonal substeps are small enough to locally treat the *decision boundary*, i.e., the boundary between the wrongly classified region and the correctly classified region in the input space, as approximately linear. In that case, roughly 50 % of the (projected) orthogonal sample candidates are expected to be misclassified. If the adversarial ratio $\psi_1^{(i)}$ is higher, this is mostly because the substeps are too small to get sufficiently close to the decision boundary, so $\sigma_1^{(i)}$ is increased. If it is significantly lower, however, the substeps are too large, so $\sigma_1^{(i)}$ is decreased. Note that the former does not necessarily have to be done, but increases the efficiency of the algorithm, so the threshold is set tightly to exactly 0.5. The latter, on the contrary, *has* ultimately to be done to ensure further progress in the following iterations, but simultaneously decreases the efficiency. In order to maintain a sufficiently high efficiency at least in the short term, the second threshold is set more loosely to 0.2, taking the risk of being too close to the decision boundary and running into a dead end, figuratively speaking.

2. **Minimization substep**: About the same applies here, i.e., $\sigma_2^{(i)}$ is increased in case of a too high adversarial ratio $\psi_2^{(i)}$ and decreased in the opposite case. The upper threshold to keep it constant, however, cannot be derived logically as for the linearity assumption above. As a consequence, both the upper and the lower threshold have to be set intuitively this time. In comparison to the orthogonal substep, an even higher risk is taken here to make much progress in relatively few iterations.

Last, algorithm 1 shows the entire algorithm of the Boundary Attack.

3.5. ADVERSARIAL TRAINING

Adversarial training is a special training technique for DNNs to reduce the susceptibility to adversarial attacks as well as adversarial examples in general. Basically, it tries to *make the problem part of the solution*: Either in addition to [20] or completely instead of the natural training data $\mathbf{x}$ in $\hat{\rho}_{\text{data}}$ [21], adversarial examples $\mathbf{x}'$ are created and fed into the training process. In the scope of this work, we concentrate on the latter approach.

The idea behind the adversarial training is similar to that of classical data augmentation; the natural training data $\mathbf{x}$ is manipulated by some semantics-preserving transformations to increase the diversity of the training data and eventually increase the robustness of the trained DNN (see section 3.3.1). However, while these transformations in classical data augmentation are usually static transformations on image data such as rotation or reflection, increasing the robustness in an unspecific manner, adversarial attacks represent rather dynamic transformations, which specifically aim to attack the model at its flaws [20]. In consequence, adversarial training allows a continuous improvement at the weak points, at least if the adversarial examples used for training as continuously generated afresh, based on the current state of the model. As for the preceding data augmentation, we create new adversarial examples dynamically per training batch when the implementing adversarial training in this work. Doing so, they are always maximally effective against the DNN in the current state of training. Adversarial examples that were created solely beforehand, by contrast, would lose their effectiveness once the DNN has adapted to them – the gain of general robustness would be marginal at best.

In the following, we present two different concrete adversarial training methods, which are both based on the FGSM. At this point, we briefly explain why not to consider the stronger PGD attack as well. Actually, since PGD is a stronger adversarial attack, robustness increase would probably be higher when using it, but this comes at a high price: Being a multi-step procedure, PGD performs several consecutive forward and backward passes in order to create an adversarial example. Unfortunately, being based on each other, these cannot be parallelized. As a consequence, when neglecting the remaining, comparatively low overhead, PGD is roughly T times slower than FGSM, when T denotes the number of PGD steps. Using the dynamic generation approach, this slowdown ultimately extends to the entire training. Hence, due to the too high computational cost, we decided to resort only to the more modest FGSM.

In concrete, the two presented methods are

(a) FGSM-based adversarial training and

(b) Stable Single Step (SSS)-based adversarial training.

To avoid confusion with the magnitude of perturbation ϵ in adversarial attacks later on, we also introduce here $\epsilon^* \in \mathbb{R}_{\geq 0}$ as the "defensive" magnitude of perturbation in adversarial training.

3.5.1. *Fast Gradient Sign Method-based*

Algorithm 2 FGSM-based adversarial training. The additional actions compared to the standard training process are highlighted in red. For simplicity, basic SGD is used to update the DNN parameters $\boldsymbol{\theta}$, and the batch size is set to 1.

Parameters

$\hat{\rho}_{\text{data}}$	Training dataset
$\boldsymbol{\theta}^{(0)}$	Initial DNN parameters
η	SGD learning rate
ϵ^*	Maximum magnitude of perturbation
B	Number of batches

function FGSMAdversarialTraining($\hat{\rho}_{\text{data}}$, $\boldsymbol{\theta}^{(0)}$, η, ϵ^*, B)
  for $i = 1, \ldots, B$ **do**
    $(\mathbf{x}, \mathbf{y}) \sim \hat{\rho}_{\text{data}}$
    $\mathbf{x}' \leftarrow \Pi_{\mathcal{X}} \left(\mathbf{x} + \epsilon^* \operatorname{sign} \nabla_{\mathbf{x}} J(\hat{f}(\mathbf{x}), \mathbf{y}) \right)$ ▹ FGSM
    $\mathbf{x} \leftarrow \mathbf{x}'$
    $\boldsymbol{\theta}^{(i)} \leftarrow \boldsymbol{\theta}^{(i-1)} - \eta \nabla_{\boldsymbol{\theta}} J(\hat{f}(\mathbf{x}), \mathbf{y})$
  end for
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(B)}$ ▹ Final DNN parameters
  return $\boldsymbol{\theta}$
end function

FGSM-based adversarial training, being the most basic approach of adversarial training, was first used by Goodfellow, Shlens, and Szegedy [20] in 2015 as well.

After setting ϵ^*, it simply creates FGSM-perturbed versions $\mathbf{x}'$ of the original training images $\mathbf{x}$ according to eq. (3.13), which are then fed into the training process. Note here that it is not checked whether $\mathbf{x}'$ is actually adversarial or not. Algorithm 2 shows the algorithm of FGSM-based adversarial training.

By training their DNNs in such a way, Goodfellow, Shlens, and Szegedy observed that the robustness against adversarial examples was significantly higher than with the standardly trained DNNs [20], marking virtually the "hour of birth" of adversarial training.

3.5.2. *Stable Single Step-based*

Stable Single Step (SSS)-based adversarial training is a very recent method, being published by Kim, Lee, and Lee [53] in May, 2021. As already mentioned, it also uses the FGSM to generate perturbed versions of a natural training image $\mathbf{x}$.

The main difference to the FGSM-based adversarial training is that it does not operate with a fixed magnitude of perturbation ϵ^*, but a flexible one ϵ_i^* in the half-open interval $(0, \epsilon^*]$. More specifically, the interval is discretized in an equidistant manner, i.e.,

$$\epsilon_i^* \in \left\{ \frac{1}{L}\epsilon^*, \frac{2}{L}\epsilon^*, \ldots, \epsilon^* \right\}, \tag{3.34}$$

where $L \in \mathbb{N}_{\geq 1}$ represents the level of discretization and thus the number of possible magnitudes of perturbation in the SSS-based adversarial training. It has to be set beforehand. Then, informally, the perturbed version of $\mathbf{x}$ that exhibits *the smallest magnitude of perturbation ϵ_i^* while being adversarial* is fed into the training process, the rest is discarded. In addition, just as for PGD, the SSS algorithm is not initialized at the natural image $\mathbf{x}$ itself, but some random point $\mathbf{x'^{(0)}}$ in the closed ℓ_∞ ball $\overline{\mathcal{B}}_{\mathcal{X}}(\mathbf{x}, \epsilon^*)$

Formalizing the aforementioned, we first sample a random initialization point $\mathbf{x'^{(0)}}$, which is done the same way as for PGD (see eq. (3.20)). After that, we compute the gradient $\nabla_{\mathbf{x}'^{(0)}} J(\hat{f}(\mathbf{x'^{(0)}}), \mathbf{y})$. Then, a perturbed version $\mathbf{x'^{(i)}}$ of the original input data $\mathbf{x}$ is created for each possible magnitude of perturbation ϵ_i^* using the FGSM:

$$\begin{gathered} \forall i \in \{1, 2, \ldots, L\} : \mathbf{x'^{(i)}} = \Pi_{\mathcal{X}} \left(\mathbf{x'^{(0)}} + \epsilon_i^* \operatorname{sign} \nabla_{\mathbf{x}'^{(0)}} J(\hat{f}(\mathbf{x'^{(0)}}), \mathbf{y}) \right) \\ \text{with} \\ \epsilon_i^* = \frac{i}{L}\epsilon^*. \end{gathered} \tag{3.35}$$

Subsequently, we check whether there is at least one adversarial example $\mathbf{x'^{(i)}}$ for $i \in \{1, 2, \ldots, L\}$. If there is, the perturbed data $\mathbf{x'^{(i)}}$ with the smallest magnitude of perturbation ϵ_i^* that is also adversarial is used as the actual adversarial example $\mathbf{x}'$ in adversarial training:

$$\begin{gathered} \exists i \in \{1, 2, \ldots, L\} : \hat{f}_c(\mathbf{x'^{(i)}}) \neq c \quad \rightarrow \quad \mathbf{x}' = \mathbf{x'^{(j)}} \\ \text{with} \\ j = \underset{k \in \{1,2,\ldots,L\}}{\arg\min}\ \epsilon_k^* \quad \textbf{s.t.} \quad \hat{f}_c(\mathbf{x'^{(k)}}) \neq c. \end{gathered} \tag{3.36}$$

If there is no adversarial example $\mathbf{x'^{(i)}}$ for all $i \in \{1, 2, \ldots, L\}$, the perturbed data $\mathbf{x'^{(L)}}$ with maximum magnitude of perturbation $\epsilon_M^* = \epsilon^*$ is used, as for the FGSM-based adversarial training:

Algorithm 3 SSS-based adversarial training. The additional actions compared to the standard training process are highlighted in red. For simplicity, basic SGD is used to update the DNN parameters $\boldsymbol{\theta}$, and the batch size is set to 1.

Parameters

$\hat{\rho}_{\text{data}}$	Training dataset
$\boldsymbol{\theta}^{(0)}$	Initial DNN parameters
η	SGD learning rate
ϵ^*	Maximum magnitude of perturbation
L	Level of discretization
B	Number of batches

function SSSAdversarialTraining($\hat{\rho}_{\text{data}}$, $\boldsymbol{\theta}^{(0)}$, η, ϵ^*, L, B)
  for $i = 1, \ldots, B$ **do**
    $(\mathbf{x}, \mathbf{y}) \sim \hat{\rho}_{\text{data}}$
    $c \longleftarrow= \arg\max_{j \in C} y_i$
    $u_1, \ldots, u_n \sim \mathcal{U}(-\epsilon^*, \epsilon^*)$
    $\mathbf{x}'^{(0)} \leftarrow \mathbf{x} + \mathbf{u}$
    $\mathbf{g} \leftarrow \operatorname{sign} \nabla_{\mathbf{x}'^{(0)}} J(\hat{f}(\mathbf{x}'^{(0)}), \mathbf{y})$
    $k \leftarrow L$
    for $j = 1, \ldots, L$ **do** ▹ Discretization
      $\epsilon_j^* \leftarrow \frac{j}{L}\epsilon^*$
      $\mathbf{x}'^{(\mathbf{j})} \leftarrow \Pi_{\mathcal{X}}\left(\mathbf{x}'^{(0)} + \epsilon_j^* \mathbf{g}\right)$ ▹ FGSM
      if $\hat{f}_c(\mathbf{x}'^{(\mathbf{j})}) \neq c$ **then**
        $k \leftarrow j$
        break
      end if
    end for
    $\mathbf{x}' \leftarrow \mathbf{x}'^{(\mathbf{k})}$ ▹ Final adversarial example
    $\mathbf{x} \leftarrow \mathbf{x}'$
    $\boldsymbol{\theta}^{(\mathbf{i})} \leftarrow \boldsymbol{\theta}^{(\mathbf{i-1})} - \eta \nabla_{\boldsymbol{\theta}} J(\hat{f}(\mathbf{x}), \mathbf{y})$
  end for
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(\mathbf{B})}$ ▹ Final DNN parameters
  return $\boldsymbol{\theta}$
end function

$$\nexists i \in \{1, 2, \ldots, L\} : \hat{f}_c(\mathbf{x}'^{(\mathbf{i})}) \neq c \quad \rightarrow \quad \mathbf{x}' = \mathbf{x}'^{(\mathbf{L})}. \tag{3.37}$$

Algorithm 2 shows the algorithm of SSS-based adversarial training.

Last, we expound the motivation for this procedure as well. While the FGSM-based adversarial training of DNNs has proven that it can actually increase their robustness against adversarial examples [20], it has also been shown that one has to be careful when using this method. In 2020, Wong, Rice, and Kolter [54] pointed out that there is a "failure mode" called *catastrophic overfitting*, referring to the following problem: After a certain number of completed epochs, in which the accuracy for both FGSM-perturbed and PGD-perturbed images steadily increased, the accuracy for the former suddenly raises up to nearly 100 % and the latter drops to about 0 %. They proposed *early stopping* in such a moment, i.e., saving the last model parameters $\boldsymbol{\theta}$ that did not lead to the described divide between both accuracies. However, as catastrophic overfitting occurs at a time when the natural validation accuracy is still improving, this kind of early stopping results in a trained model that is somewhat robust against adversarial examples, but falls far short of the classification accuracy of a standardly trained DNN when it comes to natural image data.

Kim, Lee, and Lee [53] then investigated the problem in detail, figuring out that this is to one part caused by the fixed magnitude of perturbation ϵ^* used by the FGSM in adversarial training. So at a certain moment in FGSM-based adversarial training, the loss *in between* the natural image $\mathbf{x}$ and the surface of the closed ℓ_∞ ball $\overline{\mathcal{B}}_\mathcal{X}(\mathbf{x}, \epsilon^*)$ on which the FGSM-perturbed data is located suddenly begins to increase strongly. With PGD not being restricted to that surface, but being able to find maxima of the loss function in between, any previously gained robustness against it is lost.

To another part, the constant initialization of FGSM at the natural image can lead to similar adversarial examples over epochs, if the direction of the gradient only slightly changes. Random initialization, by contrast, leads inevitably to a higher diversity of adversarial examples. At this point, one might have noticed that the way the random initialization is performed has the same risk of potentially "running out of scope" during loss maximization as for PGD. However, opposite to PGD, this could become critical here, as only a single step is performed. In total, the risk is still considered to be low and due to time reasons, we only resort to the original formulation of the SSS algorithm [53]. Nevertheless, this is an aspect that should be investigated further in some future work.

3.6. EXPERIMENTS

In this section, we present the experiments that are conducted in the scope of this work, based on the previously shown materials and methods. Before doing so, however, we briefly explain the general circumstances and computational environment on which the experiments are based or in which they take place.

First of all, *Python* [55] is used throughout as a programming language. Furthermore, we use the *PyTorch Framework* [56] to build and train DNNs and also implement the previously introduced adversarial attacks as well as adversarial training methods. PyTorch also includes the *Torchvision* package, providing various datasets and algorithms for computer vision. Besides, we resort to the Python package *Numpy* [57] to perform further computations on the generated data. Last, plots are created using the Python package *Matplotlib* [58]. The exact version numbers of these are stated in table 3.1.

Table 3.1.: Used software. The left column contains the name, the middle column the version number and the right column the purpose.

Name	Version number	Purpose
Python	3.7.7	Programming language
PyTorch	1.8.1	Deep learning
Torchvision	0.9.1	Computer vision
Numpy	1.20.3	Linear algebra and n-dimensional arrays
Matplotlib	3.4.2	Data plots

The experiments are conducted on a computer cluster of the German Aerospace Center (DLR). It consists of four CPU-nodes, each equipped with four Intel Xeon Gold 6132 CPUs and 384 GB DDR4 RAM, and one GPU node equipped with two Intel Xeon Silver 4112 CPUs, 192 GB DDR4 RAM, as well as four NVIDIA Tesla V100s connected via NVLink. To accelerate both the training of DNNs and the creation of adversarial examples, we utilize the latter. But, since distributed training across multiple GPUs requires a more complex configuration and could therefore include bugs that are difficultly to find later on, we only resort to a single GPU.

Concerning the experiments themselves, they can be divided into the following five sections:

1. Standard training of DNNs,

2. Adversarial examples and robustness after standard training,

3. Adversarial training of DNNs,
4. Adversarial examples and robustness after adversarial training, and
5. Investigation of the loss landscape.

In the following, the setup for each of them is explained in detail.

3.6.1. *Standard Training of Deep Neural Networks*

We train two DNNs with PreAct-ResNet-50 architecture on the CIFAR-10 and GTSRB datasets for 100 epochs with batch size 128 each. Here, the number of epochs is mostly limited by the available computation time, whereas the chosen batch size is from experience a reasonable trade-off between acceleration through parallelization (higher batch size) and stochasticity (lower batch size).

We use cross entropy as a loss function and AdamW as an optimization algorithm. Concerning the latter, we choose the weight decay coefficient $\alpha^* = 0.01$. This choice is equivalent to the default value in the PyTorch implementation of AdamW; due to the limited available amount of time, it is unfortunately not possible for us to make an extensive optimization of hyperparameters. Hence, other hyperparameters of AdamW are set to their default values as well ($\beta_1 = 0.9$ and $\beta_2 = 0.999$).

Furthermore, cosine annealing is used as a learning rate scheduler, where the initial and minimum learning rate are set to $\eta_{max} = 0.001$ and $\eta_{min} = 0$, being further default values of the PyTorch implementation[8]. Since in the paper of Loshchilov and Hutter [47], the positive effect of warm restarts comes to bear only for cycle lengths T_{max} larger than 100 epochs, but we only train for 100 epochs, we also set $T_{max} = 100$.

We use the separation into training and validation datasets which is already provided by both CIFAR-10 and GTSRB. All images are resized to $32\,\text{px} \times 32\,\text{px}$ if they are not already in this resolution. In addition, we normalize image data by mapping discrete pixel intensities from $[0..255]$ linearly to $[0, 1]$. The training data is also dynamically augmented according to section 3.3.1. Images from the validation dataset, however, are not augmented this way to maintain a constant reference for validation. To avoid repetitive gradient patterns across epochs, the training data is shuffled every epoch.

Last but not least, when an epoch is finished, we evaluate the accuracy as well as the loss as averages over the training dataset and the validation dataset. In the end, we save the model parameters under which the lowest overall validation loss was achieved. Therefore, we perform a kind of early stopping without missing the possibility of a later loss decrease, but minimizing overfitting and ensuring good generalization.

[8]Note that in PyTorch 1.8.1, the initial learning rate $\eta_{max} = 0.001$ is not a parameter of the learning rate scheduling algorithm, but of the optimization algorithm and thus of AdamW as well.

3.6.2. *Adversarial Examples and Robustness after Standard Training*

We use two white-box attacks (FGSM, PGD) and one black-box attack (Boundary Attack) to create adversarial examples on the previously trained DNNs. The loss function is still cross entropy, as in training. For both CIFAR-10 and GTSRB, the respective validation dataset serves as a basis to create the perturbed images.

Moreover, we create uniformly distributed random perturbations

$$\mathbf{u} = \operatorname{sign} \mathbf{u}^* \quad \text{with} \quad u_1^*, u_2^*, \ldots, u_n^* \sim \mathcal{U}(-1, 1), \tag{3.38}$$

scale them by the appropriate magnitude of perturbation ϵ and eventually apply them to natural images $\mathbf{x}$ by addition and subsequent projection back onto the input space $\mathcal{X}$:

$$\mathbf{x}' = \Pi_{\mathcal{X}} \left(\mathbf{x} + \epsilon \mathbf{u} \right). \tag{3.39}$$

Note here that concerning the FGSM, PGD and the random perturbations, we use the ∞-metric as a dissimilarity metric. It has the advantage that it can be interpreted well in the case at hand, operating on image data: For two images $\mathbf{x}^{(1)}, \mathbf{x}^{(2)} \in \mathbf{x}$, e.g., $\|\mathbf{x}^{(1)} - \mathbf{x}^{(2)}\|_\infty = 1/255$ means that the pixel-wise difference between $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$ is at most 1 in any color channel, subject to an 8-bit encoding per channel.

Now, regarding PGD, we perform 20 iterations with step size $\eta' = 2/255$, which is the same as Madry et al. used in their work on CIFAR-10 [21]. We then construct four threat models for each of the three ℓ_∞-bounded attacks, where the maximum permitted dissimilarity is set to $\epsilon = 4/255, 8/255, 12/255, 16/255$, respectively. For every threat model, we perform the corresponding attack on all images of the respective validation dataset and therefore evaluate the robustness of the standardly trained DNN against adversarial examples. We finally save the perturbed images that were created out of the first 100 natural images of the considered validation dataset.

Concerning the Boundary Attack, which is internally based on the Euclidean metric, we perform $N = 160$ iterations on each image of a 2000-image subset of the respective validation dataset. Besides, the number of sampled directions is set to $M = 16$. Because of the high computational cost of the Boundary Attack, it is unfortunately not possible for us to perform either more iterations, consider the entire validation dataset or compute more sample candidates. After every 10 epochs, we save the ℓ_2 distance between the current version of the perturbed image $\mathbf{x}'$ and the natural image $\mathbf{x}$. Furthermore, after every 40 epochs, we save the current version of the perturbed image $\mathbf{x}'$ itself, if the corresponding natural image $\mathbf{x}$ was under the first 100 images of the validation dataset.

Last, we also save classification results as well as corresponding certainties for all created perturbed images.

3.6.3. *Adversarial Training of Deep Neural Networks*

We train further DNNs with PreAct-ResNet-50 architecture on the CIFAR-10 and GTSRB datasets. This time, however, we perform adversarial training, using dynamically generated adversarial examples as the training data. For this purpose, we define four different threat models under which these adversarial examples are created, respectively:

(A) FGSM (white-box), untargeted, $d = d_\infty$, $\epsilon^* = 8/255$;

(B) SSS (white-box), untargeted, $d = d_\infty$, $\epsilon^* = 8/255$;

(C) FGSM (white-box), untargeted, $d = d_\infty$, $\epsilon^* = 16/255$; and

(D) SSS (white-box), untargeted, $d = d_\infty$, $\epsilon^* = 16/255$.

Thus, together with the two datasets, we consider eight different defense scenarios.

Concerning the training settings and hyperparameters, each scenario is set up in the same way. Actually, we use the same settings here as for the standard training (see section 3.6.1), with differences in the measurements on the validation dataset and the condition for eventual saving of model parameters:

First, beyond the computations of loss and accuracy on the (now perturbed) training dataset and the natural validation dataset, we also compute them for the FGSM- and PGD-perturbed data from the validation dataset, called *FGSM loss/accuracy* and *PGD loss/accuracy*, respectively. Here, PGD settings are the same as in section 3.6.2. Due to the additional computational cost of adversarial examples' generation just for validation purposes, especially when using PGD, we only take a random subset of the validation dataset here, which is one-tenth the size of the entire validation dataset. After each epoch, it is replaced by another randomly sampled subset.

Second, we save the model parameters under which the lowest overall *mixed loss* was obtained. We define the mixed loss as the average of natural validation loss and FGSM loss at a specific epoch. Doing so, we take both the accuracy w.r.t. natural images and the robustness against FGSM-perturbed images into account, since it is not guaranteed that the latter implies the former.

At this point, modeling the mixed loss as the average of the natural validation loss and the PGD loss instead would admittedly be the better approach to avoid potential catastrophic overfitting for the saved model. However, we defined the mixed loss on the basis of the FGSM loss on purpose, being able to ultimately investigate trained DNNs' suffering from overfitting against FGSM perturbations.

Last, regarding SSS-based adversarial training, we use levels of discretization $L = 2$ and $L = 4$ for $\epsilon^* = 8/255$ and $\epsilon^* = 16/255$, respectively. Doing so, we maintain the same width of discretization bins $4/255$.

3.6.4. *Adversarial Examples and Robustness after Adversarial Training*

Here, we essentially repeat the experiments in section 3.6.2 for the DNNs that were trained adversarially on the datasets CIFAR-10 and GTSRB, according to section 3.6.3.

3.6.5. *Investigation of the Loss Landscape*

Finally, we also make some qualitative investigation of the so-called *loss landscape* in the input space, i.e., the surface that originates from evaluation of the loss function $J(\hat{f}(\mathbf{x}), \mathbf{y})$ for a DNN classifier $\hat{f}$ and all input images $\mathbf{x} \in \widetilde{\mathcal{X}}$ as well as corresponding ground truth $\mathbf{y} \in \mathcal{Y}$. Here, $\widetilde{\mathcal{X}} \subseteq \mathcal{X}$ is a subset of the full input space. Moreover, we use cross entropy as the loss function, just as for DNN training and the creation of adversarial examples.

Qualitative investigation means that we create plots of the loss landscape and evaluate them visually. Since we can only plot in three dimensions and one dimension is already reserved by the loss itself, we define $\widetilde{\mathcal{X}}$ as the two-dimensional plane

$$\widetilde{\mathcal{X}} = \left\{ \mathbf{x}' \mid \mathbf{x}' = \Pi_{\mathcal{X}} \left(\mathbf{x} + \lambda \mathbf{u} + \mu \mathbf{v} \right) \right\}. \tag{3.40}$$

In this context, $\mathbf{x} \in \mathcal{X}$ denotes a specific natural image from the validation dataset with regard to which we want to evaluate the loss landscape.

Next, $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ represent roughly a random and the adversarial direction, starting from $\mathbf{x}$ and spanning the plane. To align with the ∞-metric-based threat model used for adversarial attacks, both $\mathbf{u}$ and $\mathbf{v}$ are results of applying the sign function to the original random and adversarial direction vectors. That is, they are defined as

$$\mathbf{u} = \operatorname{sign} \mathbf{u}^* \quad \text{with} \quad u_1^*, u_2^*, \ldots, u_n^* \sim \mathcal{U}(-1, 1) \tag{3.41}$$

and

$$\mathbf{v} = \operatorname{sign} \nabla_{\mathbf{x}} J(\hat{f}(\mathbf{x}), \mathbf{y}). \tag{3.42}$$

As we operate in high-dimensional spaces and the vectors $\mathbf{u}$ and $\mathbf{v}$ are uncorrelated, we can assume them being approximately orthogonal, which is required to use them as axes of the plots later on.

This can be shown quickly in the spacial case at hand: First, components of both vectors $\mathbf{u}$ and $\mathbf{v}$ are in the most cases and with equal probability either equal to 1 or -1 and with some significantly lower probability equal to 0. In consequence, the probability that a pair of components (u_i, v_i) with $i \in \{1, 2, \ldots, n\}$ equals either $(1, 1)$, $(-1, -1)$, $(1, -1)$ or $(-1, 1)$ is about the same.

Hence, applying the law of large numbers w.r.t. n, the scalar product $\mathbf{u}^\intercal \mathbf{v}$ tends to

zero; 1 and -1 as possible dimension-wise products of two components u_i, v_i have equal probability, thus canceling out each other, and if one or both components are equal to 0, the product does not contribute to $\mathbf{u}^\top \mathbf{v}$ at all:

$$\lim_{n\to\infty} \mathbf{u}^\top \mathbf{v} = 0. \tag{3.43}$$

At the same time, with some large n, we cannot neglect the possibility of $u_i = 0$ or $v_i = 0$ completely anymore, while it is still comparatively rare. As a result, the product of the Euclidean norms of both $\mathbf{u}$ and $\mathbf{v}$ is lower than the theoretical maximum n^2, but clearly larger than zero:

$$0 \ll \lim_{n\to\infty} \|\mathbf{u}\|_2 \|\mathbf{v}\|_2 < n^2. \tag{3.44}$$

Now, with the identity

$$\frac{\mathbf{u}^\top \mathbf{v}}{\|\mathbf{u}\|_2 \|\mathbf{v}\|_2} = \cos \sphericalangle(\mathbf{u}, \mathbf{v}) \tag{3.45}$$

as well as eqs. (3.43) and (3.44), we see that the angle $\sphericalangle(\mathbf{u}, \mathbf{v})$ tends to 90°, so $\mathbf{u}$ and $\mathbf{v}$ can be considered being roughly orthogonal.

Returning to the definition of the two-dimensional plane, we aim to investigate the loss landscape specifically in the area of adversarial perturbations that are possible according to the used threat model. Consequently, the two plane parameters $\lambda, \mu \in \left[0, \frac{16}{255}\right]$ are both constrained to vary between 0 (no perturbation) and 16/255 (maximum considered magnitude of perturbation). Besides, note the projection back onto the input space, which is necessary as the plane potentially exceeds the input space otherwise.

Last, we cannot model the loss landscape with arbitrary precision because there is no simple, closed formula that describes it, but the loss function must be evaluated at specific points. Hence, we discretize $\widetilde{\mathcal{X}}$ by redefining the plane parameters λ and μ as $\lambda, \mu \in \left\{\frac{i}{40}\frac{16}{255} \mid i \in \{0, 1, \ldots, 40\}\right\}$, resulting in 1681 points in $\widetilde{\mathcal{X}}$ at which the loss function has to be evaluated.

To do so, we do the following for all previously trained DNNs $\hat{f}$, whether standardly or adversarially, and a selection of the images $\mathbf{x}$ as well as corresponding ground truth $\mathbf{y}$ from the validation datasets of CIFAR-10 and GTSRB: At first, we perform a forward and backward pass with the original input image $\mathbf{x}$ to compute the gradient $\nabla_{\mathbf{x}} J(\hat{f}(\mathbf{x}), \mathbf{y})$. We also sample a random vector from the uniform distribution, compute the direction vectors $\mathbf{u}$ and $\mathbf{v}$ and eventually compute the set of points $\widetilde{\mathcal{X}}$. Subsequently, we consider $\widetilde{\mathcal{X}}$ as a batch of input images with all the same ground truth $\mathbf{y}$ and propagate them through the DNN to create classification results $\hat{\mathbf{y}}$. In the end, we evaluate the loss function J for all retained classification results $\hat{\mathbf{y}}$ and the constant ground truth $\mathbf{y}$.

4 RESULTS

In this chapter, we show the results of the experiments specified in section 3.6. Basically, the structure here follows that of section 3.6. In addition to that, the results of each experiment are divided according to the two considered datasets CIFAR-10 and GTSRB.

4.1. STANDARD TRAINING OF DEEP NEURAL NETWORKS

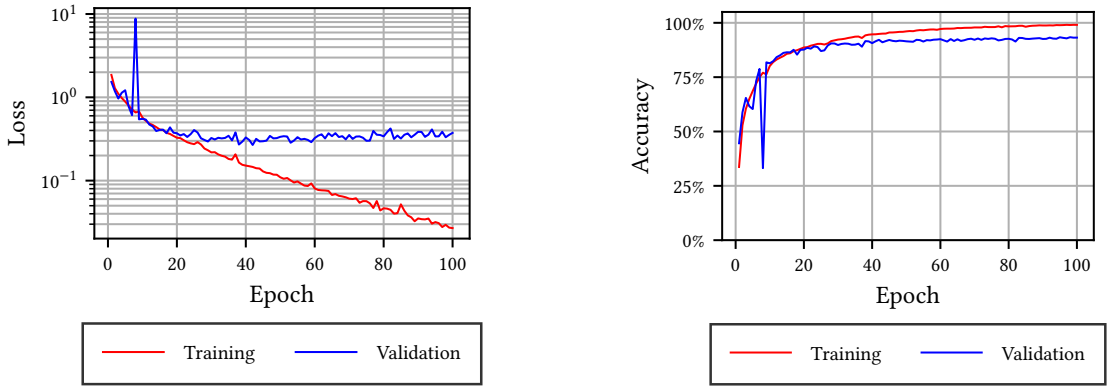
Figures 4.1 and 4.2 show the development of both the loss and the accuracy over the training process for CIFAR-10 and GTSRB, respectively.

4.1.1. *CIFAR-10*

Regarding the training on the CIFAR-10 dataset, the minimum validation loss of 0.268 was achieved at epoch 42. Moreover, after epoch 42, validation accuracy was at 92.19 %. From this point, both validation loss and accuracy increased slightly, but nearly steadily until the end of training. In general, the development of the validation loss and accuracy was quite smooth, with a single outlier at epoch 8, where the loss increased sharply to 8.81 and the accuracy decreased sharply to 33.14 %.

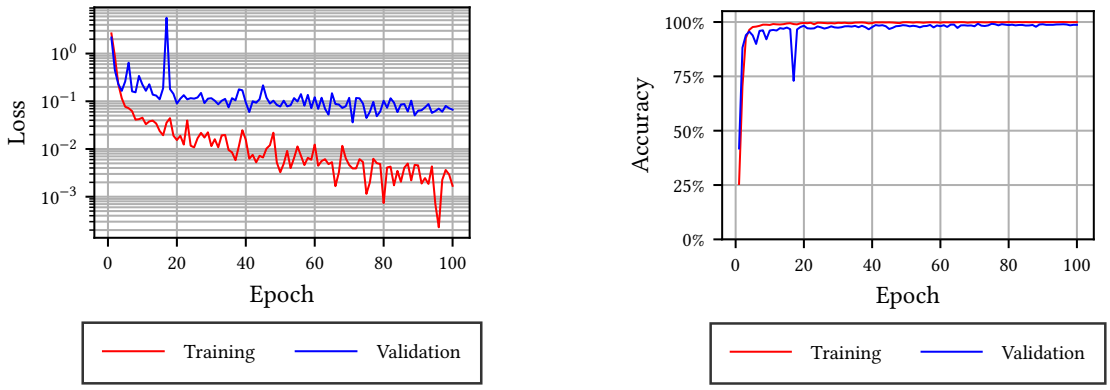
4.1.2. *GTSRB*

For the GTSRB dataset, the minimum validation loss of 0.036 was achieved at epoch 71. At this point, validation accuracy was at 99.22 %, which was the highest overall value as well. In contrast to the training on CIFAR-10, the validation loss did not increase recognizably after that minimum, but fluctuates minimally over it. Analogous applies to the validation accuracy. Last, similar to the training on CIFAR-10, there was a single outlier at epoch 17, where the loss increased sharply to 5.616 and the accuracy decreased sharply to 72.93 %.



(a) Loss **(b)** Accuracy

Figure 4.1.: Loss and accuracy development of PreAct-ResNet-50, trained standardly on CIFAR-10. For both the loss (left) and the accuracy plot (right), evaluation on the training dataset is shown in red and evaluation on the validation dataset is shown in blue.



(a) Loss **(b)** Accuracy

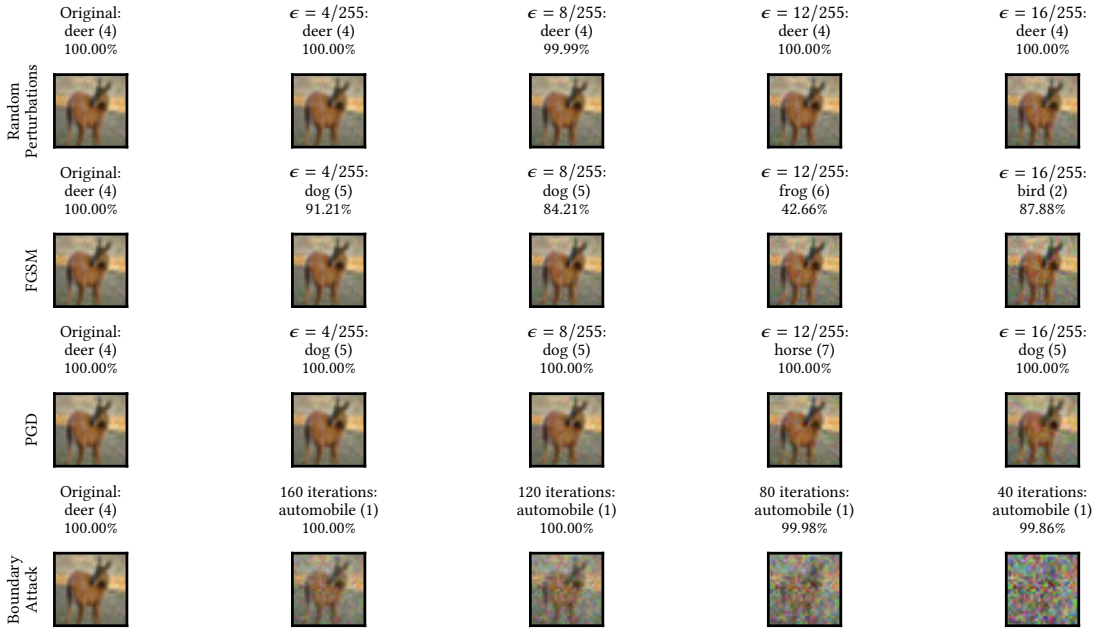
Figure 4.2.: Loss and accuracy development of PreAct-ResNet-50, trained standardly on GTSRB. For both the loss (left) and the accuracy plot (right), evaluation on the training dataset is shown in red and evaluation on the validation dataset is shown in blue.

4.2. ADVERSARIAL EXAMPLES AND ROBUSTNESS AFTER STANDARD TRAINING

4.2.1. *CIFAR-10*

Adversarial Examples Figure 4.3 shows perturbed versions of two images from the validation dataset of CIFAR-10, depicting a deer and a horse, respectively. First, the random perturbations led to no misclassification, independent of the applied magnitude of perturbation. Besides, the classification certainty was constantly at a high level.

With FGSM, the created perturbed images are invariably adversarial examples. But for the image of a deer, the class and the certainty varied for different values of ϵ, e.g., with



(a) Deer

(b) Horse

Figure 4.3.: Adversarial examples of CIFAR-10, created on standardly trained PreAct-ResNet-50 (Deer, Horse). Each row shows perturbed versions of a natural image, created with one of the adversarial attacks: random perturbations, FGSM, PGD and Boundary Attack. The original image is shown in the left column. Remaining columns show perturbed versions with maximum ℓ_∞ distance $\epsilon = 4/255, 8/255, 12/255, 16/255$ (from left to right, for random perturbations, FGSM and PGD) and after 40, 80, 120, 160 iterations of Boundary Attack (from right to left), respectively. Classification information is provided above the corresponding image.

only 42.66 % of recognizing a frog at $\epsilon = 12/255$. FGSM-perturbed versions of the second image, on the contrary, were misclassified as a horse without exception. Certainty here was close to or at 100 %.

The PGD attack caused misclassification along all scenarios as well. Comparing to FGSM, however, both the class and the certainty were more constant, with the latter being constantly maximized.

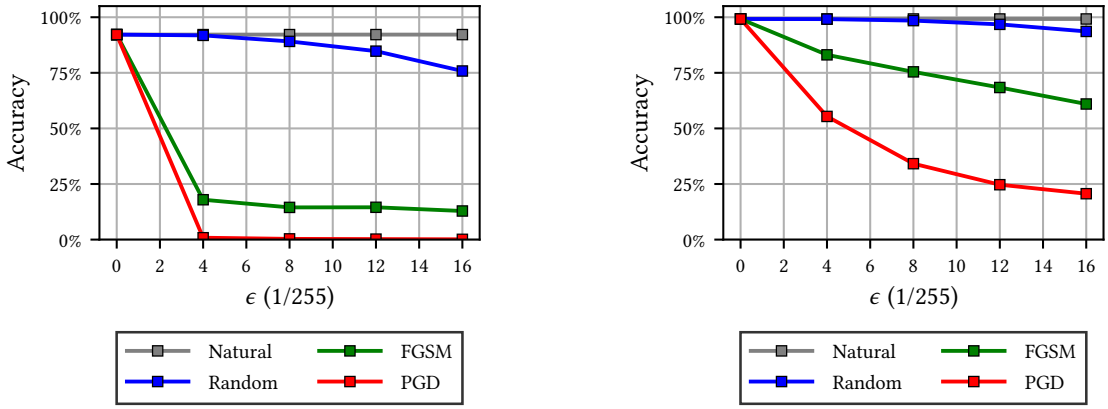
Last, due to its mechanics, the Boundary Attack always led to misclassification as well. Certainty here was also close to 100 %, independent of the progress of the attack. In addition, the created adversarial examples were constantly classified as an automobile.

Concerning the three ϵ-based attacks, differences to the original image are only barely recognizable at $\epsilon = 4/255$. From $\epsilon = 8/255$ on, however, the noise comes more and more to the fore. In particular, the maximally PGD-perturbed images exhibit interesting changes. For the first image, the shadow on the deer's head was slightly lightened and the shadow beneath was simultaneously darkened. In consequence, the impression of a dog looking directly at the viewer is originated. Not surprisingly, this image is also classified as a dog. Something similar happened to the second image of a horse, which was given antlers and got classified as a deer. Moreover, noise on the green appears to be structured and lengthen its legs, with the former head being one another leg.

For the adversarial examples created by the Boundary Attack, only those after 160 iterations can even close compete with the white box attacks in terms of imperceptibility. Nevertheless, they contain still more visible noise than the latter at $\epsilon = 16/255$. The images that were created after fewer iterations are clearly dominated by noise.

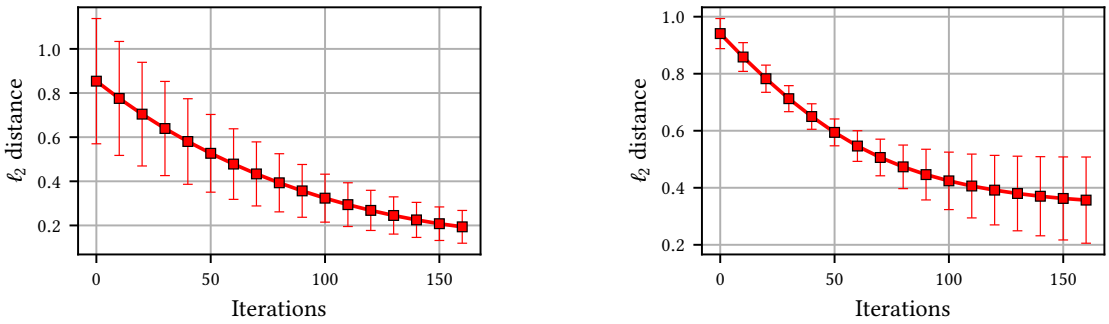
Adversarial Robustness Figure 4.4a shows the adversarial robustness of PreAct-ResNet-50, trained standardly on CIFAR-10. The plot largely reflects the above observations. PGD caused almost guaranteed misclassification already at the smallest tested perturbation magnitude $\epsilon = 4/255$. FGSM also led to substantial decline of accuracy to 17.93 % at $\epsilon = 4/255$. For higher values of ϵ, the accuracy stayed approximately at the same level, with some slight fluctuations. In comparison, random perturbations produced only a slight decline of accuracy. Nevertheless, for higher magnitudes of perturbation, the decrease of accuracy became stronger up to a minimum of 75.82 % at $\epsilon = 16/255$.

Boundary Attack Last, fig. 4.5a shows the development of the average ℓ_2 distance between the original images and the current perturbed versions during the Boundary Attack. One can see some kind of exponential decrease for increasing iteration count, starting at a distance of roughly 0.85 and ending at about 0.2. Furthermore, the standard deviation of the ℓ_2 distance seems to be correlated to the average distance itself, exhibiting the same exponential decrease.



(a) CIFAR-10 **(b)** GTSRB

Figure 4.4.: Adversarial robustness of PreAct-ResNet-50, trained standardly on CIFAR-10 (left) and GTSRB (right). For both datasets, the accuracy under the adversarial attacks of random perturbations (blue), FGSM (green) and PGD (red) was evaluated. Attacks were performed using the ∞-metric and varied in terms of the maximum permitted dissimilarity ϵ, being plotted on the x-axis. As a baseline for natural images, the validation accuracy of the trained model is shown in gray.



(a) CIFAR-10 **(b)** GTSRB

Figure 4.5.: Development of ℓ_2 distance during Boundary Attack on PreAct-ResNet-50, trained standardly on CIFAR-10 (left) and GTSRB (right). For both datasets, the average ℓ_2 distance between the target image and the current perturbed version was evaluated every 10th iteration. 160 iterations were performed in total. Standard deviations are plotted as error bars.

4.2.2. *GTSRB*

Adversarial Examples Figure 4.6 shows perturbed versions of two images from the validation dataset of GTSRB. The first one depicts a sign that that prescribes to drive straight ahead, on the second one there is a sign warning of pedestrians.

First, results for random perturbations are essentially the same as on CIFAR-10, proving the method's non-effectiveness.

Next, by contrast to CIFAR-10, the FGSM attack was also not effective at all on the GTSRB dataset, since the classification was always correct. Moreover, certainty remained over 80 % for the first image and was even constantly at 100 % for the second one.

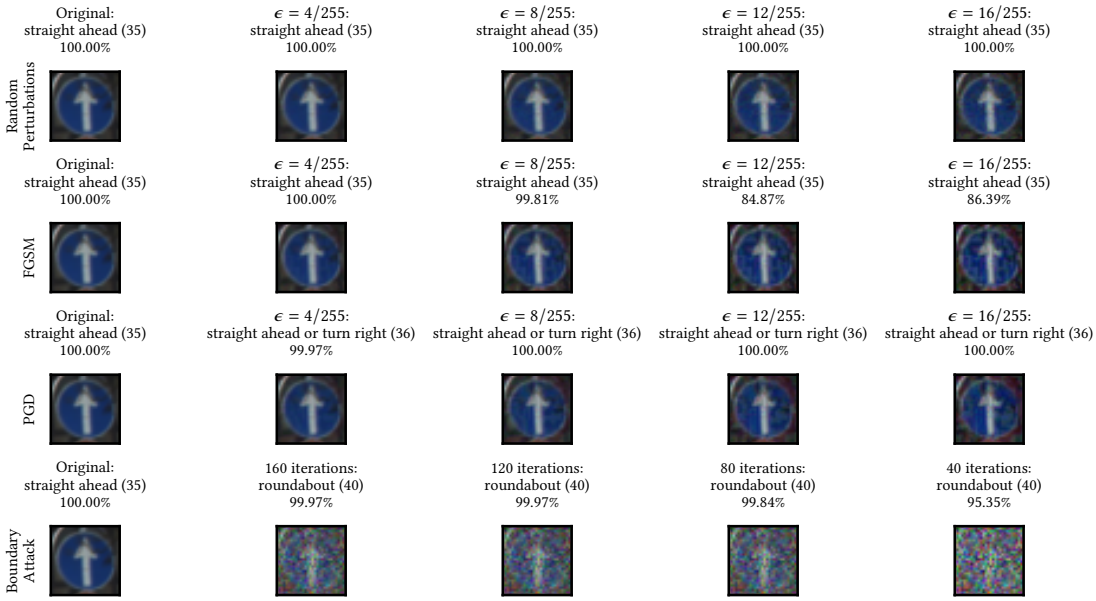
With PGD, in turn, the created perturbed images are again invariably adversarial examples. The predicted class remained constant per original image, independent of ϵ, and the certainty slightly fluctuated between 98 % and 100 %.

Last, adversarial examples created by the Boundary Attack were always classified as the same class (roundabout sign), similar to CIFAR-10. Certainty here was close to 100 % across different states of progress as well, but on average lower than for CIFAR-10.

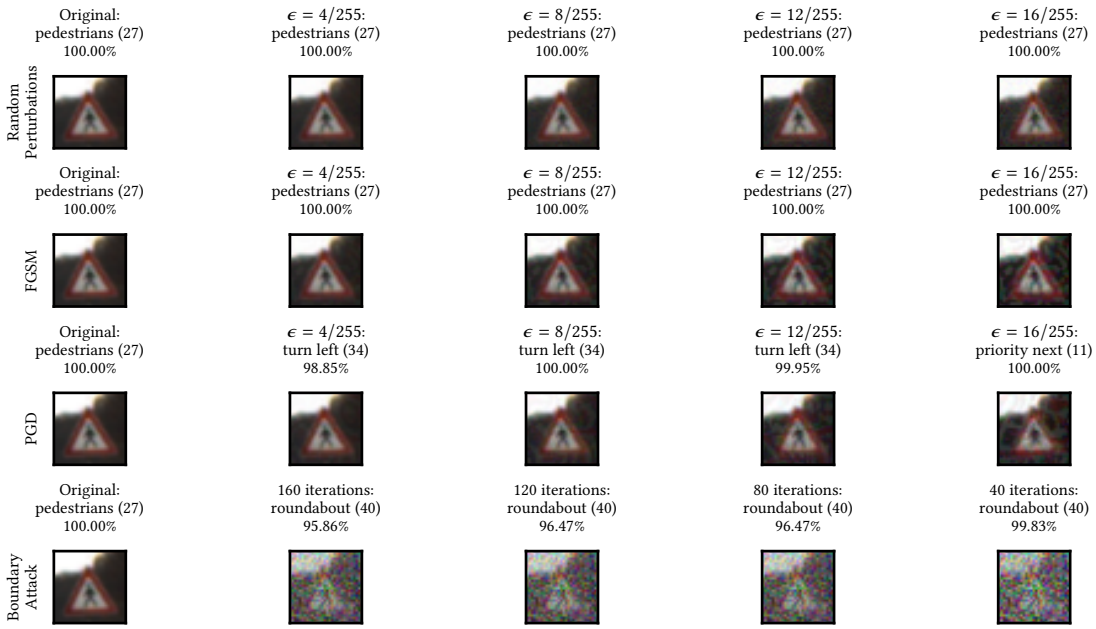
Regarding random perturbations, FGSM and PGD, the qualitative evaluation of perturbations' perceptibility gives roughly the same result as on CIFAR-10; under close inspection, they are imperceptible only at $\epsilon = 4/255$. Again, the maximally PGD-perturbed versions of the images show remarkable resemblance to the visual representation of their wrongly predicted classes. In detail, the former "straight ahead" sign now contains the bright shade of an additional arrow to the right, and limbs of the pedestrian are clearly shortened as well as blurred.

The Boundary Attack was apparently less effective on GTSRB than on CIFAR-10, as the high level of noise even at the maximum count of 160 iterations indicates. Actually, perturbations equal each other at different states of progress, meaning that the attack was not able to make any progress for longest stretches.

Adversarial Robustness Figure 4.4b shows the adversarial robustness of PreAct-ResNet-50, trained standardly on GTSRB. In comparison to CIFAR-10, a higher accuracy was achieved under every considered threat model. At $\epsilon = 4/255$, PGD leads still to 55.39 % accuracy. For larger perturbation magnitudes, however, the loss of accuracy became larger, ending at 20.64 % for $\epsilon = 16/255$. Under FGSM attack, the accuracy declined nearly linearly with a minimum of 60.98 % at $\epsilon = 16/255$. Finally, random perturbations had almost no negative consequences up to $\epsilon = 8/255$. From this point, the accuracy slightly decreased to 96.79 % at $\epsilon = 12/255$ and 93.62 % at $\epsilon = 16/255$.



(a) "Straight ahead" sign



(b) "Pedestrians" sign

Figure 4.6.: Adversarial examples of GTSRB, created on standardly trained PreAct-ResNet-50 (Straight ahead, Pedestrians). Each row shows perturbed versions of a natural image, created with one of the adversarial attacks: random perturbations, FGSM, PGD and Boundary Attack. The original image is shown in the left column. Remaining columns show perturbed versions with maximum ℓ_∞ distance $\epsilon = 4/255, 8/255, 12/255, 16/255$ (from left to right, for random perturbations, FGSM and PGD) and after 40, 80, 120, 160 iterations of Boundary Attack (from right to left), respectively. Classification information is provided above the corresponding image.

Boundary Attack Eventually, fig. 4.5b shows the development of the average ℓ_2 distance between the original images and the current perturbed versions when manipulating the GTSRB dataset with the Boundary Attack. As on CIFAR-10, the average ℓ_2 distance seems to decrease approximately exponentially for increasing iteration count, at least in the first half. From 80 iterations on, it steadily decreases less than former exponential course would predict. Besides, the behavior of the standard deviation of the ℓ_2 distance appears to be roughly inverse to the same on CIFAR-10 at first. In detail, however, instead of slowly increasing, it keeps nearly constant in the aforementioned first half.

4.3. ADVERSARIAL TRAINING OF DEEP NEURAL NETWORKS

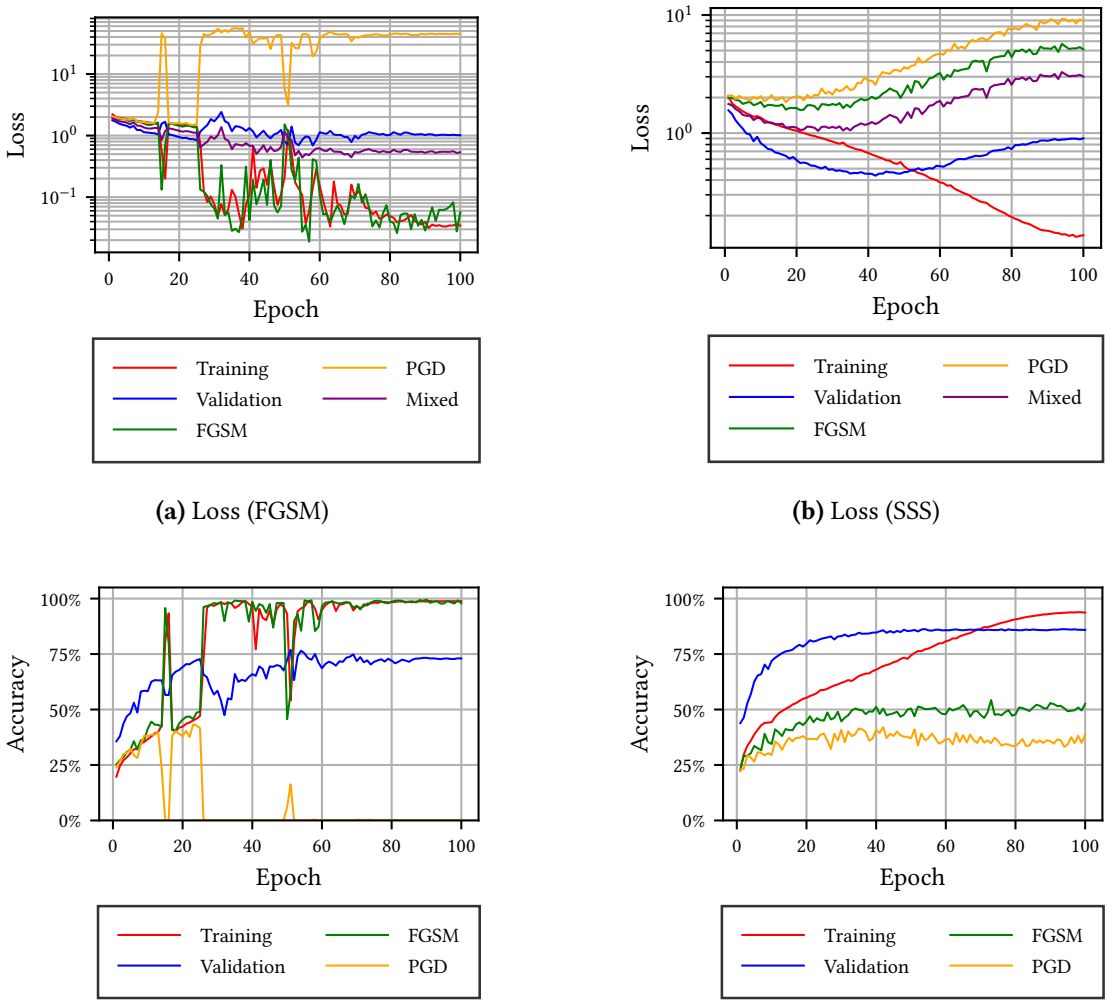
4.3.1. *CIFAR-10*

Figures 4.7 and 4.8 show the development of both the loss and the accuracy over the adversarial training process for CIFAR-10 at $\epsilon^* = 8/255$ and $\epsilon^* = 16/255$, respectively.

Beginning with the results of the threat models (A) and (B), the lowest mixed loss was achieved at epoch 55 for FGSM-based adversarial training and at epoch 26 for SSS-based. These correspond with validation accuracies of 75.28 % and 82.5 %, FGSM accuracies of 99.22 % and 48.67 %, as well as PGD accuracies of 0 % and 35.94 %, respectively. Thus, SSS-based adversarial training outperforms FGSM-based here in terms of the natural validation accuracy and the PGD accuracy, but is being outperformed regarding the FGSM accuracy.

After epoch 55, the results of all metrics remained on average at the same level for FGSM-based adversarial training, although fluctuating in some cases. For SSS-based adversarial training, on the other hand, training loss and accuracy were still decreasing and increasing steadily after epoch 26, respectively, with a slow-down at the very end. The FGSM loss and the PGD loss behaved similarly, but increasing. Besides, the validation loss continued decreasing until epoch 42 and followed the FGSM loss and the PGD loss after that. On the contrary, their counterparts on the accuracy metric remain nearly on the same level, with the validation accuracy and the FGSM accuracy being slightly rising and the PGD accuracy slightly falling.

In general, the loss and accuracy development is way more chaotic for FGSM-based adversarial training than for SSS-based, exhibiting several large fluctuations of the training loss/accuracy, the FGSM loss/accuracy and the PGD loss/accuracy. Furthermore, there are clearly visible correlations between the results of these metrics for FGSM-based adversarial training: First, the training loss/accuracy follows the FGSM loss/accuracy quite well. Second, the FGSM/training loss/accuracy and the PGD loss/accuracy behave inversely to each other, i.e., the former rises when the latter falls and vice versa.



(a) Loss (FGSM)

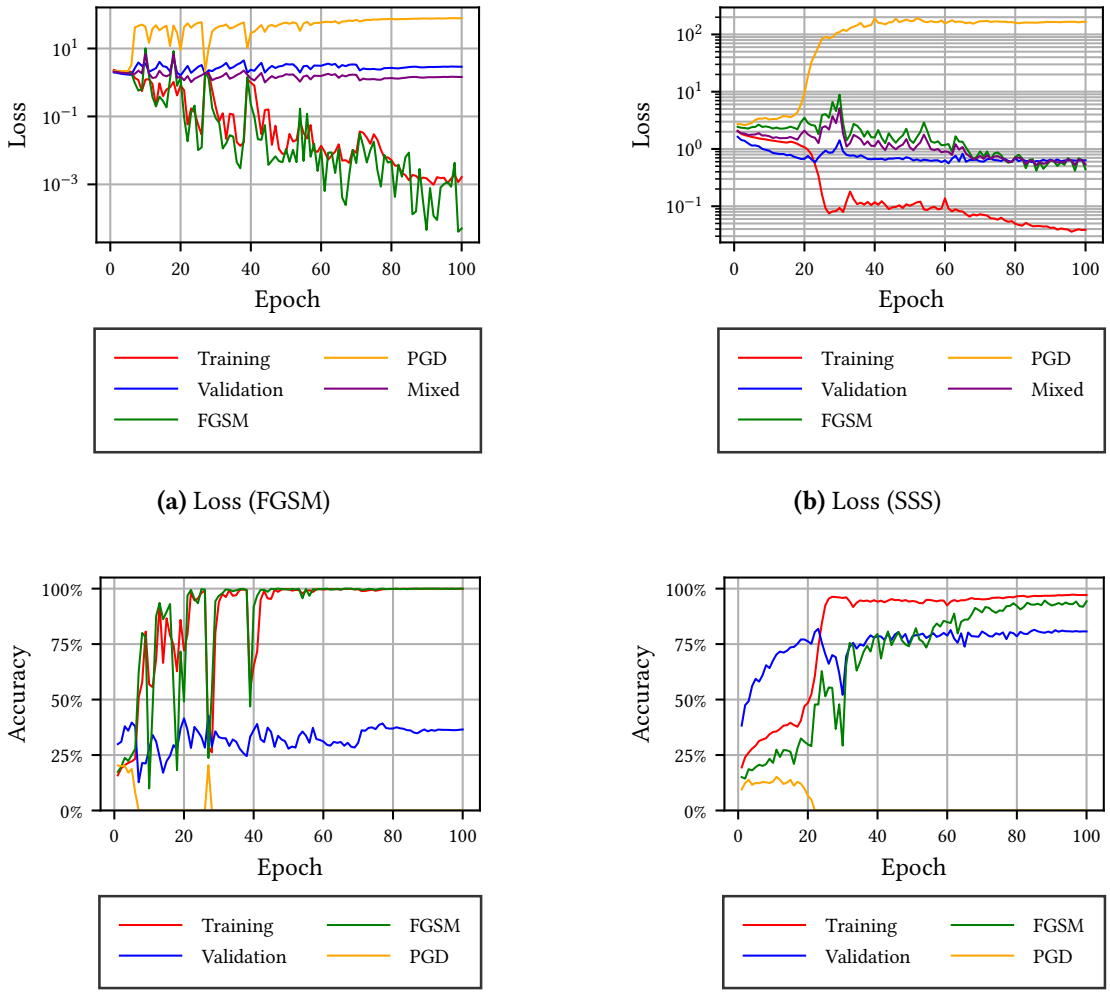
(b) Loss (SSS)

(c) Accuracy (FGSM)

(d) Accuracy (SSS)

Figure 4.7.: Loss (top) and accuracy (bottom) development of PreAct-ResNet-50, trained adversarially on CIFAR-10 using FGSM (left) and SSS (right) at maximum ℓ_∞ distance $\epsilon^* = 8/255$. The training loss/accuracy is colored red, the validation loss/accuracy blue, the FGSM loss/accuracy green, the PGD loss/accuracy yellow, and the mixed loss purple.

Next, considering the threat models (C) and (D), epoch 44 marks the moment of the lowest mixed loss for FGSM-based adversarial training and epoch 86 does for SSS-based. With the model parameters at that time, the following results were obtained on the accuracy metrics: 37.35 % and 80.76 % for the natural validation accuracy, 98.55 % and 93.19 % for the FGSM accuracy, as well as 0 % in both cases for the PGD accuracy, where the first value refers to FGSM-based adversarial training and the second one to SSS-based, respectively. The summary turns out similar to threat models (A) and (B), with FGSM-based adversarial training being superior in view of the FGSM accuracy and SSS-based being superior in view of the natural validation accuracy. But with respect to the PGD accuracy, the latter also resulted in 0 % this time.



(a) Loss (FGSM)

(b) Loss (SSS)

(c) Accuracy (FGSM)

(d) Accuracy (SSS)

Figure 4.8.: Loss (top) and accuracy (bottom) development of PreAct-ResNet-50, trained adversarially on CIFAR-10 using FGSM (left) and SSS (right) at maximum ℓ_∞ distance $\epsilon^* = 16/255$. The training loss/accuracy is colored red, the validation loss/accuracy blue, the FGSM loss/accuracy green, the PGD loss/accuracy yellow, and the mixed loss purple.

In contrast to the corresponding case with $\epsilon^* = 8/255$, the training loss and the FGSM loss decreased further after the epoch of minimum mixed loss for the FGSM-based adversarial training. Besides, the PGD loss increased slightly, while the natural validation loss remained quite stable. The related accuracy metrics developed in a constant way as well, whereas one has to note that both the training accuracy and the FGSM accuracy already almost reached the 100 % maximum at the mentioned epoch 44. Concerning the results of SSS-based adversarial training, all metrics remained approximately at the same level after epoch 86. This comparison is, however, less meaningful, since the number of remaining epochs is highly different here.

Last, the chaotic behavior of the FGSM-based adversarial training is evident once again.

SSS-based adversarial training also proved to be less stable in terms of fluctuations this time, while the extent of fluctuations was still higher for the FGSM-based. Moreover, the correlations identified between different metrics for FGSM-based adversarial training occurred again. On the one hand, the mirroring of FGSM/training loss/accuracy and PGD loss/accuracy is less consistent than with $\epsilon^* = 8/255$, on the other hand, it is now slightly detectable for the SSS-based adversarial training as well, when comparing training loss/accuracy and PGD loss/accuracy.

4.3.2. *GTSRB*

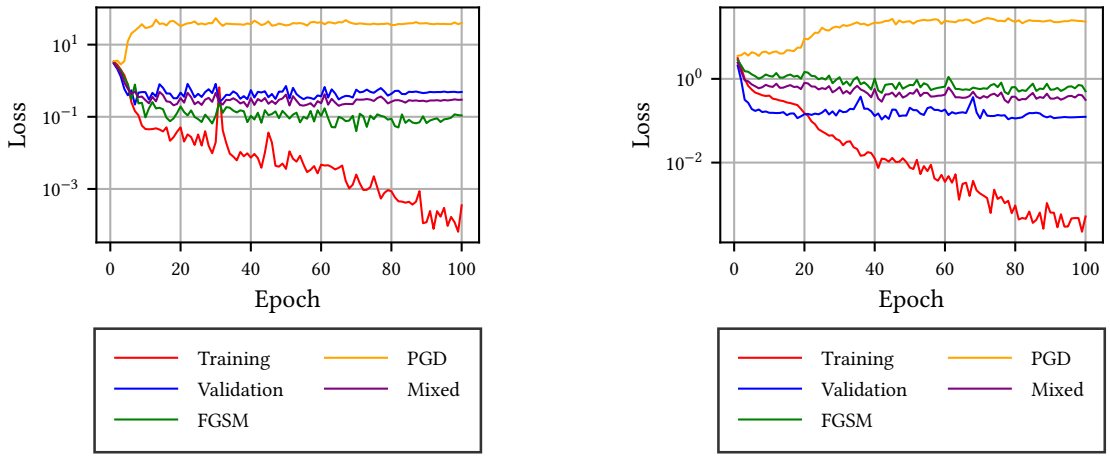
Figures 4.9 and 4.10 show the development of both the loss and the accuracy over the adversarial training process for GTSRB at $\epsilon^* = 8/255$ and $\epsilon^* = 16/255$, respectively.

First of all, considering the threat models (A) and (B), the lowest mixed loss was achieved at epoch 39 for FGSM-based adversarial training and at epoch 78 for SSS-based. The corresponding accuracies are 92.38 % and 97.67 % for the validation accuracy, 97.92 % and 93.66 % for the FGSM accuracy, as well as 28.09 % and 44.53 % for the PGD accuracy. Again, the first value refers to FGSM-based adversarial training and the second one to SSS-based, respectively. Hence, the behavior is equal to the threat models (A) and (B) on CIFAR-10, with SSS-based adversarial training outperforming FGSM-based in terms of the natural validation accuracy and the PGD accuracy, but being outperformed regarding the FGSM accuracy.

Comparing the basic development of loss and accuracy metrics for the FGSM-based and the SSS-based adversarial training, it is altogether quite similar, even though concrete values differ. In both cases, the majority of loss metrics remain nearly on a constant level, only the training loss falls steadily from macroscopical view, disregarding minor fluctuations. Concerning the accuracy metrics, training and natural validation accuracy quickly raised to their maximum in general, although it took some epochs more for SSS-based adversarial training than for FGSM-based.

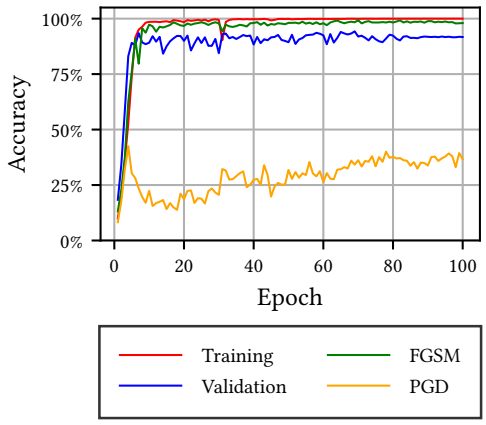
Major difference is, as being visible on CIFAR-10, that the FGSM loss/accuracy is constantly lower/higher than the natural validation loss/accuracy for FGSM-based adversarial training, but the opposite holds for SSS-based adversarial training. In addition, for FGSM-based adversarial training, the FGSM accuracy is almost immediately at its maximum, just like the training accuracy. By contrast, for SSS-based adversarial training, it increased steadily, but slowly until the end of training.

Finally, the previously identified correlation between training loss and FGSM loss is not detectable on GTSRB. This does, however, not hold for their accuracy counterparts as well as the second kind of correlations: For the FGSM-based adversarial training, the known mirroring can be observed right at the start of training, where training loss/accuracy and

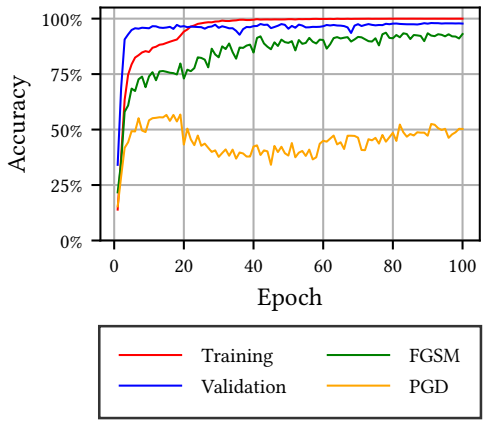


(a) Loss (FGSM)

(b) Loss (SSS)



(c) Accuracy (FGSM)



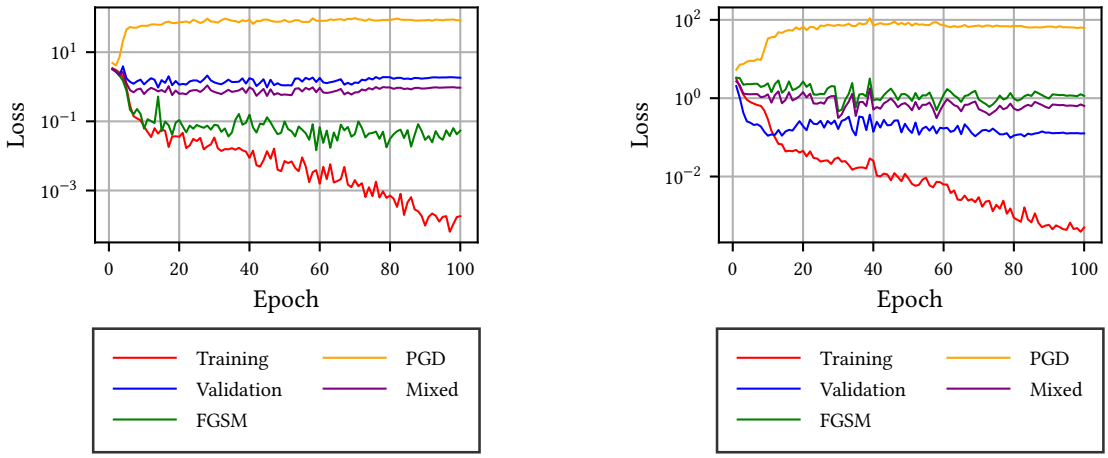
(d) Accuracy (SSS)

Figure 4.9.: Loss (top) and accuracy (bottom) development of PreAct-ResNet-50, trained adversarially on GTSRB using FGSM (left) and SSS (right) at maximum ℓ_∞ distance $\epsilon^* = 8/255$. The training loss/accuracy is colored red, the validation loss/accuracy blue, the FGSM loss/accuracy green, the PGD loss/accuracy yellow, and the mixed loss purple.

PGD loss/accuracy diverge. Analogous applies for the SSS-based adversarial training at epoch 20.

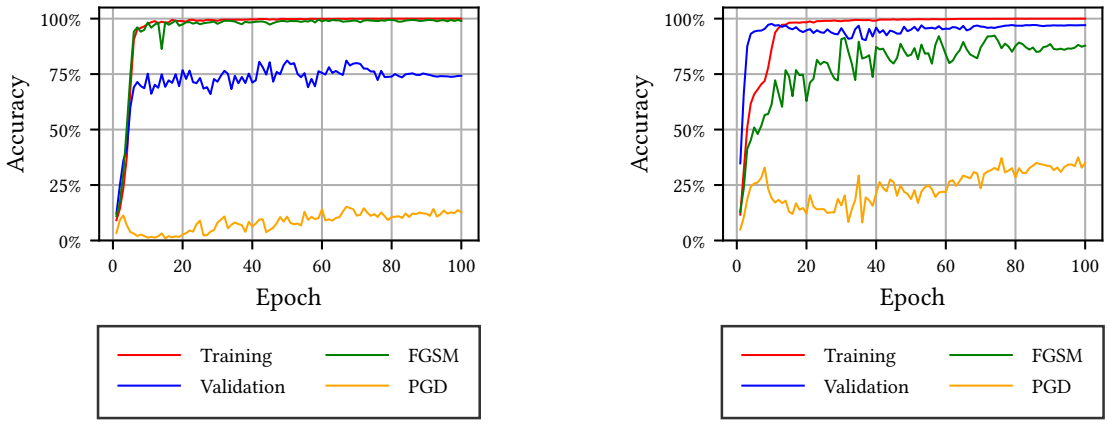
Moving on to the threat models (C) and (D), the minimum mixed loss was here measured at epoch 45 for FGSM-based adversarial training and epoch 58 for SSS-based. Simultaneously, validation accuracies of 80.35 % and 95.49 %, FGSM accuracies of 97.31 % and 88.19 %, as well as PGD accuracies of 4.77 % and 19.62 % were achieved, respectively. As for the $\epsilon^* = 8/255$ scenarios, the FGSM-based adversarial training outperformed the SSS-based in terms of the FGSM accuracy, while being just the opposite in view of the natural validation accuracy and the PGD based accuracy.

Comparing the results between the equivalent $\epsilon^* = 8/255$ and $\epsilon^* = 16/255$ cases, there



(a) Loss (FGSM)

(b) Loss (SSS)



(c) Accuracy (FGSM)

(d) Accuracy (SSS)

Figure 4.10.: Loss (top) and accuracy (bottom) development of PreAct-ResNet-50, trained adversarially on GTSRB using FGSM (left) and SSS (right) at maximum ℓ_∞ distance $\epsilon^* = 16/255$. The training loss/accuracy is colored red, the validation loss/accuracy blue, the FGSM loss/accuracy green, the PGD loss/accuracy yellow, and the mixed loss purple.

are many similarities and only a few differences. In the following, we point out the latter and refer here to the results measured with $\epsilon^* = 16/255$, comparing them with those observed at $\epsilon^* = 8/255$: First, the natural validation loss/accuracy was constantly higher/lower for the FGSM-based adversarial training. Second, the PGD loss/accuracy was constantly higher/lower for both the FGSM-based and the SSS-based adversarial training. Last, fluctuations were overall stronger for the SSS-based adversarial training.

4.4. ADVERSARIAL EXAMPLES AND ROBUSTNESS AFTER ADVERSARIAL TRAINING

Due to repetitiveness, we only present adversarial examples here that were created on the DNNs being trained with SSS and the maximum permitted ℓ_∞ distance $\epsilon^* = 8/255$ (threat model (B)). Adversarial examples that originate from models trained adversarially on the basis of the remaining threat models (A), (C) and (D) can be found in appendix C.

4.4.1. *CIFAR-10*

Adversarial Examples Figure 4.11 shows perturbed versions of the same two CIFAR-10 images as in fig. 4.3. This time, however, they are created on an adversarially trained PreAct-ResNet-50.

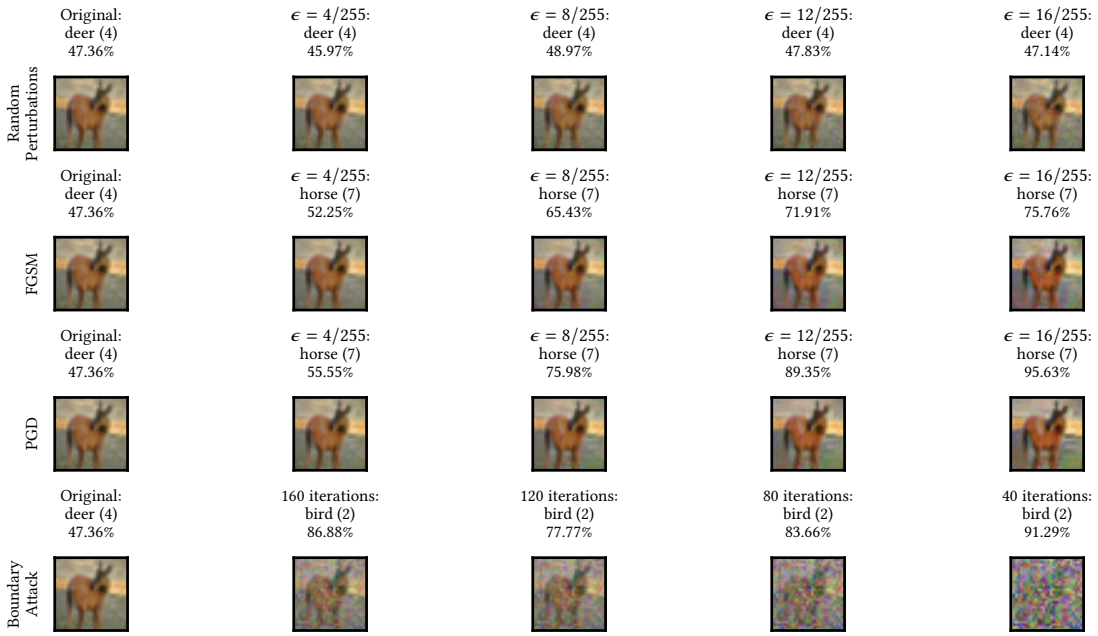
First of all, in comparison to fig. 4.3, it is noticeable that the certainties were considerably lower for the original images, where the standardly trained DNN had achieved 99 % to 100 %. Classification results were, however, still correct. The same applied for randomly perturbed images, independent of the magnitude of perturbation.

Next, results for FGSM and PGD differed between the two images: Considering the first one, misclassification occurred already at $\epsilon = 4/255$. With increasing ϵ, misclassification continued with increasing certainty. The classification result of the second image was, by contrast, wrong only after $\epsilon = 12/255$ for FGSM and $\epsilon = 8/255$ for PGD. Besides, the behavior of the increasing certainty can be observed here as well. Comparing the adversarial examples with their equivalents in fig. 4.3, one notices a change in the manipulation pattern: Where manipulation had been of *additive nature* for the standardly trained model, either being unstructured noise or actually semantic additives like the antlers, it was now rather *an amplification of the structures being already present in the natural images.* This reminds of an increase in contrast, although not necessarily the whole image was affected; for the second image, e.g., only color variations of the grass were amplified perceivably, while the horse itself remained almost unchanged.

Last, concerning the results of the Boundary Attack, one can still identify the depicted object, but the noise takes unmistakeably the prevalent part. Moreover, in comparison to fig. 4.3, it is overall more dominant. Again, the wrongly predicted class remained constant during the attack and was also the same for both images. Opposite to that, certainties tended to vary, with no clear pattern being recognizable.

Adversarial Robustness Next, fig. 4.12 visualizes the overall adversarial robustness of PreAct-ResNet-50, trained adversarially on CIFAR-10 under the different threat models.

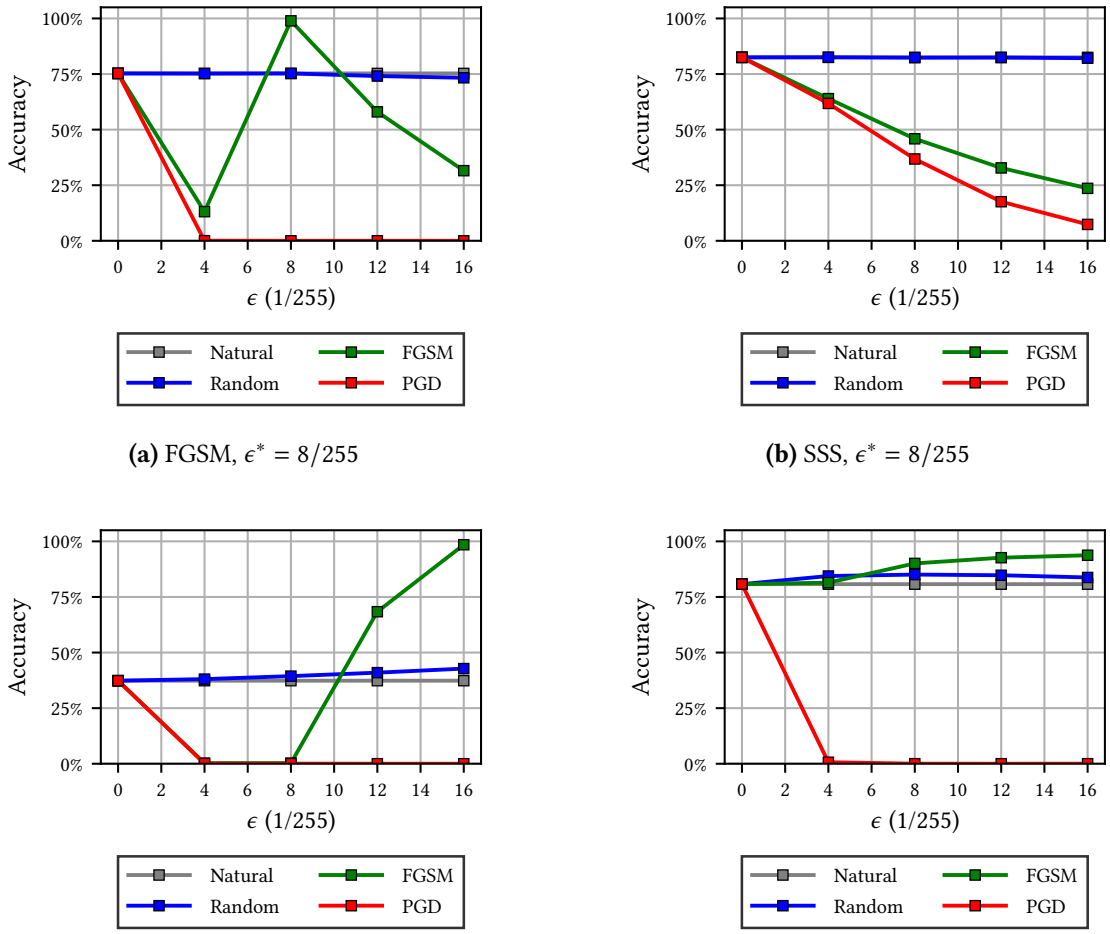
First, the natural validation accuracy of threat model (B) was overall the highest.



(a)

(b)

Figure 4.11.: Adversarial examples of CIFAR-10, created on adversarially (SSS, maximum ℓ_∞ distance $\epsilon^* = 8/255$) trained PreAct-ResNet-50 (Deer, Horse). Each row shows perturbed versions of a natural image, created with one of the adversarial attacks: random perturbations, FGSM, PGD and Boundary Attack. The original image is shown in the left column. Remaining columns show perturbed versions with maximum ℓ_∞ distance $\epsilon = 4/255, 8/255, 12/255, 16/255$ (from left to right, for random perturbations, FGSM and PGD) and after 40, 80, 120, 160 iterations of Boundary Attack (from right to left), respectively. Classification information is provided above the corresponding image.



(a) FGSM, $\epsilon^* = 8/255$

(b) SSS, $\epsilon^* = 8/255$

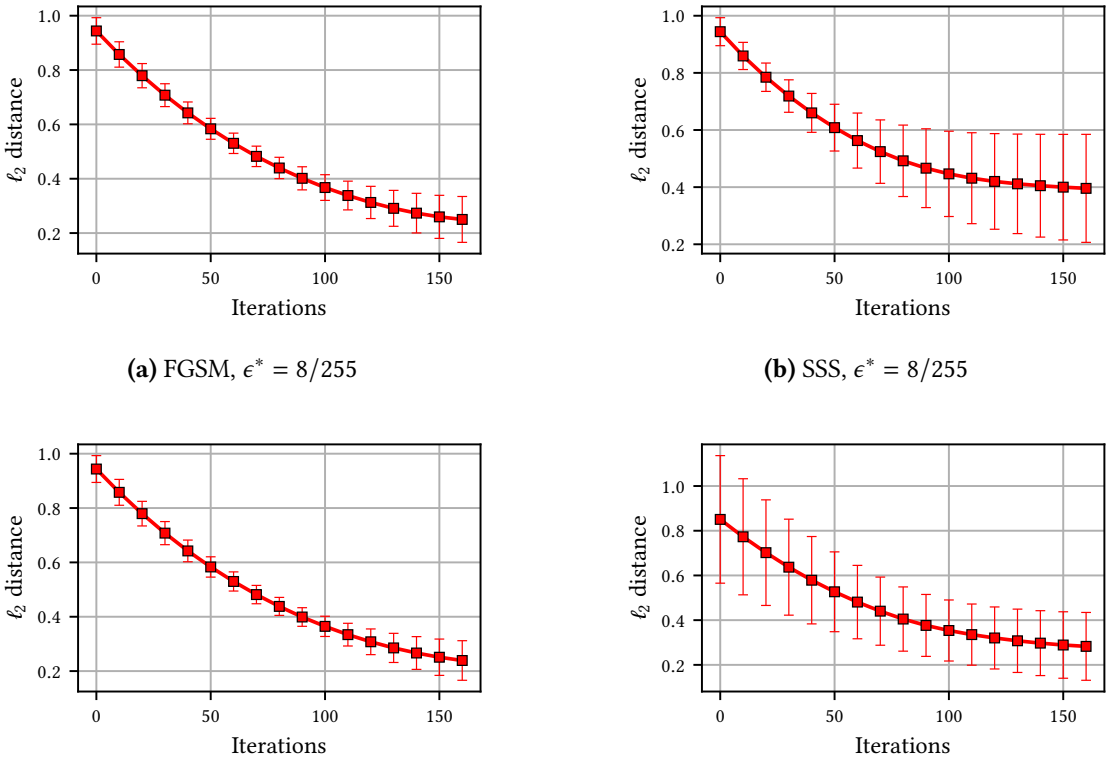
(c) FGSM, $\epsilon^* = 16/255$

(d) SSS, $\epsilon^* = 16/255$

Figure 4.12.: Adversarial robustness of PreAct-ResNet-50, trained adversarially on CIFAR-10 using FGSM (left) and SSS (right) at maximum ℓ_∞ distances $\epsilon^* = 8/255$ (top) and $\epsilon^* = 16/255$ (bottom). For all scenarios, the accuracy under the adversarial attacks of random perturbations (blue), FGSM (green) and PGD (red) was evaluated. Attacks were performed using the ∞-metric and varied in terms of the maximum permitted dissimilarity ϵ, being plotted on the x-axis. As a baseline for natural images, the validation accuracy of the trained model is shown in gray.

In addition, the natural validation accuracies were generally higher for SSS than for their FGSM counterparts (82.5 % vs. 75.28 % at $\epsilon^* = 8/255$ and 80.76 % vs. 37.35 % at $\epsilon^* = 16/255$). Note, however, that the natural validation accuracy was 92.19 % for the standardly trained DNN and therefore even higher.

Regarding the robustness against random perturbations, the behavior was similar to that after standard training, with the accuracy at about the same level as the natural accuracy. In detail, with increasing magnitude of perturbation, it slightly fell for threat model (A), but raised for threat models (C) and (D) and also exceeded the natural validation accuracy here. Threat model (B), by contrast, exhibited constant accuracy on a par with the natural validation accuracy.



(a) FGSM, $\epsilon^* = 8/255$

(b) SSS, $\epsilon^* = 8/255$

(c) FGSM, $\epsilon^* = 16/255$

(d) SSS, $\epsilon^* = 16/255$

Figure 4.13.: Development of the ℓ_2 distance during Boundary Attack on PreAct-ResNet-50, trained adversarially on CIFAR-10 using FGSM (left) and SSS (right) at maximum ℓ_∞ distances $\epsilon^* = 8/255$ (top) and $\epsilon^* = 16/255$ (bottom). For all scenarios, the average ℓ_2 distance between the target image and the current perturbed version was evaluated every 10th iteration. 160 iterations were performed in total. Standard deviations are plotted as error bars.

Moving on to FGSM and PGD, results became more diverse. For FGSM-perturbed images, FGSM-trained models showed a nearly optimal performance of about 98 % to 99 %, if the magnitude of perturbations ϵ equaled the magnitude of perturbations ϵ^* during adversarial training. Otherwise, accuracies were considerably below this, with a tendency to be higher for higher values of ϵ. PGD, on the other hand, led to the same picture as after standard training: The validation accuracy was constantly at or close to 0 %.

Threat model (B), in turn, caused the accuracies for both FGSM and PGD to continuously decrease, somewhat more so for the latter. But even at $\epsilon = 16/255$, they were still above zero (23.6 % for FGSM and 7.37 % for PGD).

Last but not least, the results of threat model (D) pointed out the same susceptibility against PGD as for threat models (A) and (C). However, against FGSM, the validation accuracy was slightly above the natural validation accuracy at $\epsilon = 4/255$. For higher values of ϵ, the validation accuracy increased here further.

Boundary Attack Figure 4.13 shows the development of the average ℓ_2 distance between the original images and the current perturbed versions during the Boundary Attack for all threat models.

The results here were quite similar for the threat models threat models (A) and (C): Average ℓ_2 distances fell exponentially, with a low standard deviation in the beginning and a slightly higher one in the end. In comparison to the results on the standardly trained model (fig. 4.5a), the initial average distance was about 0.1 and the last one only marginally greater. However, the development of the standard deviation was inverse, with the final value being approximately the same.

For threat model (B), on the contrary, the picture was now almost the same as after standard training on GTSRB (fig. 4.5b). That is, the initial average distance was on a par with that of threat models (A) and (C), but the final one was greater, measuring about 0.4. Furthermore, the change of the standard deviation was similar to that in fig. 4.5b, whereas it was even greater for the last iterations.

Eventually, considering threat model (D), the basic development of the average ℓ_2 distance equaled that after standard training (fig. 4.5a). The standard deviations were also initially approximately equal and decreased, as the attack progressed. However, after the half of iterations, it converged here, leading to a greater final value than for the standardly trained DNN.

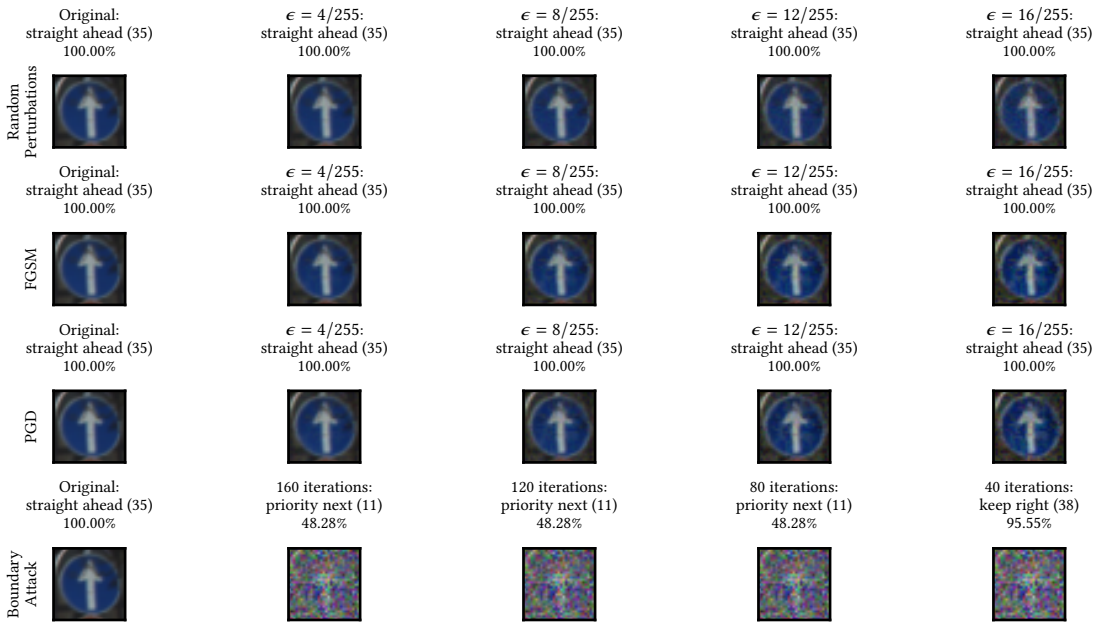
4.4.2. *GTSRB*

Adversarial Examples Analogously, fig. 4.14 shows perturbed versions of the same two GTSRB images as in fig. 4.6, this time after adversarial training.

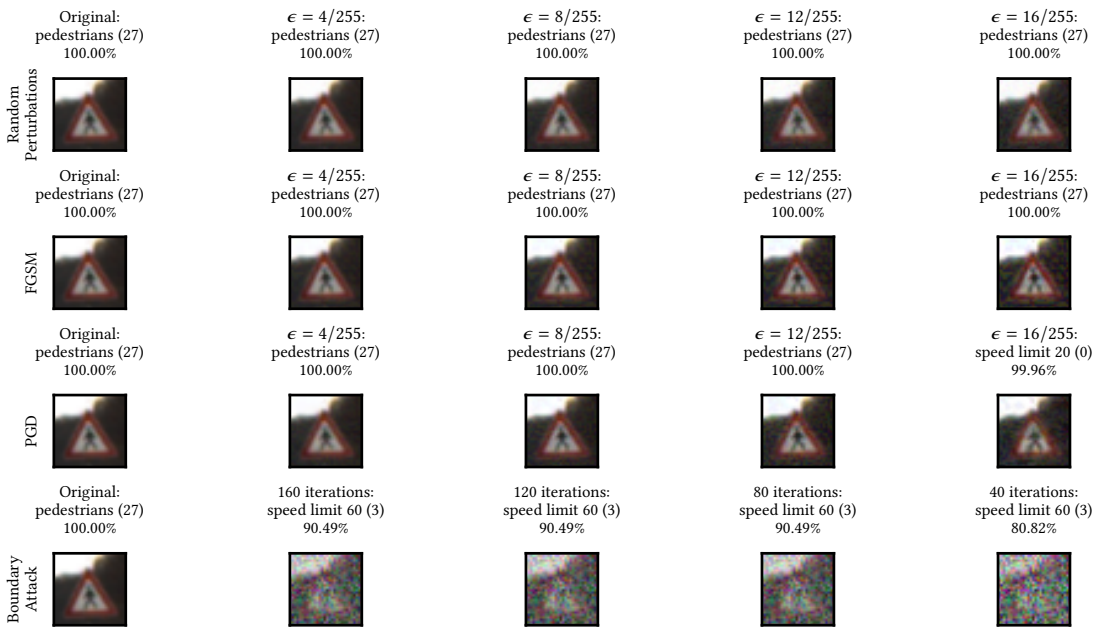
Beginning with the classification results themselves, they were all correct except for the PGD-perturbed second image at $\epsilon = 16/255$, which was wrongly classified as a "speed limit 20" sign, as well as the perturbations resulting from the Boundary Attack. Opposite to the results on CIFAR-10, certainties remained throughout at or very close to 100 %, again with the exception of some results of the Boundary Attack.

Moreover, the change in the manipulation pattern that was noticed on CIFAR-10 could not be fully confirmed here: On the one hand, semantic changes did not occur anymore or at least to a lesser extent – for the PGD-perturbed first image, the shade of a right pointing arrow's line segment was still recognizable at $\epsilon = 16/255$. But on the other hand, no significant contrast increase could be detected. At least for PGD, perturbations tended to be visually less perceptible at higher values of ϵ than with standard training.

Last, similar to the results on CIFAR-10, the Boundary Attack also led to more perceptible perturbations than on the standardly trained model. Even after 160 iterations, one can only barely recognize which traffic sign is actually depicted because of the high

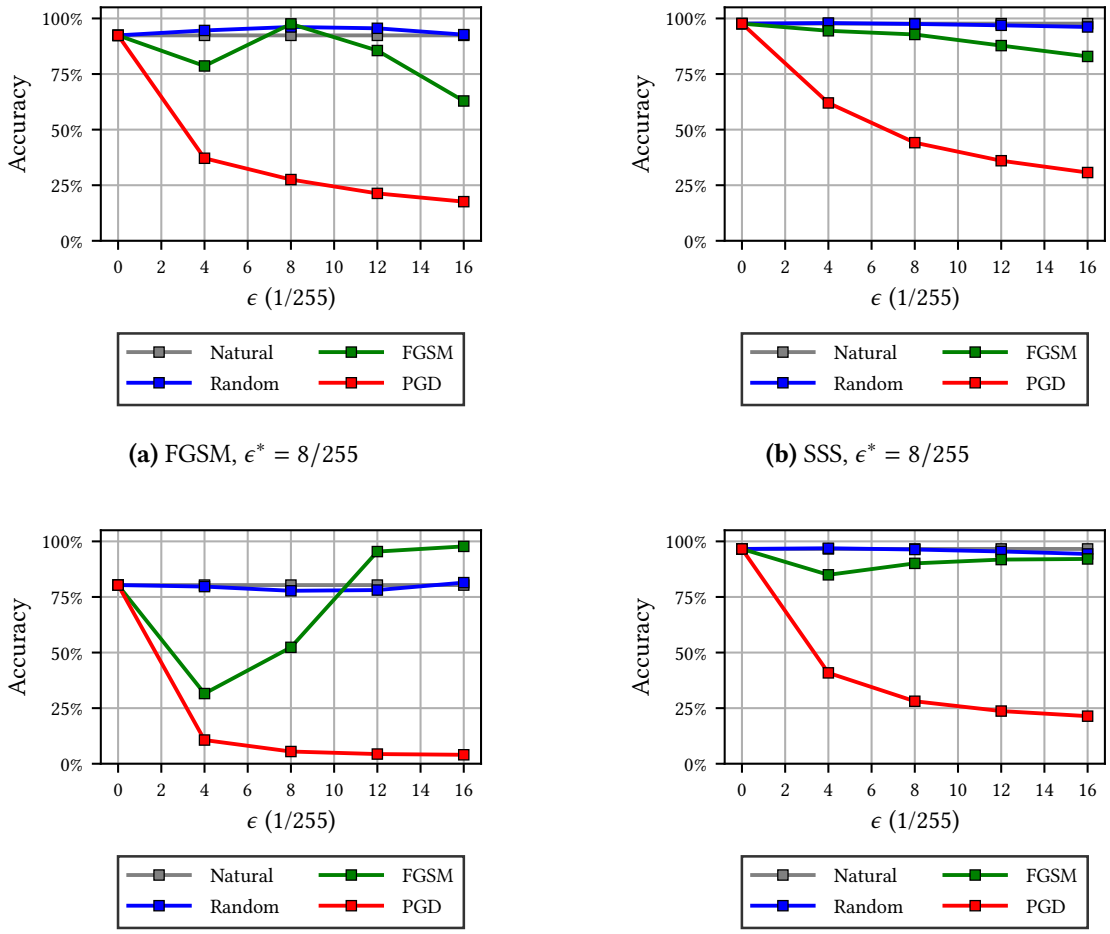


(a)



(b)

Figure 4.14.: Adversarial examples of GTSRB, created on adversarially (SSS, maximum ℓ_∞ distance $\epsilon^* = 8/255$) trained PreAct-ResNet-50 (Straight ahead, Pedestrians). Each row shows perturbed versions of a natural image, created with one of the adversarial attacks: random perturbations, FGSM, PGD and Boundary Attack. The original image is shown in the left column. Remaining columns show perturbed versions with maximum ℓ_∞ distance $\epsilon = 4/255, 8/255, 12/255, 16/255$ (from left to right, for random perturbations, FGSM and PGD) and after 40, 80, 120, 160 iterations of Boundary Attack (from right to left), respectively. Classification information is provided above the corresponding image.



(a) FGSM, $\epsilon^* = 8/255$

(b) SSS, $\epsilon^* = 8/255$

(c) FGSM, $\epsilon^* = 16/255$

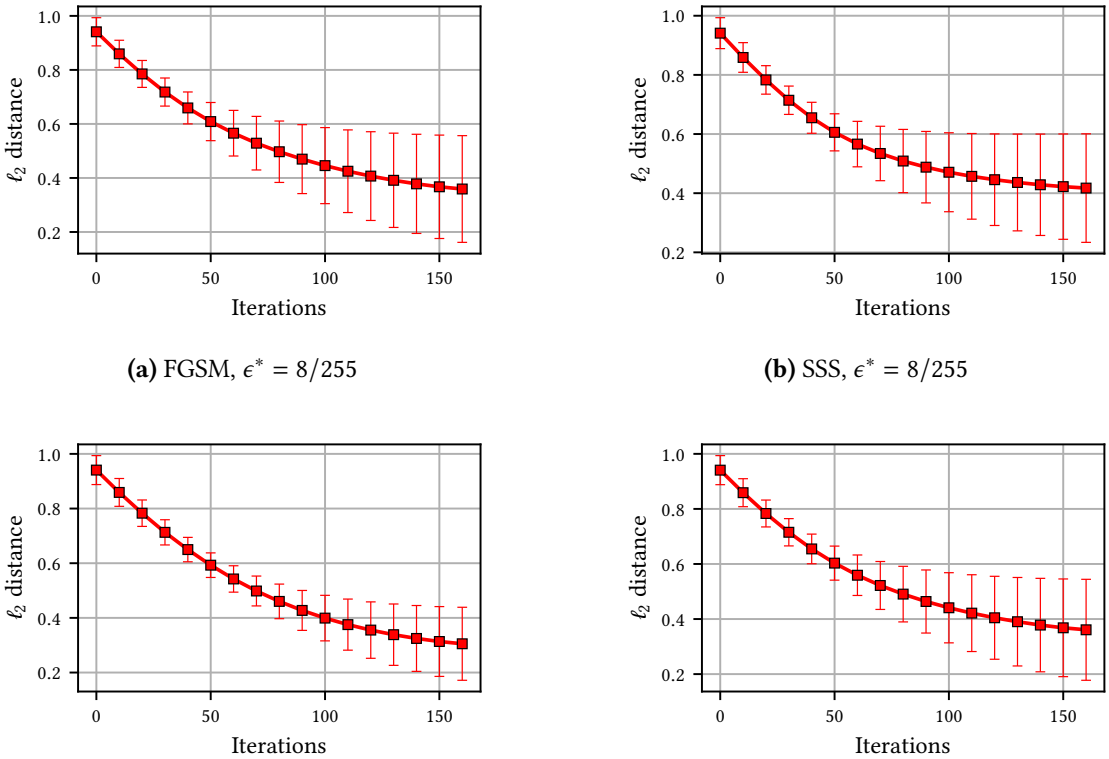
(d) SSS, $\epsilon^* = 16/255$

Figure 4.15.: Adversarial robustness of PreAct-ResNet-50, trained adversarially on GTSRB using FGSM (left) and SSS (right) at maximum ℓ_∞ distances $\epsilon^* = 8/255$ (top) and $\epsilon^* = 16/255$ (bottom). For all scenarios, the accuracy under the adversarial attacks of random perturbations (blue), FGSM (green) and PGD (red) was evaluated. Attacks were performed using the ∞-metric and varied in terms of the maximum permitted dissimilarity ϵ, being plotted on the x-axis. As a baseline for natural images, the validation accuracy of the trained model is shown in gray.

noise level.

Adversarial Robustness Figure 4.15 visualizes the overall adversarial robustness of PreAct-ResNet-50, trained adversarially on GTSRB under the different threat models.

All in all, the results are once again comparable in principle to those of CIFAR-10, although, as for standardly trained models, the absolute accuracies are consistently higher. First, the DNN trained adversarially under threat model (B) achieves the highest natural validation accuracy. Next, the natural validation accuracies were generally higher for SSS than for their FGSM counterparts (97.67 % vs. 92.38 % at $\epsilon^* = 8/255$ and 96.6 % vs. 80.35 % at $\epsilon^* = 16/255$). And again, the natural validation accuracy was lower than after



(a) FGSM, $\epsilon^* = 8/255$

(b) SSS, $\epsilon^* = 8/255$

(c) FGSM, $\epsilon^* = 16/255$

(d) SSS, $\epsilon^* = 16/255$

Figure 4.16.: Development of the ℓ_2 distance during Boundary Attack on PreAct-ResNet-50, trained adversarially on GTSRB using FGSM (left) and SSS (right) at maximum ℓ_∞ distances $\epsilon^* = 8/255$ (top) and $\epsilon^* = 16/255$ (bottom). For all scenarios, the average ℓ_2 distance between the target image and the current perturbed version was evaluated every 10th iteration. 160 iterations were performed in total. Standard deviations are plotted as error bars.

standard training (99.22 %).

Furthermore, the robustness against random perturbations remains principally on a high level, with accuracies close to the natural validation accuracy. In some cases, it slightly exceeds the natural validation accuracy, in others, it is just as slightly below.

Concerning FGSM perturbations, there is again the big difference between models trained adversarially with FGSM and those trained with SSS. The former showed almost perfect validation accuracies of roughly 98 %, if $\epsilon = \epsilon^*$ held. In turn, the higher the difference was between them, the lower the accuracy was. For threat model (B), by contrast, it decreased with increasing ϵ, resulting in 94.45 % at $\epsilon = 4/255$ and 82.91 % at $\epsilon = 16/255$. Eventually, the model which trained with threat model (D) exhibited inverse behavior, as the validation accuracy steadily increased from $\epsilon = 4/255$ (84.96 %) to $\epsilon = 16/255$ (92.1 %).

Last, considering PGD, the validation accuracy decreased slightly (threat model (C)) to moderately (threat model (B)) for increasing ϵ. However, it was consistently higher

for $\epsilon^* = 8/255$ than for $\epsilon^* = 16/255$. Also, when comparing SSS-based training with FGSM-based, the former led throughout to higher accuracies, independent of ϵ^*.

Boundary Attack Last but not least, fig. 4.16 visualizes the development of the average ℓ_2 distance between the original images and the current perturbed versions during the Boundary Attack on GTSRB, considering threat models (A) to (D).

This time, results were highly similar for the threat models threat models (A), (B) and (D) and also close to those after standard training (see fig. 4.5b): Average ℓ_2 distances fell exponentially, with a low standard deviation in the beginning and a slightly higher one in the end. The only difference is that for threat model (B), both the average value itself and the standard deviation are slightly higher on convergence.

Eventually, for threat model (C), basic tendencies were the same, but both the final ℓ_2 average distance and the final standard deviation were somewhat lower than for other cases.

4.5. INVESTIGATION OF THE LOSS LANDSCAPE

4.5.1. *CIFAR-10*

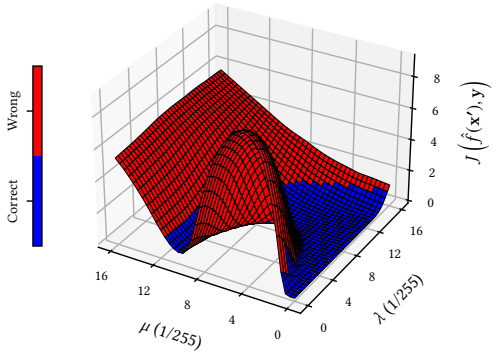
Figure 4.17 shows the loss landscape for the same natural image as in fig. 4.3a and the differently trained DNNs as a three-dimensional surface plot.

First of all, one sees that the adversarial direction, pointing to the left and being scaled by the parameter μ, was significantly more susceptible to an increase of loss than the random direction, which points to the right and is scaled by λ: For all DNNs, the first misclassification occurred here already in the range $0 \leq \mu \leq 4$ and the loss raised steeply to its peek. By contrast, it remained on a relatively constant level for increasing λ.

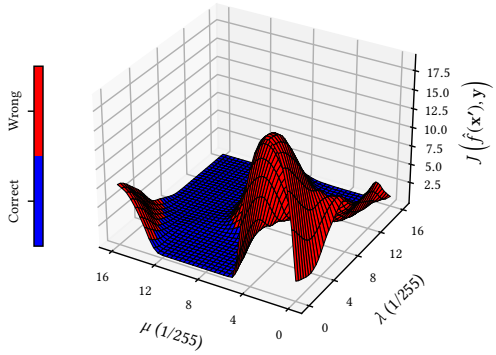
A major difference between the networks trained adversarially with FGSM and the rest is the natural image itself was classified wrongly and exhibited a comparatively high loss value for the former.

Besides, the scenarios' plots differ in what happens after that peak in the adversarial direction, if there is an after: First, for the standardly trained network as well as the one trained adversarially with threat model (A), there was another local minimum of loss at $\mu \approx 10/255$ and $6/255 \leq \mu \leq 12/255$, respectively, before it started to increase again. Second, for threat models (C) and (D), the loss remained on a relatively constant level. Nevertheless, with the latter, some slight fluctuations are recognizable, leading to a single misclassification at $\mu \approx 10/255$. Last, for threat model (B), the peak was only reached at $\mu = 16/255$, as the loss increased nearly linearly over the entire observed range.

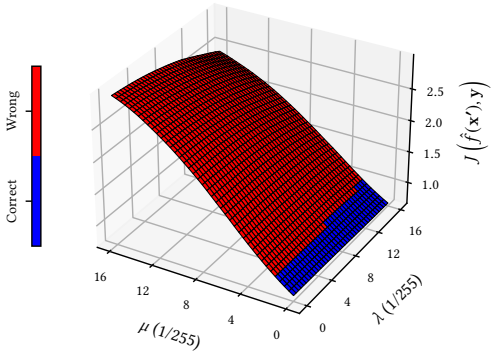
Furthermore, there are differences in the height of loss as well: For the DNNs that



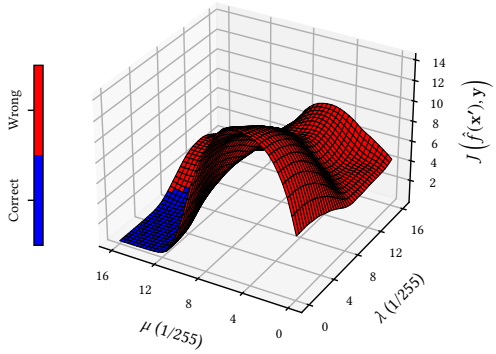
(a) Standard Training



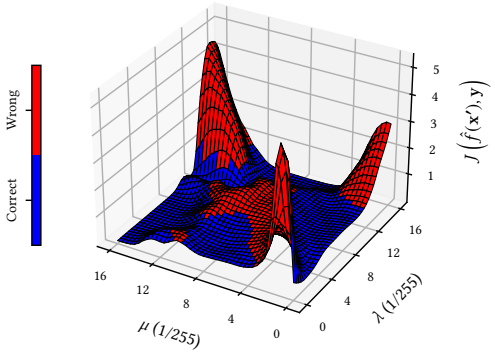
(b) Adversarial training: FGSM, $\epsilon^* = 8/255$



(c) Adversarial training: SSS, $\epsilon^* = 8/255$

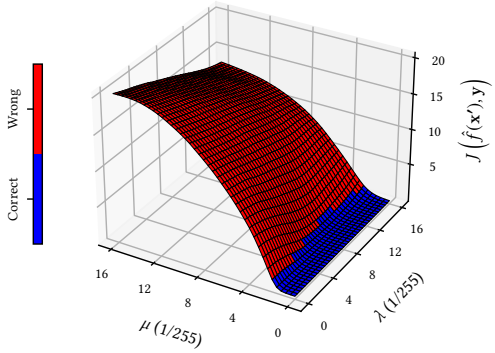


(d) Adversarial training: FGSM, $\epsilon^* = 16/255$

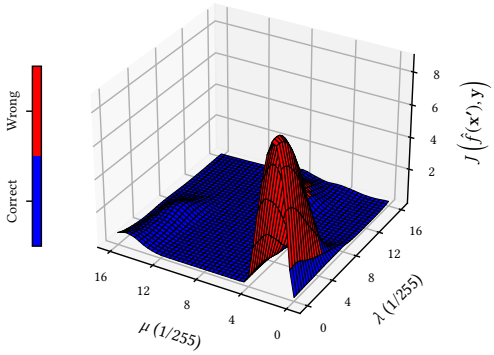


(e) Adversarial training: SSS, $\epsilon^* = 16/255$

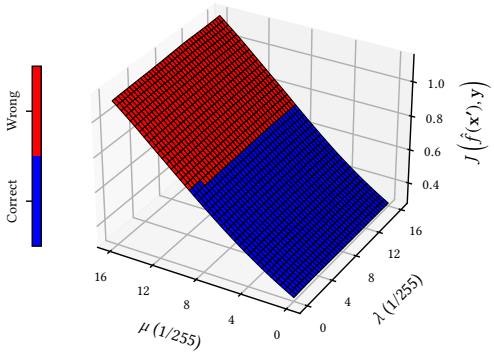
Figure 4.17.: Loss landscapes from PreAct-ResNet-50, trained standardly (top) and adversarially using FGSM (left) and SSS (right) at maximum ℓ_∞ distances $\epsilon^* = 8/255$ (middle) and $\epsilon^* = 16/255$ (bottom) on CIFAR-10 (Deer). In all cases, the natural image from the validation dataset (see fig. 4.3a) marks the origin. Moreover, the left axis points in the adversarial direction and the right one in a random one. Both are scaled in multiples of 1/255 according to the ∞-norm. Last, correctly classified perturbations are colored blue, wrongly ones red.



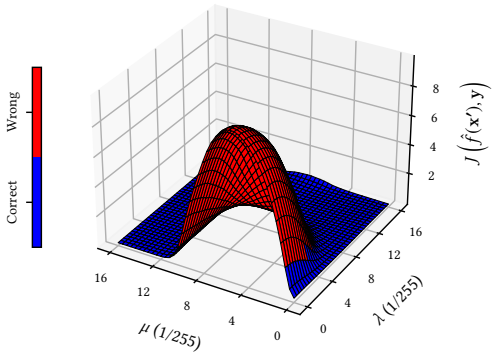
(a) Standard Training



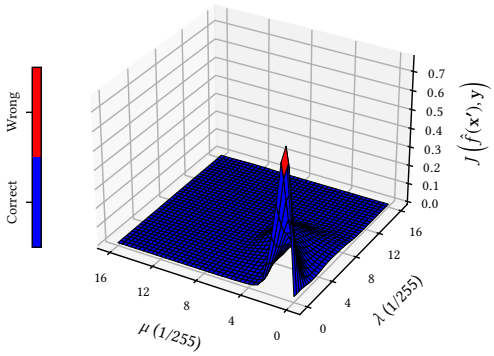
(b) Adversarial training: FGSM, $\epsilon^* = 8/255$



(c) Adversarial training: SSS, $\epsilon^* = 8/255$



(d) Adversarial training: FGSM, $\epsilon^* = 16/255$



(e) Adversarial training: SSS, $\epsilon^* = 16/255$

Figure 4.18.: Loss landscapes from PreAct-ResNet-50, trained standardly (top) and adversarially using FGSM (left) and SSS (right) at maximum ℓ_∞ distances $\epsilon^* = 8/255$ (middle) and $\epsilon^* = 16/255$ (bottom) on CIFAR-10 (Horse). In all cases, the natural image from the validation dataset (see fig. 4.3b) marks the origin. Moreover, the left axis points in the adversarial direction and the right one in a random one. Both are scaled in multiples of $1/255$ according to the ∞-norm. Last, correctly classified perturbations are colored blue, wrongly ones red.

were trained adversarially with FGSM, the maximum loss was by a factor of 1.5 to 2 higher than for the standardly trained model. For SSS-trained ones, on the other hand, it was proportionally about the same much lower than for the latter, with threat model (B) showing the lowest.

Finally, concerning the interaction of adversarial and random perturbation, loss values in most cases change when applying random perturbation in addition to sole adversarial perturbation. Threat model (B) is the only one which does not show this behavior.

Next, fig. 4.18 also visualizes the loss landscape for the natural image depicted in fig. 4.3b.

In general, there are many similarities to fig. 4.17, so we focus on the differences: First, the loss landscape after standard training has a fundamentally different appearance, more resembling fig. 4.17c with a slightly higher curvature. Second, the FGSM-trained DNNs classified the unperturbed or randomly perturbed image this time correctly. Third, for the model that was trained adversarially with threat model (B), classification was now correct until $\mu \approx 9/255$. Forth, the combination of adversarial and random perturbations had in most cases the same effect as sole adversarial perturbation. Here, only the standardly trained DNN exhibited a different behavior, with a slight decrease of loss in the direction of random perturbation. Last but not least, loss values were throughout lower than in the corresponding scenarios w.r.t. the first image, except for the standardly trained model as well.

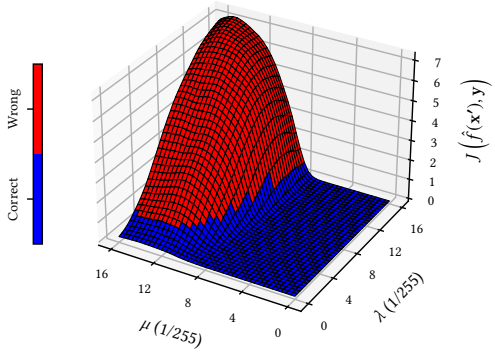
4.5.2. *GTSRB*

Figure 4.19 shows analogously the loss landscape for the same natural image as in fig. 4.6a and the differently trained DNNs as a three-dimensional surface plot.

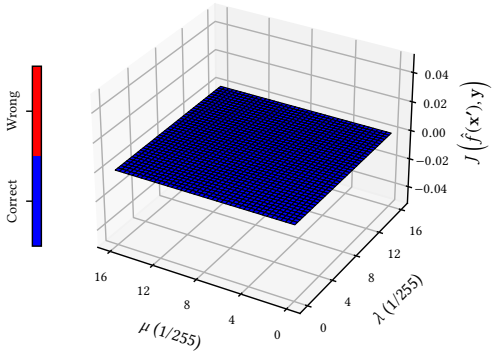
Particularly in comparison with the loss landscapes on CIFAR-10, it stands out that the network trained adversarially with threat models (A), (B) and (D) do not exhibit any areas of misclassification at all. Concerning the standardly trained DNN, neither sole adversarial nor random perturbation did cause misclassification as well, but the combination of both. Last, for the DNN resulting from adversarial training with threat model (C), misclassification occurred roughly in the range $1/255 \leq \mu \leq 8/255$ for sole or nearly sole adversarial perturbations, with a loss maximum at $\mu \approx 4$. Moreover, simultaneously maximized adversarial and random perturbation caused another local maximum of loss and also misclassification.

Eventually, fig. 4.20 visualizes the loss landscapes w.r.t. the natural image shown in fig. 4.6b.

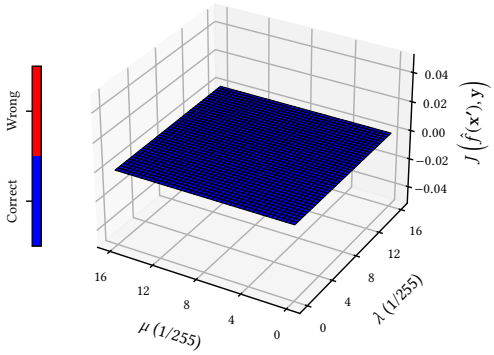
Again, the loss landscapes are quite flat overall and in particular compared to previous results – in cases of peaks, note the scaling of the vertical axis. The model that was trained



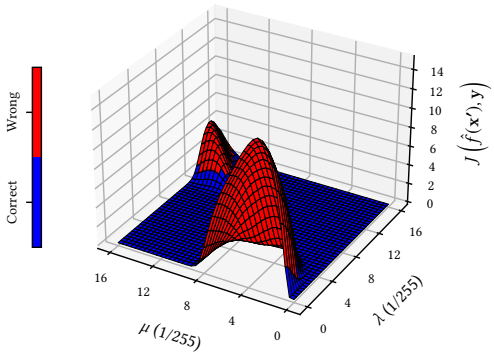
(a) Standard training



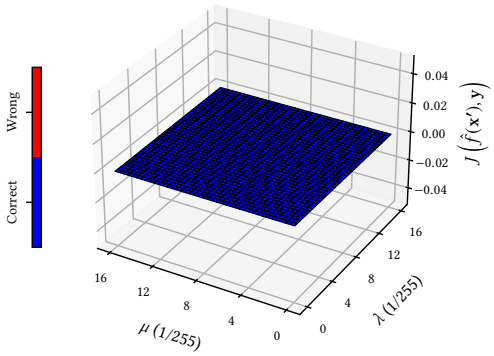
(b) Adversarial training: FGSM, $\epsilon^* = 8/255$



(c) Adversarial training: SSS, $\epsilon^* = 8/255$

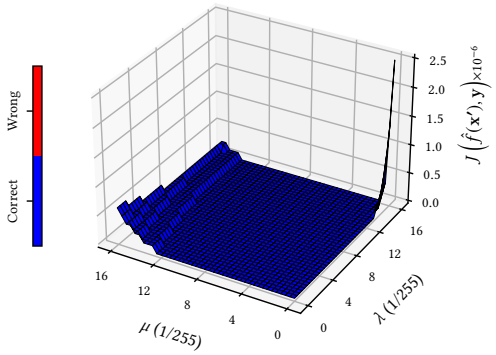


(d) Adversarial training: FGSM, $\epsilon^* = 16/255$

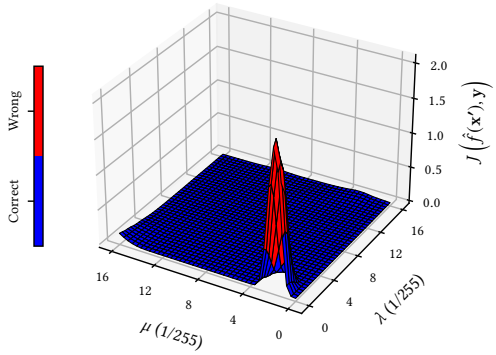


(e) Adversarial training: SSS, $\epsilon^* = 16/255$

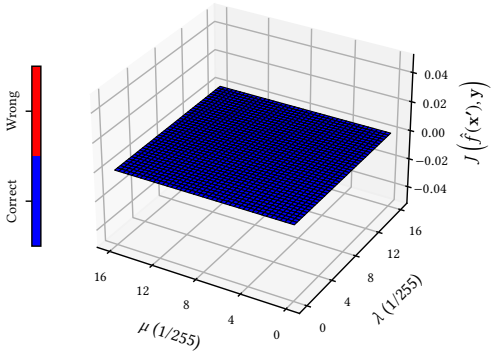
Figure 4.19.: Loss landscapes from PreAct-ResNet-50, trained standardly (top) and adversarially using FGSM (left) and SSS (right) at maximum ℓ_∞ distances $\epsilon^* = 8/255$ (middle) and $\epsilon^* = 16/255$ (bottom) on GTSRB (Straight ahead). In all cases, the natural image from the validation dataset (see fig. 4.6a) marks the origin. Moreover, the left axis points in the adversarial direction and the right one in a random one. Both are scaled in multiples of $1/255$ according to the ∞-norm. Last, correctly classified perturbations are colored blue, wrongly ones red.



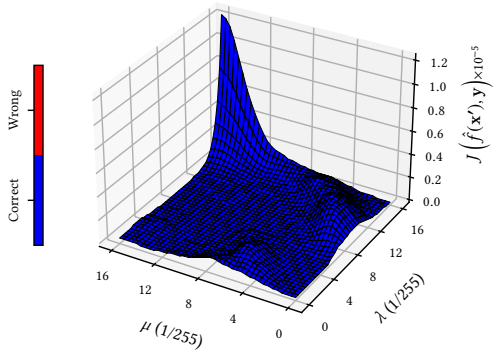
(a) Standard training



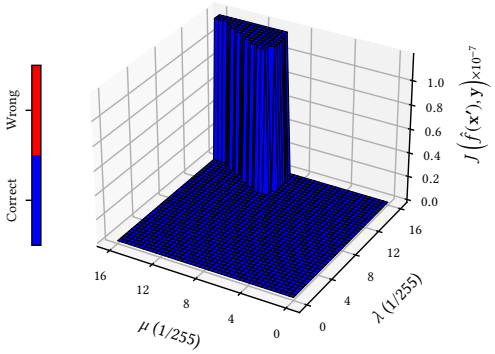
(b) Adversarial training: FGSM, $\epsilon^* = 8/255$



(c) Adversarial training: SSS, $\epsilon^* = 8/255$



(d) Adversarial training: FGSM, $\epsilon^* = 16/255$



(e) Adversarial training: SSS, $\epsilon^* = 16/255$

Figure 4.20.: Loss landscapes from PreAct-ResNet-50, trained standardly (top) and adversarially using FGSM (left) and SSS (right) at maximum ℓ_∞ distances $\epsilon^* = 8/255$ (middle) and $\epsilon^* = 16/255$ (bottom) on GTSRB (Pedestrians). In all cases, the natural image from the validation dataset (see fig. 4.6b) marks the origin. Moreover, the left axis points in the adversarial direction and the right one in a random one. Both are scaled in multiples of 1/255 according to the ∞-norm. Last, correctly classified perturbations are colored blue, wrongly ones red.

adversarially with threat model (A) is the only exception, with a maximum loss of nearly 2 at $\mu \approx 2/255, \lambda = 0$. Besides, the latter was the only one that exhibited misclassification at the mentioned peak of loss.

5 DISCUSSION

In this chapter, we evaluate the results presented in chapter 4 and discuss possible reasons why they turned out the way they did. Besides, we point out aspects of this work which leave room for further research in the future. The chapter is structured by nine questions, for each of which a plausible answer is tried to be given.

1. Why are DNNs susceptible to adversarial perturbations, but not to random perturbations? The plots on adversarial robustness (see, e.g., fig. 4.4) and on loss landscapes (see section 4.5) indicate that random perturbations – opposite to adversarial perturbations – are only barely able to increase the loss significantly enough to cause misclassification. A plausible reason for this observation would be that the set of all adversarial examples for a given dataset and classifier spans a comparatively low-dimensional subspace in the input space. Therefore, randomly perturbed images are unlikely to be part of that subspace.

Indeed, Tramèr et al. [59] have shown in 2017 that the adversarial examples for the 784-dimensional MNIST dataset span an approximately 15-dimensional subspace, subject to a trained DNN with convolutional layers. If a fully-connected, non-convolutional DNN is used, the subspace is somehow larger (about 45 dimensions), but still low-dimensional in comparison to the input space itself. So while the actual size of the subspace might depend on aspects like the considered dataset, the used DNN architecture and also various other hyperparameters of training, it can be assumed that the basic statement regarding a lower-dimensional adversarial subspace is valid, explaining the inherent robustness against random perturbations.

2. How can the observed development of ℓ_2 distances during the Boundary Attack be explained? The results on the Boundary Attack show that the average ℓ_2 distance between the target image is roughly exponentially decreasing with progressing attack, but the detailed course and – more striking – the development of standard deviations differ between different training settings and datasets (see figs. 4.5, 4.13 and 4.16). Interpreting the results, it actually turns out that they reflect the robustness of the different models against adversarial perturbations quite well:

First of all, the average ℓ_2 distances are generally lower for CIFAR-10 than for GTSRB. As a low ℓ_2 distance correlates with a low visual dissimilarity, this coincides with the

observation that DNNs trained on CIFAR-10 are generally more susceptible to adversarial perturbations than those trained on GTSRB (see, e.g., fig. 4.4).

Second, SSS-based adversarial training with $\epsilon^* = 8/255$ has proven to increase the robustness against adversarial perturbations the most. In an analogous manner, the average ℓ_2 distances are here at least on a par with other threat models of adversarial training during the attack, but throughout higher when it is finished.

Third, concerning the standard deviations of ℓ_2 distances for standardly trained DNNs, they are initially high and eventually low for CIFAR-10, with the opposite holding for GTSRB. This reflects the observed difference in achieved robustness between both datasets as well: For CIFAR-10, a high initial standard deviation means that there is some probability that the initial perturbation is even closer to the target image. In consequence, there is a higher probability that some smaller random perturbation leads to misclassification than for GTSRB, which is suggested by fig. 4.4 too. On the other hand, there is also a higher probability that a larger than average random perturbation is needed to initialize adversarially, but note that the average ℓ_2 distance is initially lower than for GTSRB. With a lower final standard deviation, in turn, almost all created adversarial examples are similarly close to the target image. In combination with the fact, that the final average ℓ_2 distance is lower for CIFAR-10, this means that almost all created adversarial examples are that close to the target image, with only few exceptions. Regarding GTSRB, the opposite holds: It is both more difficult to initialize adversarially with a high ℓ_2 distance and to get close to the target image, while staying adversarial.

Forth, the standard deviations achieved on the DNNs trained adversarially with threat model (B) further verify that this specific setting leads to the highest increase of robustness: The results for CIFAR-10 now nearly equal those for GTSRB after standard training. Moreover, the final standard deviations for GTSRB after adversarial training are even higher than those after standard training.

3. Why do adversarial examples look different after adversarial training? This question refers to two observations: First, adversarial examples generated on standardly trained DNNs more often showed structural elements of a second class, as which they are wrongly classified (see, e.g., figs. 4.3b and 4.6b). Second, especially adversarial examples of CIFAR-10 images, created on adversarially trained DNNs tend to exhibit changes from the natural image in larger, already existing structures (see, e.g., figs. 4.11a and C.5a) than after standard training (see, e.g., figs. 4.3a and B.1a), where one would interpret the perturbation more as fine-grained noise.

Beginning with the former, one has to note that this kind of perturbations only occur with PGD at very high magnitudes of perturbation. It can be assumed that these per-

turbations exhibit a significantly high loss in the considered closed ℓ_∞ ball, otherwise PGD would not end up with them. This makes sense, as the classification becomes more difficult for humans as well; with semantic changes, a misclassification can no longer be clearly considered a misclassification. However, we cannot give a definite reason why PGD does not find the path to these special adversarial examples anymore, when being performed on a DNN that was trained with single-step-based adversarial examples. Perhaps, the single-step-based adversarial examples become somehow close to those "mixed images" in the course of training, so the model gets adapted to them. In any case, for a standardly trained DNN, they are not.

Concerning the second observation, the different size of structures suggests a connection to the used DNN architecture, which incorporates convolutional layers. Therefore, it would be interesting to study the phenomenon for different architectures in some future work. Unfortunately, there is no immediate explanation how adversarial training is involved in this and why it apparently only occurs with the CIFAR-10 dataset, but not with the GTSRB dataset.

4. Why does adversarial training work at all? While it is trivial that a DNN is adapted to what it is confronted with during training, this question is aimed at the following: The adversarial perturbations that are created during training are specifically computed to maximize the loss at this time. Then, why does robustness gained against adversarial perturbations of the training data cause a gain of robustness against adversarial perturbations of the validation data, where the training dataset and the validation dataset are disjoint?

The observation of the latter implies that different adversarial perturbations must be somehow correlated. We know from standard training that DNNs are able to generalize from training data to unknown validation data to some extent, where both of them reside in the input space. This is actually possible, as the data w.r.t. a specific classification task is inherently correlated. Consequently, the so-called *manifold hypothesis* assumes that the data actually lies on a lower-dimensional manifold inside the input space [60].

At this point, the above introduced concept of a low-dimensional adversarial subspace could possibly explain the facts once again. We can interpret this low-dimensional adversarial subspace as a low-dimensional manifold embedded in the input space as well, on which all adversarial examples reside. Then, the same logic according to which standard training is possible extends to the adversarial training, with the little caveat that the manifold of natural data remains constant during training, while the manifold of adversarial examples continuously changes, as the loss landscape changes. These changes must be negligible to a limited extent at least for single-step-based adversarial training,

because the training loss measured during adversarial training clearly decreased (see, e.g., fig. 4.7), indicating progressing adaption to single-step-based adversarial examples. Although not being verified in this work, we can assume that the same holds for PGD-based adversarial training, according to the results of Madry et al. [21].

However, the generalization to unknown adversarial examples, e.g. created from a withheld validation dataset, worked differently well: For the CIFAR-10 dataset, FGSM-based adversarial training caused catastrophic overfitting (see figs. 4.7 and 4.8). By contrast, SSS-based adversarial training with small ϵ^* led to comparatively remarkable generalization[1] both to FGSM-based and PGD-based adversarial examples (see fig. 4.7). In turn, SSS-based adversarial training with high ϵ^* resulted in catastrophic overfitting again (see fig. 4.8). And last, using the GTSRB dataset, there was overfitting in varying degrees, depending on the concrete threat model of adversarial training (see figs. 4.9 and 4.10).

Actually, in all scenarios, the generalization to unknown adversarial examples after adversarial training worked worse than the generalization to natural images after standard training. At the same time, the classification accuracy on the training dataset was definitely comparable between adversarial and standard training (see, e.g., figs. 4.1b and 4.7c). Hence, we can draw conclusion that adversarially trained DNNs generally suffered even more from overfitting than standardly trained ones, which already exhibited signs of overfitting (see figs. 4.1 and 4.2). The answer to the following question on catastrophic overfitting deals with an explanation for that.

5. Why does catastrophic overfitting occur? The results of this work confirm the work of Wong, Rice, and Kolter [54], stating that FGSM-based adversarial training leads to catastrophic overfitting at least for the CIFAR-10 dataset (see figs. 4.7a and 4.8a). Moreover, this work shows that it also occurs for SSS-based adversarial training with too high ϵ^*.

First of all, resorting to the previous answer, we can treat catastrophic overfitting as a special case of overfitting that occurs generally in adversarial training. Problematic is particularly the degree of overfitting, which turned out to be much higher than for standardly trained models. Therefore, the better question is: Why are adversarially trained DNNs generally that susceptible to overfitting, and what are the aspects that further increase the susceptibility up to catastrophic overfitting?

Beginning with the first part of the question, we first take the manifold hypothesis for granted. Consequently, we can assume that the ability of a DNN to generalize from known training data to unknown validation data strongly depends on the complex-

[1] This is to be seen with emphasis on "comparatively". In absolute terms, the robustness is still quite low.

ity/dimensionality of the manifold, on which all the data belonging to the considered data distribution resides. More specifically, generalization should work better the less complex/lower-dimensional that manifold is, and vice versa.

Now, we observed that adversarially trained models overfit more strongly to known adversarial examples and generalize more poorly to unknown adversarial examples than standardly trained models overfit to known natural images and generalize to unknown natural images. We only used single-step adversarial attacks in this work, so the reason for the observation could be that we only covered a highly limited part of the manifold of all adversarial examples during adversarial training, i.e., the adversarial examples that can be created with single-step methods, which possibly form a less complex/lower-dimensional manifold themselves. As a result, we cannot expect generalization to data that is not even part of the manifold being considered during training, such as certain PGD-based adversarial examples. The fact that there exist other adversarial examples not being reachable by single-step methods is also evident from the conducted qualitative investigation of the loss landscape, where the combination of the adversarial and some random direction sometimes pointed to further local maxima (see, e.g., fig. 4.19). This is due to the non-concavity of the loss function.

By contrast, others [21, 53] have shown that PGD-based adversarial training does not have significant benefits over single-step-based adversarial training concerning the generalization to unknown adversarial examples. Hence, restriction to single-step-based adversarial training cannot be the main cause.

Another hypothesis that could reason the poor generalization is that *the manifold of adversarial examples is generally more complex/higher-dimensional than that of the natural images.* This, however, would have to be investigated in some future work. If it is true, it might be connected with deficiencies of the used DNN architectures.

Regarding the second part of the question, three factors actually worsened the generalization: adversarial training on CIFAR-10 instead of GTSRB, the usage of FGSM-based instead of SSS-based adversarial training, and higher values of ϵ^*. The first of these is covered by the answer to question 9.

The second one can be explained simply by the fact that adversarial examples created in SSS-based adversarial training are more diverse than those generated in FGSM-based adversarial training. This is because of the variable magnitude of perturbation as well as the random initialization that is used by the SSS algorithm.

Last, concerning too high values of ϵ^*, we have to differentiate between FGSM-based and SSS-based adversarial training. For the former, the reason is exactly what motivated the development of the latter: The loss landscape then tends to exhibit local maxima in the areas between a natural image $\mathbf{x}$ and the surface of the closed ℓ_∞ ball $\overline{\mathcal{B}}_\mathcal{X}(\mathbf{x}, \epsilon^*)$, which

was shown by the performed loss landscape investigation as well (see, e.g., fig. 4.18). For SSS-based adversarial training, in turn, this principally occurred as well, but between the surface of the closed ℓ_∞ ball $\overline{\mathcal{B}}_{\mathcal{X}}(\mathbf{x}, \epsilon_1^*)$ and the natural image $\mathbf{x}$, where ϵ_1^* is the lowest magnitude of perturbation in the discretization of ϵ^* (see, e.g., fig. 4.18 as well). A solution to that could be to increase the level of discretization L. While Kim, Lee, and Lee [53] already tested this and could not achieve significant improvements, the situation could possibly change when combining this with a non-equidistant discretization, which is dense close to the natural image and becomes coarser with increasing distance. Checking if and to which extent this approach works is to be done in some future work as well.

6. Why does adversarial training cause a lower accuracy w.r.t. natural images? This question refers to the result that the classification accuracy of adversarially trained DNNs w.r.t. natural images turned out to be lower than for their standardly trained counterparts (see figs. 4.4, 4.12 and 4.15).

This could plausibly be related to the fact that we replaced the natural images completely by adversarial examples with the implementation of adversarial training in this work. Indeed, results show that the natural validation accuracy was lower the higher the magnitude of perturbation ϵ^* was, i.e., the farther the adversarial examples used for training were from the natural images (see figs. 4.12 and 4.15). In addition, for a constant value of ϵ^*, SSS-based adversarial training led to a higher natural validation accuracy than FGSM-based; that fits into the hypothesized answer to this question, since the effective magnitude of perturbation in SSS-based adversarial training is still bounded by ϵ^*, but lower on average.

As part of some future work, one could try different ratios between the adversarial examples and natural images used for adversarial training and check, in how far it increases the natural validation accuracy. At the same time, the influence on the adversarial robustness would have to be investigated.

As an alternative that can do without a further hyperparameter, it would be interesting to adapt the SSS algorithm. In detail, when no adversarial example is found for the considered discretization of ϵ^*, the natural image is fed into the training process instead of the perturbation with maximum magnitude of perturbation ϵ^*. Thus, natural images and adversarial examples would balance each other dynamically; the basic focus is still on the adversarial training, but the more the model is apparently robust against certain perturbations, the more training is based on natural images. The latter, in turn, could cause some decrease of robustness again, which is tackled in subsequent iterations of training by increasing the part of adversarial examples again.

7. Are adversarial examples inevitable? The identical question was tackled by the work of Shafahi et al. [61], which was published in 2018. Basically, they used so-called isoperimetric inequalities to give a lower bound on the probability that for an arbitrary classifier and any natural image with ground truth class c, it is either misclassified or an adversarial example with a certain maximal magnitude of perturbation exists, subject to some p-metric.

We do not discuss their work here in detail, but while being quite tight for the Euclidean metric, this lower bound turned out to be rather loose w.r.t. the ∞-metric, which is mainly considered in this work. Nevertheless, their formula states that the lower bound of the aforementioned probability is proportional to the so-called *intra-class variance* of the class c in the considered data distribution. Vividly, the intra-class variance measures the degree of spread of all images $\mathbf{x}$ in the data distribution ρ_{data} that are labeled as c.

Now, even if the lower bound is loose w.r.t. the ∞-metric, the tendency remains the same, with a higher intra-class variance implying a higher probability that an adversarial example exists for any natural image that is classified correctly by some classifier. Besides, the authors mentioned that this looseness might be rather a result of difficult proving than being something fundamental in case of using the ∞-metric.

Last, they concluded that while the exact properties of real-world image distributions are unknown, especially high-resoluted images with a high level of detail also suggest high intra-class variances. Note that their statements are independent of a DNN's concrete architecture. Thus, on the one hand, the existence of adversarial examples might be a fundamental problem that is not solvable in general. On the other hand, they stress that in addition to the properties of the data distribution, both the considered dissimilarity metric d and the maximum permitted dissimilarity score ϵ still highly influence the actual risk.

In this work, we focused mainly on the ∞-metric, except for the Boundary Attack, which is based on the Euclidean metric. To consider other dissimilarity metrics in some future work would potentially promote further insights into the nature of adversarial examples. The same applies to targeted adversarial examples, which were not regarded here as well.

8. Why are models trained on CIFAR-10 dataset more vulnerable to adversarial examples than those trained on GTSRB? The results of this work show that the rate of misclassification after adversarial attacks is generally higher for DNNs trained on the CIFAR-10 dataset than those trained on GTSRB (see figs. 4.4, 4.12 and 4.15).

There are two different perspectives to explain this. For the first one, we simply resort to the previously introduced concept of the intra-class variance. It intuitively has to

be larger for CIFAR-10, as this dataset contains images of animals and objects like cars, having different breeds and subtypes with different appearance, respectively. GTSRB, on the other hand, contains standardized traffic signs. Then, with the statement made by Shafahi et al. [61] (see question 7), it is clear that CIFAR-10 is more susceptible to adversarial examples than GTSRB.

For the second one, we have to assume that the manifold hypothesis and the hypothesis of an adversarial manifold being more complex/higher-dimensional than the manifold of natural images (see question 5) are true. Then, the fact that models being trained standardly on the CIFAR-10 dataset generalize worse to natural validation images than those being trained standardly on GTSRB (see figs. 4.1 and 4.2) points out that the natural data manifold of the former has to be more complex/higher-dimensional than that of the latter. This is also strongly connected to the above stated higher intra-class variance of CIFAR-10. Finally, it is very likely that the adversarial manifold of the CIFAR-10 dataset is more complex than the adversarial manifold of GTSRB, leading to a worse generalization to unknown adversarial examples after adversarial training on CIFAR-10 and thus higher susceptibility to adversarial attacks. But just as for the basic hypothesis regarding adversarial manifolds, this data distribution-dependent complexity/dimensionality of the adversarial manifold has to be verified in some future work. To do so, it would especially be interesting to consider higher dimensional image datasets such as the ImageNet dataset [62].

9. Are adversarial examples a real threat? To the end of this chapter, we discuss whether adversarial examples actually pose a threat in real applications of DNN-based image classification, or whether it is rather an academically interesting, but in reality irrelevant phenomenon.

We first consider the methods for creating adversarial examples that are covered in this work. The white-box attacks presented in this work require extensive knowledge on the DNN architecture and its parameters θ after training to compute the needed gradients. In the defined black-box setting, extensive access to the model is still needed, even if it can be treated as a black-box. If potential adversaries do not have the above capabilities, they cannot create adversarial examples using these methods. With the DNN being a crucial part of a system like an autonomously driving car, it can be assumed that there would be various security features, such as encryption, to prevent unauthorized external access. However, this only regards the process of creating an adversarial example.

The second part of the job would be to expose the DNN to the created adversarial example. Since the presented adversarial attacks all create adversarial examples as byte representations of images, adversaries would have to gain access to the DNN during

deployment. To stick with the application of autonomous driving, they would essentially have to hack the car while it is in traffic. At this point, it becomes clear that the threat from such adversarial examples is actually less than the threat from hacking the surrounding system. This is because by the time the attackers are able to do so, it is too late anyway; then, they do not have to take the detour via adversarial examples, but can directly manipulate the behavior of the system, e.g., the driving actions of the car.

As a result, the question arises whether there are adversarial perturbations that can be applied in the physical world, rather than at the virtual byte level, and that are sufficiently invariant to aspects such as camera-induced image distortion, lighting conditions, clarity of view, recording angle, recording distance, and so on.

In 2016, Kurakin, Goodfellow, and Bengio [49] conducted first experiments concerning physical adversarial examples. They initially created digital adversarial examples of natural images from the ImageNet dataset [62] using different white-box attacks, including FGSM and a variant of PGD. Subsequently, they printed these digital adversarial examples on paper and took pictures, whereas the recording conditions were intentionally not kept as constant as possible. Eventually, they classified the physical adversarial examples created in this way. They found out that while causing fewer misclassifications on digital adversarial examples than the stronger iterative attacks, the FGSM led to the highest rate of misclassification for the physical adversarial examples (between 35 % and 55 % classification accuracy, depending on the applied magnitude of perturbation ϵ). They also evaluated the accuracy for prints of natural validation images, being around 70 % and thus noticeably higher.

However, although the realization of usual adversarial perturbations being partly invariant to the camera transformation is quite fascinating, the impact on the real threat being posed by adversarial examples can be assumed to be rather moderate. Resorting once again to the application of autonomous driving, it would be conspicuous if someone started to cover traffic signs by slightly noisy paper prints. Even if they could be attached unnoticed, the difference to normal traffic signs would be clearly visible, e.g., the missing reflections.

Things look different when we consider the work of Eykholt et al. [63], which was published in 2018. They experimented with the GTSRB dataset and were able to create adversarial perturbations in the form of graffiti and small, black/white stickers that were relatively invariant to both different recording angles and distances. Although any modification of traffic signs is prohibited according to § 315b Strafgesetzbuch (StGB), this kind of physical adversarial perturbations is at first less conspicuous to humans, especially, if the traffic sign itself is still discernible. Hence, the existence of such physical adversarial perturbations would definitely require increased monitoring of traffic signs

in the era of autonomous driving.

Last, it is unclear in how far adversarial training can also increase the robustness to physical adversarial examples. This is to be investigated in some future work.

6 CONCLUSION

In this work, we conducted an empirical study regarding the robustness of DNN-based image classifiers against perturbations of the input data. Here, we particularly sought to gain further insights into the nature of adversarial examples. In the following, we summarize the main results.

First of all, this work points out that adversarial examples can only barely be created by random perturbations, but require very specific perturbations that are found by purposefully exploiting the deficiencies of the trained models. Next, the evaluations show that in the vast majority of cases, a strong white-box attack such as PGD can generate adversarial examples, even if they are constrained to a small ℓ_∞ distance from the corresponding natural image and are therefore nearly imperceptible to the human eye. With the Boundary Attack and operating in the black-box setting, on the other hand, it was more difficult to create adversarial examples with a comparatively low degree of perceptibility.

The results after adversarial training confirm that it increases the robustness against adversarial examples to a limited extent, coming at the price of higher training effort and lower accuracy w.r.t. natural image data. Here, the analysis of the training progress indicated that the insufficiency of adversarial training is generally a problem of substantial overfitting, of which the failure mode "catastrophic overfitting" is only the worst case. Furthermore, this work confirms that the recently published SSS algorithm is effective to mitigate the overfitting, subject to a moderate magnitude of perturbation. With higher magnitude of perturbation, however, the overfitting behavior turned out to be similar to FGSM-based adversarial training again.

Finally, the comparison of the two datasets CIFAR-10 and GTSRB revealed that the latter is inherently more robust against adversarial perturbations that the former. The results obtained were corroborated by a qualitative study of the loss landscapes that emerged for the trained DNNs.

A PROOFS

A.1. EUCLIDEAN PROJECTION ONTO A CLOSED ℓ_∞ BALL

Lemma 1 *Let $x, a, b \in \mathbb{R}$ be real numbers. Then, the following equivalence holds:*

$$|x - a| < |b - a| \iff (a < b \wedge 2a - b < x < b) \vee (a > b \wedge b < x < 2a - b)\,.$$

Proof: Proof by case analysis. Note that the cases with $a = b$ lead to a contradiction and thus do not allow any statement about x.

Case 1: $x > a \wedge a < b$:

$$x - a < b - a \iff x < b$$

Case 2: $x < a \wedge a < b$:

$$a - x < b - a \iff x > 2a - b$$

Case 3: $x > a \wedge b < a$:

$$x - a < a - b \iff x < 2a - b$$

Case 4: $x < a \wedge b < a$:

$$a - x < a - b \iff x > b$$

Case 5: $x = a \wedge a < b$:

$$0 < b - a \iff a < b$$

Case 6: $x = a \wedge b < a$:

$$0 < a - b \iff b < a$$

Case 7: $x > a \wedge a = b$:

$$x - a < 0 \iff x < a \quad \lightning$$

Case 8: $x < a \wedge a = b$:

$$a - x < 0 \iff x > a \quad \lightning$$

Theorem 1 *Let $\ell_\infty^{(n)} \subseteq \mathbb{R}^n$ be an n-dimensional metric ℓ_∞ space. Moreover, let $\overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x^{(0)}}, r)$ be the closed ℓ_∞ ball in $\ell_\infty^{(n)}$ that is centered in the point $\mathbf{x^{(0)}} \in \ell_\infty^{(n)}$ and has radius $r \in \mathbb{R}_{\geq 0}$. Then, the Euclidean projection of the point $\mathbf{x^*} \in \ell_\infty^{(n)}$ onto $\overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x^{(0)}}, r)$ is*

$$\Pi_{\overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x^{(0)}},r)}\left(\mathbf{x^*}\right) = \max\left(\mathbf{x^{(0)}} - r \cdot \mathbf{1}, \min\left(\mathbf{x^*}, \mathbf{x^{(0)}} + r \cdot \mathbf{1}\right)\right).$$

Proof: The proof is done by contradiction: Resorting to the definition of the Euclidean projection (eq. (2.14)), the Euclidean distance between the point $\mathbf{x^*}$ and its projection $\Pi_{\overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x^{(0)}},r)}\left(\mathbf{x^*}\right)$ is said to be minimal. In a first step, we assume that there is another point $\tilde{\mathbf{x}} \in \overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x^{(0)}}, r)$ s.t. the Euclidean difference between $\mathbf{x^*}$ and $\tilde{\mathbf{x}}$ is smaller than that between $\mathbf{x^*}$ and $\Pi_{\overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x^{(0)}},r)}\left(\mathbf{x^*}\right)$. Subsequently, we show that this assumption leads to a logical contradiction and therefore prove the original statement.

First of all, if we assume that there is a point $\tilde{\mathbf{x}}$ being closer to $\mathbf{x^*}$ according to the Euclidean metric, we can conclude that there must be at least one dimension in which it is closer.

$$\begin{gathered}
\exists \tilde{\mathbf{x}} \in \overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x^{(0)}}, r) : \sqrt{\sum_{i=1}^{n} \left|\tilde{x}_i - x_i^*\right|^2} < \sqrt{\sum_{i=1}^{n} \left|\max\left(x_i^{(0)} - r, \min\left(x_i^*, x_i^{(0)} + r\right)\right) - x_i^*\right|^2} \\
\Downarrow \\
\exists i \in \{1, 2, \ldots, n\} : \forall j \in \{1, 2, \ldots, n\} \setminus \{i\} : \tilde{x}_j = \max\left(x_j^{(0)} - r, \min\left(x_j^*, x_j^{(0)} + r\right)\right) \wedge \\
\left(x_i^{(0)} - r \leq \tilde{x}_i \leq x_i^{(0)} + r \rightarrow \left|\tilde{x}_i - x_i^*\right| < \left|\max\left(x_i^{(0)} - r, \min\left(x_i^*, x_i^{(0)} + r\right)\right) - x_i^*\right|\right)
\end{gathered} \tag{A.1}$$

At this point, we define the components $\tilde{x}_i$ and x_i^* as a function of the corresponding component $x_i^{(0)}$ of the closed ball's center point $\mathbf{x^{(0)}}$, respectively. To do so, we introduce two auxiliary variables $\tilde{r}, r^* \in \mathbb{R}$:

$$\tilde{x}_i := x_i^{(0)} + \tilde{r}, \tag{A.2}$$

$$x_i^* := x_i^{(0)} + r^*. \tag{A.3}$$

Substitution in eq. (A.1) yields

$$\begin{aligned}
&\exists i \in \{1,2,\dots,n\} : \forall j \in \{1,2,\dots,n\} \setminus \{i\} : \tilde{x}_j = \max\left(x_j^{(0)} - r, \min\left(x_j^*, x_j^{(0)} + r\right)\right) \wedge \\
&\qquad \left(-r \leq \tilde{r} \leq r \rightarrow |\tilde{r} - r^*| < \left|\max\left(x_i^{(0)} - r, \min\left(x_i^{(0)} + r^*, x_i^{(0)} + r\right)\right) - x_i^{(0)} - r^*\right|\right) \\
&\qquad\qquad \Downarrow \\
&\exists i \in \{1,2,\dots,n\} : \forall j \in \{1,2,\dots,n\} \setminus \{i\} : \tilde{x}_j = \max\left(x_j^{(0)} - r, \min\left(x_j^*, x_j^{(0)} + r\right)\right) \wedge \\
&\qquad\qquad (-r \leq \tilde{r} \leq r \rightarrow |\tilde{r} - r^*| < |\max(-r, \min(r^*, r)) - r^*|)
\end{aligned} \tag{A.4}$$

We now perform a case distinction w.r.t. r^* and show that all cases lead to contradiction. For simplification, we only consider the second term of the conjunction in eq. (A.4) – the first term is independent of r^* and can be assumed to be true, but a contradiction is already given when the second term evaluates to false and we therefore imply something false out of something true.

Case 1: $r^* > r$:

$$\begin{gathered}
-r \leq \tilde{r} \leq r \rightarrow |\tilde{r} - r^*| < |\max(-r, \min(r^*, r)) - r^*| = |r - r^*| \\
\Downarrow \text{(lemma 1)} \\
-r \leq \tilde{r} \leq r \rightarrow r < \tilde{r} < 2r^* - r \quad \lightning
\end{gathered}$$

Case 2: $-r \leq r^* \leq r$:

$$-r \leq \tilde{r} \leq r \rightarrow |\tilde{r} - r^*| < |\max(-r, \min(r^*, r)) - r^*| = 0 \quad \lightning$$

Case 3: $r^* < -r$:

$$\begin{gathered}
-r \leq \tilde{r} \leq r \rightarrow |\tilde{r} - r^*| < |\max(-r, \min(r^*, r)) - r^*| = |-r - r^*| \\
\Downarrow \text{(lemma 1)} \\
-r \leq \tilde{r} \leq r \rightarrow 2r^* + r < \tilde{r} < -r \quad \lightning
\end{gathered}$$

Q. E. D.

A.2. FAST GRADIENT SIGN METHOD BEING CONSTRAINED BY THE ℓ_∞ METRIC

Lemma 2 *Let $a, b \in \mathbb{R}$ be real numbers and let $c \in \mathbb{R}_{\geq 0}$ be a non-negative real number. Then, the following inequality holds:*

$$|\max(-a, \min(c \operatorname{sign} b, 1-a))| \leq c.$$

Proof: Proof by case analysis.

Case 1: $c \operatorname{sign} b > 1 - a$:

$$|\max(-a, \min(c \operatorname{sign} b, 1-a))| = |\max(-a, 1-a)| = 1 - a < c \operatorname{sign} b \leq c$$

Case 2: $-a \leq c \operatorname{sign} b \leq 1 - a$:

$$|\max(-a, \min(c \operatorname{sign} b, 1-a))| = |\max(-a, c \operatorname{sign} b)| = |c \operatorname{sign} b| = c\,|\operatorname{sign} b| \leq c$$

Case 3: $c \operatorname{sign} b < -a$:

$$|\max(-a, \min(c \operatorname{sign} b, 1-a))| = |\max(-a, c \operatorname{sign} b)| = \|-a\| = a < -c \operatorname{sign} b \leq c$$

Theorem 2 *Let*

$$\mathbf{x}' = \Pi_{\mathcal{X}}\left(\mathbf{x} + \epsilon \operatorname{sign} \nabla_{\mathbf{x}} J(\hat{f}(\mathbf{x}), \mathbf{y})\right)$$

be the FGSM-perturbed version of the natural image $\mathbf{x} \in \mathcal{X}$ with the magnitude of perturbation $\epsilon \in \mathbb{R}_{\geq 0}$, according to eq. (3.13). Then, $\mathbf{x}'$ is part of the closed ℓ_∞ ball $\overline{\mathcal{B}}_{\mathcal{X}}(\mathbf{x}, \epsilon)$, i.e., $\mathbf{x}' \in \overline{\mathcal{B}}_{\mathcal{X}}(\mathbf{x}, \epsilon)$.

Proof: We start by taking lemma 2 for granted and set $a = x_i$, $b = \frac{\partial J(\hat{f}(\mathbf{x}), \mathbf{y})}{\partial x_i}$ and $c = \epsilon$ for all $i \in \{1, 2, \ldots, n\}$:

$$\forall i \in \{1, 2, \ldots, n\} : \left|\max\left(-x_i, \min\left(\epsilon \operatorname{sign} \frac{\partial J(\hat{f}(\mathbf{x}), \mathbf{y})}{\partial x_i}, 1 - x_i\right)\right)\right| \leq \epsilon. \tag{A.5}$$

From this point, we can simply transform the inequality's left-hand side and finally obtain the condition for $\mathbf{x}'$ being part of the closed ℓ_∞ ball $\overline{\mathcal{B}}_{\mathcal{X}}(\mathbf{x}, \epsilon)$:

$$\forall i \in \{1,2,\ldots,n\} : \left|\max\left(-x_i, \min\left(\epsilon \operatorname{sign} \frac{\partial J(\hat{f}(\mathbf{x}),\mathbf{y})}{\partial x_i}, 1-x_i\right)\right)\right| \leq \epsilon$$

$$\Updownarrow$$

$$\forall i \in \{1,2,\ldots,n\} : \left|\max\left(0, \min\left(x_i + \epsilon \operatorname{sign} \frac{\partial J(\hat{f}(\mathbf{x}),\mathbf{y})}{\partial x_i}, 1\right)\right) - x_i\right| \leq \epsilon$$

$$\Updownarrow (2.18)$$

$$\forall i \in \{1,2,\ldots,n\} : \left|\Pi_{\mathcal{X}}\left(x_i + \epsilon \operatorname{sign} \frac{\partial J(\hat{f}(\mathbf{x}),\mathbf{y})}{\partial x_i}\right) - x_i\right| \leq \epsilon \tag{A.6}$$

$$\Updownarrow$$

$$\forall i \in \{1,2,\ldots,n\} : |x'_i - x_i| \leq \epsilon$$

$$\Updownarrow$$

$$\max_{i\in\{1,2,\ldots,n\}} |x_i| \overset{(2.15)}{=} \|\mathbf{x}' - \mathbf{x}\|_\infty = d_\infty\left(\mathbf{x}', \mathbf{x}\right) \leq \epsilon$$

$$\Updownarrow$$

$$\mathbf{x}' \in \overline{\mathcal{B}}_{\mathcal{X}}(\mathbf{x}, \epsilon)$$

Q. E. D.

A.3. EUCLIDEAN PROJECTION ONTO THE INTERSECTION OF TWO CLOSED ℓ_∞ BALLS

Theorem 3 *Let $\ell_\infty^{(n)} \subseteq \mathbb{R}^n$ be an n-dimensional metric ℓ_∞ space. Moreover, let $\mathcal{X}_1 = \overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x}^{(0),(1)}, r_1)$ and $\mathcal{X}_2 = \overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x}^{(0),(2)}, r_2)$ be the closed ℓ_∞ balls in $\ell_\infty^{(n)}$ that are centered in the points $\mathbf{x}^{(0),(1)}, \mathbf{x}^{(0),(2)} \in \ell_\infty^{(n)}$ and have radii $r_1, r_2 \in \mathbb{R}_{\geq 0}$, respectively. Then, if $\mathcal{X}_1$ and $\mathcal{X}_2$ are not disjoint, the successive Euclidean projection of the point $\mathbf{x}^* \in \ell_\infty^{(n)}$ first onto $\mathcal{X}_2$, then onto $\mathcal{X}_1$ equals the reversed projection order:*

$$\mathcal{X}_1 \cap \mathcal{X}_2 \neq \emptyset \rightarrow \Pi_{\mathcal{X}_1}\left(\Pi_{\mathcal{X}_2}\left(\mathbf{x}^*\right)\right) = \Pi_{\mathcal{X}_2}\left(\Pi_{\mathcal{X}_1}\left(\mathbf{x}^*\right)\right).$$

Proof: Applying theorem 1 and reformulating the precondition (to be not disjoint, at least one boundary of one ℓ_∞ ball has to be shared by both ℓ_∞ balls), the following is to be proven:

$$
\begin{aligned}
&\forall i \in \{1,2,\dots,n\}: \\
&\quad x_i^{(0),(1)} - r_1 \le x_i^{(0),(2)} - r_2 \le x_i^{(0),(1)} + r_1 \vee x_i^{(0),(2)} - r_2 \le x_i^{(0),(1)} - r_1 \le x_i^{(0),(2)} + r_2 \\
\rightarrow \quad & \max\left(x_i^{(0),(1)} - r_1, \min\left(\max\left(x_i^{(0),(2)} - r_2, \min\left(x_i^*, x_i^{(0),(2)} + r_2\right)\right), x_i^{(0),(1)} + r_1\right)\right) \\
= \; & \max\left(x_i^{(0),(2)} - r_2, \min\left(\max\left(x_i^{(0),(1)} - r_1, \min\left(x_i^*, x_i^{(0),(1)} + r_1\right)\right), x_i^{(0),(2)} + r_2\right)\right).
\end{aligned} \tag{A.7}
$$

The proof is done by case analysis w.r.t. $\mathbf{x}^*$. For the sake of brevity, we omit the consecutive quantifier $\forall i \in \{1,2,\dots,n\}$ in the following, but we point out that this is what is meant by the index i. In addition, we index the further above preconditions as follows, being used as a short notation subsequently:

$$(I.) \;:=\; x_i^{(0),(1)} - r_1 \le x_i^{(0),(2)} - r_2 \le x_i^{(0),(1)} + r_1 \tag{A.8}$$

$$(II.) \;:=\; x_i^{(0),(2)} - r_2 \le x_i^{(0),(1)} - r_1 \le x_i^{(0),(2)} + r_2 \tag{A.9}$$

Last, note that the cases $x_i^* > x_i^{(0),(1)} + r_1 \wedge x_i^* < x_i^{(0),(2)} - r_2$ as well as $x_i^* < x_i^{(0),(1)} - r_1 \wedge x_i^* > x_i^{(0),(2)} + r_2$ contradict with the precondition $(I.) \vee (II.)$.

Case 1: $((I.) \vee (II.)) \;\wedge\; x_i^* > x_i^{(0),(1)} + r_1 \;\wedge\; x_i^* > x_i^{(0),(2)} + r_2$:

$$
\begin{aligned}
& \max\left(x_i^{(0),(1)} - r_1, \min\left(\max\left(x_i^{(0),(2)} - r_2, \min\left(x_i^*, x_i^{(0),(2)} + r_2\right)\right), x_i^{(0),(1)} + r_1\right)\right) \\
= \; & \max\left(x_i^{(0),(1)} - r_1, \min\left(x_i^{(0),(1)} + r_1, x_i^{(0),(2)} + r_2\right)\right) \\
= \; & \min\left(x_i^{(0),(1)} + r_1, x_i^{(0),(2)} + r_2\right) \\
= \; & \max\left(x_i^{(0),(2)} - r_2, \min\left(x_i^{(0),(1)} + r_1, x_i^{(0),(2)} + r_2\right)\right) \\
= \; & \max\left(x_i^{(0),(2)} - r_2, \min\left(\max\left(x_i^{(0),(1)} - r_1, \min\left(x_i^*, x_i^{(0),(1)} + r_1\right)\right), x_i^{(0),(2)} + r_2\right)\right)
\end{aligned}
$$

Case 2: $((I.) \vee (II.)) \;\wedge\; x_i^* > x_i^{(0),(1)} + r_1 \;\wedge\; x_i^{(0),(2)} - r_2 \le x_i^* \le x_i^{(0),(2)} + r_2$:

$$
\begin{aligned}
& \max\left(x_i^{(0),(1)} - r_1, \min\left(\max\left(x_i^{(0),(2)} - r_2, \min\left(x_i^*, x_i^{(0),(2)} + r_2\right)\right), x_i^{(0),(1)} + r_1\right)\right) \\
= \; & x_i^{(0),(1)} + r_1 \\
= \; & \min\left(x_i^{(0),(1)} + r_1, x_i^{(0),(2)} + r_2\right) \\
= \; & \max\left(x_i^{(0),(2)} - r_2, \min\left(x_i^{(0),(1)} + r_1, x_i^{(0),(2)} + r_2\right)\right) \\
= \; & \max\left(x_i^{(0),(2)} - r_2, \min\left(\max\left(x_i^{(0),(1)} - r_1, \min\left(x_i^*, x_i^{(0),(1)} + r_1\right)\right), x_i^{(0),(2)} + r_2\right)\right)
\end{aligned}
$$

Case 3: $((I.) \vee (II.)) \;\wedge\; x_i^{(0),(1)} - r_1 \leq x_i^* \leq x_i^{(0),(1)} + r_1 \;\wedge\; x_i^* > x_i^{(0),(2)} + r_2$:

$$\begin{aligned}
&\max\left(x_i^{(0),(1)} - r_1, \min\left(\max\left(x_i^{(0),(2)} - r_2, \min\left(x_i^*, x_i^{(0),(2)} + r_2\right)\right), x_i^{(0),(1)} + r_1\right)\right) \\
&= \max\left(x_i^{(0),(1)} - r_1, \min\left(x_i^{(0),(1)} + r_1, x_i^{(0),(2)} + r_2\right)\right) \\
&= \min\left(x_i^{(0),(1)} + r_1, x_i^{(0),(2)} + r_2\right) \\
&= x_i^{(0),(2)} + r_2 \\
&= \max\left(x_i^{(0),(2)} - r_2, \min\left(x_i^*, x_i^{(0),(2)} + r_2\right)\right) \\
&= \max\left(x_i^{(0),(2)} - r_2, \min\left(\max\left(x_i^{(0),(1)} - r_1, \min\left(x_i^*, x_i^{(0),(1)} + r_1\right)\right), x_i^{(0),(2)} + r_2\right)\right)
\end{aligned}$$

Case 4: $((I.) \vee (II.)) \;\wedge\; x_i^{(0),(1)} - r_1 \leq x_i^* \leq x_i^{(0),(1)} + r_1 \;\wedge\; x_i^{(0),(2)} - r_2 \leq x_i^* \leq x_i^{(0),(2)} + r_2$:

$$\begin{aligned}
&\max\left(x_i^{(0),(1)} - r_1, \min\left(\max\left(x_i^{(0),(2)} - r_2, \min\left(x_i^*, x_i^{(0),(2)} + r_2\right)\right), x_i^{(0),(1)} + r_1\right)\right) \\
&= x_i^* \\
&= \max\left(x_i^{(0),(2)} - r_2, \min\left(\max\left(x_i^{(0),(1)} - r_1, \min\left(x_i^*, x_i^{(0),(1)} + r_1\right)\right), x_i^{(0),(2)} + r_2\right)\right)
\end{aligned}$$

Case 5: $(I.) \;\wedge\; x_i^{(0),(1)} - r_1 \leq x_i^* \leq x_i^{(0),(1)} + r_1 \;\wedge\; x_i^* < x_i^{(0),(2)} - r_2$:

$$\begin{aligned}
&\max\left(x_i^{(0),(1)} - r_1, \min\left(\max\left(x_i^{(0),(2)} - r_2, \min\left(x_i^*, x_i^{(0),(2)} + r_2\right)\right), x_i^{(0),(1)} + r_1\right)\right) \\
&= \max\left(x_i^{(0),(1)} - r_1, \min\left(x_i^{(0),(1)} + r_1, x_i^{(0),(2)} - r_2\right)\right) \\
&= \min\left(x_i^{(0),(1)} + r_1, x_i^{(0),(2)} - r_2\right) \\
&= x_i^{(0),(2)} - r_2 \\
&= \max\left(x_i^{(0),(2)} - r_2, \min\left(x_i^*, x_i^{(0),(2)} + r_2\right)\right) \\
&= \max\left(x_i^{(0),(2)} - r_2, \min\left(\max\left(x_i^{(0),(1)} - r_1, \min\left(x_i^*, x_i^{(0),(1)} + r_1\right)\right), x_i^{(0),(2)} + r_2\right)\right)
\end{aligned}$$

Case 6: $(II.) \;\wedge\; x_i^* < x_i^{(0),(1)} - r_1 \;\wedge\; x_i^{(0),(2)} - r_2 \leq x_i^* \leq x_i^{(0),(2)} + r_2$:

$$\begin{aligned}
&\max\left(x_i^{(0),(1)} - r_1, \min\left(\max\left(x_i^{(0),(2)} - r_2, \min\left(x_i^*, x_i^{(0),(2)} + r_2\right)\right), x_i^{(0),(1)} + r_1\right)\right) \\
&= x_i^{(0),(1)} - r_1 \\
&= \min\left(x_i^{(0),(1)} - r_1, x_i^{(0),(2)} + r_2\right) \\
&= \max\left(x_i^{(0),(2)} - r_2, \min\left(x_i^{(0),(1)} - r_1, x_i^{(0),(2)} + r_2\right)\right) \\
&= \max\left(x_i^{(0),(2)} - r_2, \min\left(\max\left(x_i^{(0),(1)} - r_1, \min\left(x_i^*, x_i^{(0),(1)} + r_1\right)\right), x_i^{(0),(2)} + r_2\right)\right)
\end{aligned}$$

Case 7: $((I.) \vee (II.)) \quad \wedge \quad x_i^* < x_i^{(0),(1)} - r_1 \quad \wedge \quad x_i^* < x_i^{(0),(2)} - r_2$:

$$\begin{aligned}
&\max\left(x_i^{(0),(1)} - r_1, \min\left(\max\left(x_i^{(0),(2)} - r_2, \min\left(x_i^*, x_i^{(0),(2)} + r_2\right)\right), x_i^{(0),(1)} + r_1\right)\right) \\
&= \max\left(x_i^{(0),(1)} - r_1, \min\left(x_i^{(0),(1)} + r_1, x_i^{(0),(2)} - r_2\right)\right) \\
&= \max\left(x_i^{(0),(1)} - r_1, x_i^{(0),(2)} - r_2\right) \\
&= \max\left(x_i^{(0),(2)} - r_2, \min\left(x_i^{(0),(1)} - r_1, x_i^{(0),(2)} + r_2\right)\right) \\
&= \max\left(x_i^{(0),(2)} - r_2, \min\left(\max\left(x_i^{(0),(1)} - r_1, \min\left(x_i^*, x_i^{(0),(1)} + r_1\right)\right), x_i^{(0),(2)} + r_2\right)\right)
\end{aligned}$$

Q. E. D.

Theorem 4 *Let $\ell_\infty^{(n)} \subseteq \mathbb{R}^n$ be an n-dimensional metric ℓ_∞ space. Moreover, let $\mathcal{X}_1 = \overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x}^{(0),(1)}, r_1)$ and $\mathcal{X}_2 = \overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x}^{(0),(2)}, r_2)$ be the closed ℓ_∞ balls in $\ell_\infty^{(n)}$ that are centered in the points $\mathbf{x}^{(0),(1)}, \mathbf{x}^{(0),(2)} \in \ell_\infty^{(n)}$ and have radii $r_1, r_2 \in \mathbb{R}_{\geq 0}$, respectively. Then, if $\mathcal{X}_1$ and $\mathcal{X}_2$ are not disjoint, the Euclidean projection of the point $\mathbf{x}^* \in \ell_\infty^{(n)}$ onto the intersection of both closed ℓ_∞ balls $\mathcal{X}_{12} = \overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x}^{(0),(1)}, r_1) \cap \overline{\mathcal{B}}_{\ell_\infty^{(n)}}(\mathbf{x}^{(0),(2)}, r_2)$ is equivalent to the Euclidean projection first onto $\mathcal{X}_1$, then onto $\mathcal{X}_2$:*

$$\mathcal{X}_1 \cap \mathcal{X}_2 \neq \emptyset \rightarrow \Pi_{\mathcal{X}_{12}}\left(\mathbf{x}^*\right) = \Pi_{\mathcal{X}_2}\left(\Pi_{\mathcal{X}_1}\left(\mathbf{x}^*\right)\right).$$

Proof: The proof is structured similarly to those of theorems 1 and 3.

First, as in the proof of theorem 1, this proof is done by contradiction, showing that there cannot be a valid $\tilde{\mathbf{x}} \in \mathcal{X}_{12}$ s.t. the Euclidean difference between $\mathbf{x}^*$ and $\tilde{\mathbf{x}}$ is smaller than that between $\mathbf{x}^*$ and $\Pi_{\mathcal{X}_{12}}(\mathbf{x}^*)$. Doing so, we initially assume as well that if there is a point $\tilde{\mathbf{x}}$ being closer to $\mathbf{x}^*$ according to the Euclidean metric, we can conclude that there must be at least one dimension in which it is closer.

Second, as in the proof of theorem 3, we initially apply theorem 1 and reformulate the preconditions. Besides, we also use the following short notations of preconditions that were introduced there:

$$(I.) \;:=\; \forall k \in \{1, 2, \ldots, n\} : x_k^{(0),(1)} - r_1 \leq x_k^{(0),(2)} - r_2 \leq x_k^{(0),(1)} + r_1 \tag{A.10}$$

$$(II.) \;:=\; \forall k \in \{1, 2, \ldots, n\} : x_k^{(0),(2)} - r_2 \leq x_k^{(0),(1)} - r_1 \leq x_k^{(0),(2)} + r_2 \tag{A.11}$$

Furthermore, we abbreviate the validity preconditions w.r.t. $\tilde{x}_i$, where i denotes the dimension in which $\tilde{\mathbf{x}}$ is closer to $\mathbf{x}^*$, by

$$(III.) \;:=\; x_i^{(0),(1)} - r_1 \leq \tilde{x}_i \leq x_i^{(0),(1)} + r_1, \tag{A.12}$$

$$(IV.) \;:=\; x_i^{(0),(2)} - r_2 \leq \tilde{x}_i \leq x_i^{(0),(2)} + r_2. \tag{A.13}$$

Thus, the following is to be proven:

$$\begin{aligned}
&\exists i \in \{1,2,\ldots,n\} : \forall j \in \{1,2,\ldots,n\} \setminus \{i\} : (I.) \vee (II.) \rightarrow \\
&\quad \tilde{x}_j = \max\left(x_j^{(0),(2)} - r_2, \min\left(\max\left(x_j^{(0),(1)} - r_1, \min\left(x_j^*, x_j^{(0),(1)} + r_1\right)\right), x_j^{(0),(2)} + r_2\right)\right) \wedge \\
&\qquad \left((III.) \wedge (IV.) \rightarrow \left|\tilde{x}_i - x_i^*\right| < \right. \\
&\quad \left.\left|\max\left(x_i^{(0),(2)} - r_2, \min\left(\max\left(x_i^{(0),(1)} - r_1, \min\left(x_i^*, x_i^{(0),(1)} + r_1\right)\right), x_i^{(0),(2)} + r_2\right)\right) - x_i^*\right|\right).
\end{aligned} \tag{A.14}$$

We now perform a case distinction w.r.t. x_i^* similar to the proof of theorem 3 and subsequently show that every case leads to contradiction as in the proof of theorem 1. Note here that as for the former, the cases $x_i^* > x_i^{(0),(1)} + r_1 \wedge x_i^* < x_i^{(0),(2)} - r_2$ as well as $x_i^* < x_i^{(0),(1)} - r_1 \wedge x_i^* > x_i^{(0),(2)} + r_2$ contradict with the precondition $(I.) \vee (II.)$ and are therefore not considered. Moreover, as for the latter, we only consider the second term of the outer conjunction in eq. (A.14), since the first one is irrelevant in the case distinction w.r.t. x_i^*. Last, for the sake of brevity, we omit the consecutive quantifier $\exists i \in \{1,2,\ldots,n\}$ in the following again, but we point out that this is what is meant by the index i.

Case 1: $((I.) \vee (II.)) \;\wedge\; x_i^* > x_i^{(0),(1)} + r_1 \;\wedge\; x_i^* > x_i^{(0),(2)} + r_2$:

$$\begin{aligned}
&(III.) \;\wedge\; (IV.) \rightarrow \left|\tilde{x}_i - x_i^*\right| \\
&< \left|\max\left(x_i^{(0),(2)} - r_2, \min\left(\max\left(x_i^{(0),(1)} - r_1, \min\left(x_i^*, x_i^{(0),(1)} + r_1\right)\right), x_i^{(0),(2)} + r_2\right)\right) - x_i^*\right| \\
&= \left|\max\left(x_i^{(0),(2)} - r_2, \min\left(x_i^{(0),(1)} + r_1, x_i^{(0),(2)} + r_2\right)\right) - x_i^*\right| \\
&= \left|\min\left(x_i^{(0),(1)} + r_1, x_i^{(0),(2)} + r_2\right) - x_i^*\right|
\end{aligned}$$

$$\Downarrow \text{(lemma 1)}$$

$$\begin{aligned}
&(III.) \;\wedge\; (IV.) \\
&\rightarrow \min\left(x_i^{(0),(1)} + r_1, x_i^{(0),(2)} + r_2\right) < \tilde{x}_i < 2x_i^* - \min\left(x_i^{(0),(1)} + r_1, x_i^{(0),(2)} + r_2\right) \quad \lightning
\end{aligned}$$

Case 2: $((I.) \vee (II.)) \quad \wedge \quad x_i^* > x_i^{(0),(1)} + r_1 \quad \wedge \quad x_i^{(0),(2)} - r_2 \leq x_i^* \leq x_i^{(0),(2)} + r_2$:

$$
\begin{aligned}
&(III.) \ \wedge \ (IV.) \rightarrow \left|\tilde{x}_i - x_i^*\right| \\
&< \left|\max\left(x_i^{(0),(2)} - r_2, \min\left(\max\left(x_i^{(0),(1)} - r_1, \min\left(x_i^*, x_i^{(0),(1)} + r_1\right)\right), x_i^{(0),(2)} + r_2\right)\right) - x_i^*\right| \\
&= \left|\max\left(x_i^{(0),(2)} - r_2, \min\left(x_i^{(0),(1)} + r_1, x_i^{(0),(2)} + r_2\right)\right) - x_i^*\right| \\
&= \left|\max\left(x_i^{(0),(1)} + r_1, x_i^{(0),(2)} - r_2\right) - x_i^*\right| \\
&= \left|x_i^{(0),(1)} + r_1 - x_i^*\right|
\end{aligned}
$$

$$\Downarrow \text{(lemma 1)}$$

$$(III.) \wedge (IV.) \rightarrow x_i^{(0),(1)} + r_1 < \tilde{x}_i < 2x_i^* - x_i^{(0),(1)} - r_1 \quad \lightning$$

Case 3: $((I.) \vee (II.)) \quad \wedge \quad x_i^{(0),(1)} - r_1 \leq x_i^* \leq x_i^{(0),(1)} + r_1 \quad \wedge \quad x_i^* > x_i^{(0),(2)} + r_2$:

$$
\begin{aligned}
&(III.) \ \wedge \ (IV.) \rightarrow \left|\tilde{x}_i - x_i^*\right| \\
&< \left|\max\left(x_i^{(0),(2)} - r_2, \min\left(\max\left(x_i^{(0),(1)} - r_1, \min\left(x_i^*, x_i^{(0),(1)} + r_1\right)\right), x_i^{(0),(2)} + r_2\right)\right) - x_i^*\right| \\
&= \left|\max\left(x_i^{(0),(2)} - r_2, \min\left(x_i^*, x_i^{(0),(2)} + r_2\right)\right) - x_i^*\right| \\
&= \left|x_i^{(0),(2)} + r_2 - x_i^*\right|
\end{aligned}
$$

$$\Downarrow \text{(lemma 1)}$$

$$(III.) \wedge (IV.) \rightarrow x_i^{(0),(2)} + r_2 < \tilde{x}_i < 2x_i^* - x_i^{(0),(2)} - r_2 \quad \lightning$$

Case 4: $((I.) \vee (II.)) \quad \wedge \quad x_i^{(0),(1)} - r_1 \leq x_i^* \leq x_i^{(0),(1)} + r_1 \quad \wedge \quad x_i^{(0),(2)} - r_2 \leq x_i^* \leq x_i^{(0),(2)} + r_2$:

$$
\begin{aligned}
&(III.) \ \wedge \ (IV.) \rightarrow \left|\tilde{x}_i - x_i^*\right| \\
&< \left|\max\left(x_i^{(0),(2)} - r_2, \min\left(\max\left(x_i^{(0),(1)} - r_1, \min\left(x_i^*, x_i^{(0),(1)} + r_1\right)\right), x_i^{(0),(2)} + r_2\right)\right) - x_i^*\right| \\
&= \left|x_i^* - x_i^*\right| \\
&= 0 \qquad \lightning
\end{aligned}
$$

Case 5: $(I.) \quad \wedge \quad x_i^{(0),(1)} - r_1 \leq x_i^* \leq x_i^{(0),(1)} + r_1 \quad \wedge \quad x_i^* < x_i^{(0),(2)} - r_2$:

$$
\begin{aligned}
&(III.) \ \wedge \ (IV.) \rightarrow \left|\tilde{x}_i - x_i^*\right| \\
&< \left|\max\left(x_i^{(0),(2)} - r_2, \min\left(\max\left(x_i^{(0),(1)} - r_1, \min\left(x_i^*, x_i^{(0),(1)} + r_1\right)\right), x_i^{(0),(2)} + r_2\right)\right) - x_i^*\right| \\
&= \left|\max\left(x_i^{(0),(2)} - r_2, \min\left(x_i^*, x_i^{(0),(2)} + r_2\right)\right) - x_i^*\right| \\
&= \left|x_i^{(0),(2)} - r_2 - x_i^*\right|
\end{aligned}
$$

$$\Downarrow \text{ (lemma 1)}$$

$$(III.) \wedge (IV.) \rightarrow 2x_i^* - x_i^{(0),(2)} - r_2 < \tilde{x}_i < x_i^{(0),(2)} - r_2 \quad \lightning$$

Case 6: $(II.) \quad \wedge \quad x_i^* < x_i^{(0),(1)} - r_1 \quad \wedge \quad x_i^{(0),(2)} - r_2 \leq x_i^* \leq x_i^{(0),(2)} + r_2$:

$$
\begin{aligned}
&(III.) \ \wedge \ (IV.) \rightarrow \left|\tilde{x}_i - x_i^*\right| \\
&< \left|\max\left(x_i^{(0),(2)} - r_2, \min\left(\max\left(x_i^{(0),(1)} - r_1, \min\left(x_i^*, x_i^{(0),(1)} + r_1\right)\right), x_i^{(0),(2)} + r_2\right)\right) - x_i^*\right| \\
&= \left|\max\left(x_i^{(0),(2)} - r_2, \min\left(x_i^{(0),(1)} - r_1, x_i^{(0),(2)} + r_2\right)\right) - x_i^*\right| \\
&= \left|\max\left(x_i^{(0),(1)} - r_1, x_i^{(0),(2)} - r_2\right) - x_i^*\right| \\
&= \left|x_i^{(0),(1)} - r_1 - x_i^*\right|
\end{aligned}
$$

$$\Downarrow \text{ (lemma 1)}$$

$$(III.) \wedge (IV.) \rightarrow 2x_i^* - x_i^{(0),(1)} - r_1 < \tilde{x}_i < x_i^{(0),(1)} - r_1 \quad \lightning$$

Case 7: $((I.) \vee (II.)) \quad \wedge \quad x_i^* < x_i^{(0),(1)} - r_1 \quad \wedge \quad x_i^* < x_i^{(0),(2)} - r_2$:

$$
\begin{aligned}
&(III.) \ \wedge \ (IV.) \rightarrow \left|\tilde{x}_i - x_i^*\right| \\
&< \left|\max\left(x_i^{(0),(2)} - r_2, \min\left(\max\left(x_i^{(0),(1)} - r_1, \min\left(x_i^*, x_i^{(0),(1)} + r_1\right)\right), x_i^{(0),(2)} + r_2\right)\right) - x_i^*\right| \\
&= \left|\max\left(x_i^{(0),(2)} - r_2, \min\left(x_i^{(0),(1)} - r_1, x_i^{(0),(2)} + r_2\right)\right) - x_i^*\right| \\
&= \left|\max\left(x_i^{(0),(1)} - r_1, x_i^{(0),(2)} - r_2\right) - x_i^*\right|
\end{aligned}
$$

$$\Downarrow \text{ (lemma 1)}$$

$$
\begin{aligned}
&(III.) \ \wedge \ (IV.) \\
&\rightarrow 2x_i^* - \max\left(x_i^{(0),(1)} - r_1, x_i^{(0),(2)} - r_2\right) < \tilde{x}_i < \max\left(x_i^{(0),(1)} - r_1, x_i^{(0),(2)} - r_2\right) \ \lightning
\end{aligned}
$$

Q. E. D.

B FURTHER ADVERSARIAL EXAMPLES AFTER STANDARD TRAINING

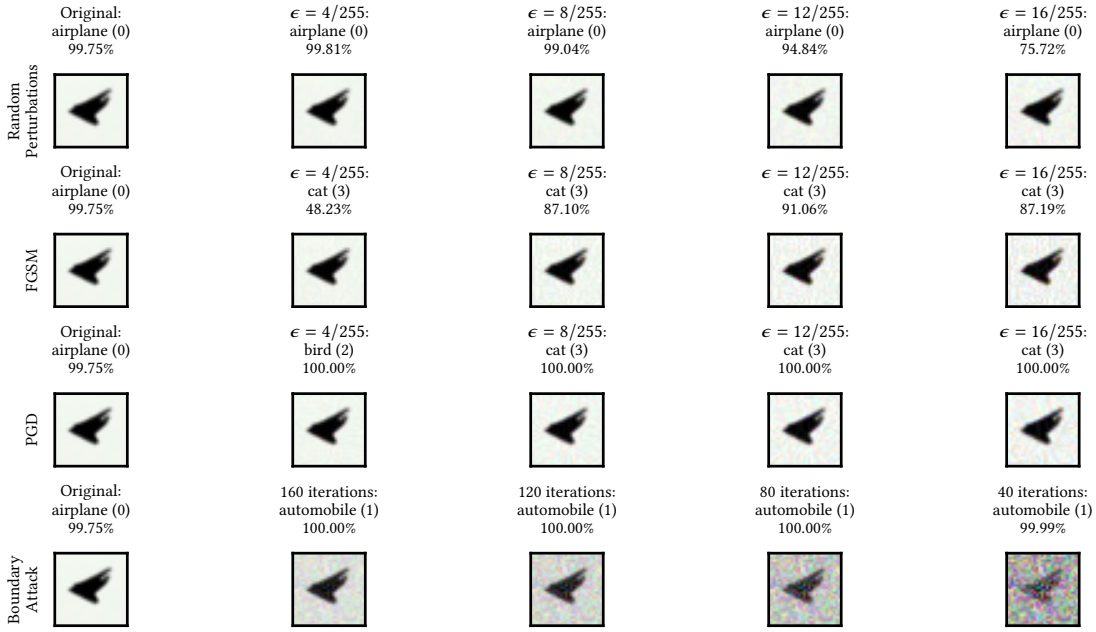
In this section, we present further adversarial examples that were created using the attacking threat models from section 3.6.2 on standardly trained DNNs.

CIFAR-10

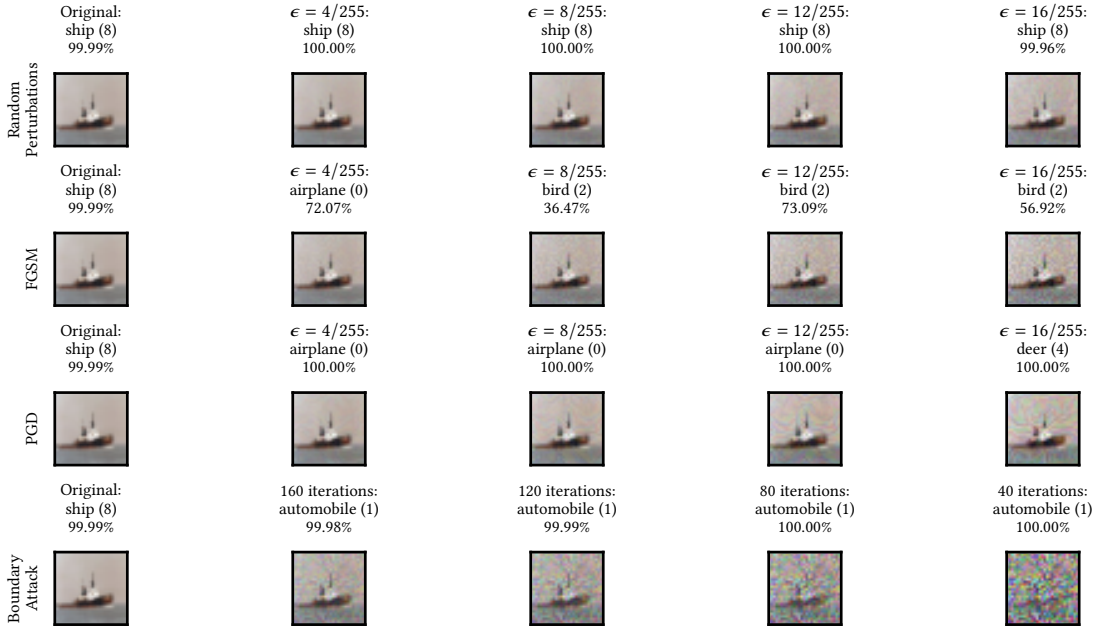
Figure B.1 shows additional adversarial examples that were generated on the PreAct-ResNet-50 being trained standardly on the CIFAR-10 dataset. The images depict an airplane and a ship, respectively.

GTSRB

Figure B.2 shows further adversarial examples being created on the PreAct-ResNet-50 that was trained standardly on the GTSRB dataset. The images depict a “Traffic lights” sign and “Speed limit 80” sign, respectively.

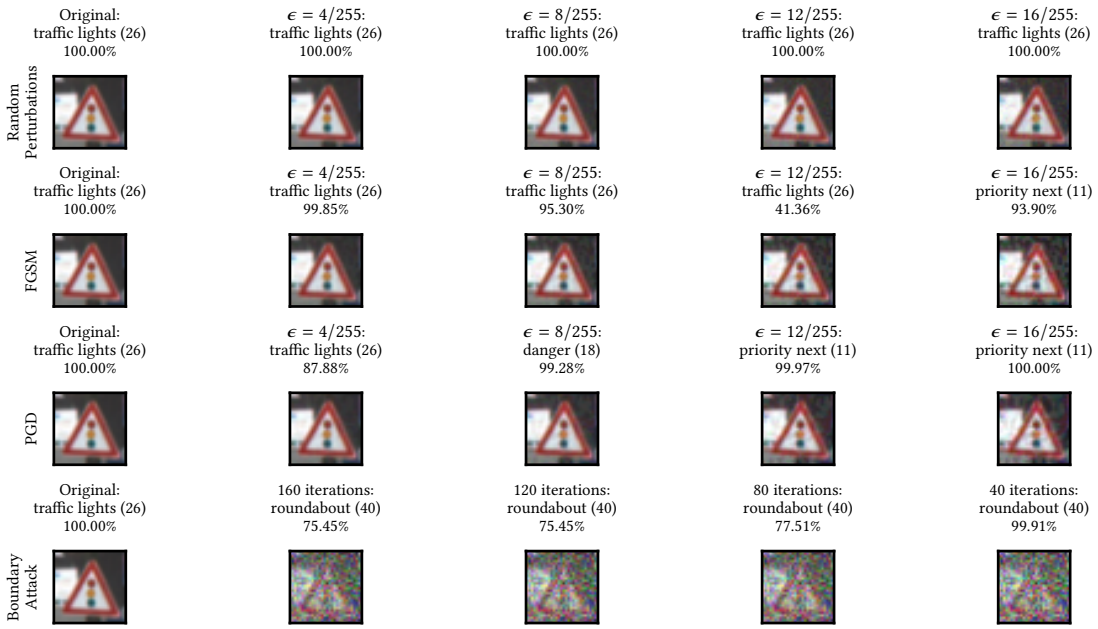


(a) Airplane

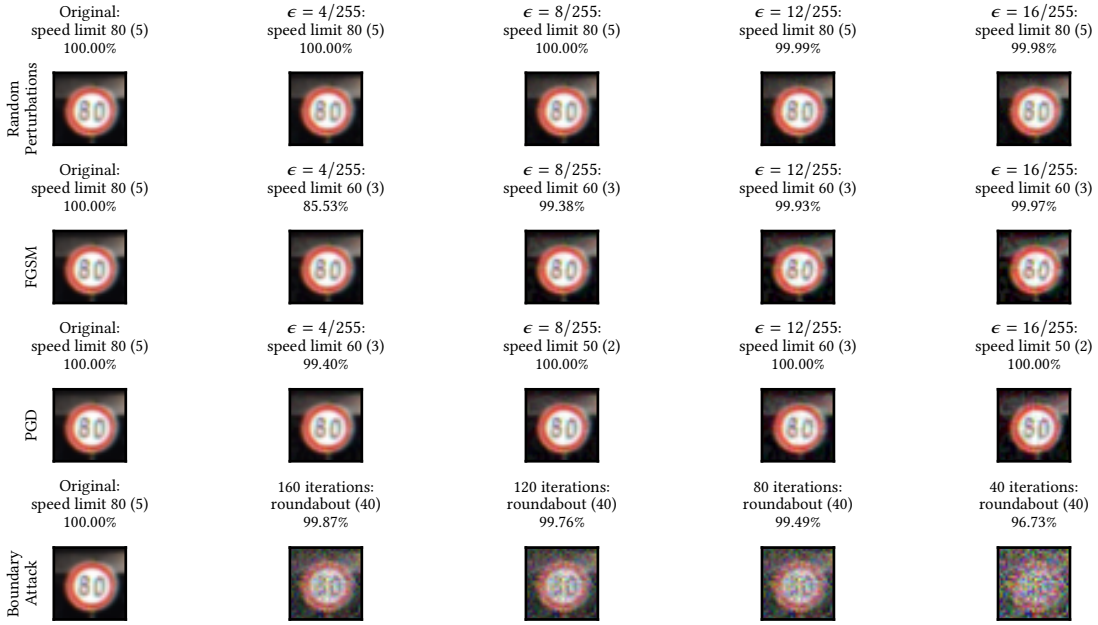


(b) Ship

Figure B.1.: Adversarial examples of CIFAR-10, created on standardly trained PreAct-ResNet-50 (Airplane, Ship). Each row shows perturbed versions of a natural image, created with one of the adversarial attacks: random perturbations, FGSM, PGD and Boundary Attack. The original image is shown in the left column. Remaining columns show perturbed versions with maximum ℓ_∞ distance $\epsilon = 4/255, 8/255, 12/255, 16/255$ (from left to right, for random perturbations, FGSM and PGD) and after 40, 80, 120, 160 iterations of Boundary Attack (from right to left), respectively. Classification information is provided above the corresponding image.



(a) "Traffic lights" sign



(b) "Speed limit 80" sign

Figure B.2.: Adversarial examples of GTSRB, created on standardly trained PreAct-ResNet-50 (Traffic Lights, Speed Limit 80). Each row shows perturbed versions of a natural image, created with one of the adversarial attacks: random perturbations, FGSM, PGD and Boundary Attack. The original image is shown in the left column. Remaining columns show perturbed versions with maximum ℓ_∞ distance $\epsilon = 4/255, 8/255, 12/255, 16/255$ (from left to right, for random perturbations, FGSM and PGD) and after 40, 80, 120, 160 iterations of Boundary Attack (from right to left), respectively. Classification information is provided above the corresponding image.

C FURTHER ADVERSARIAL EXAMPLES AFTER ADVERSARIAL TRAINING

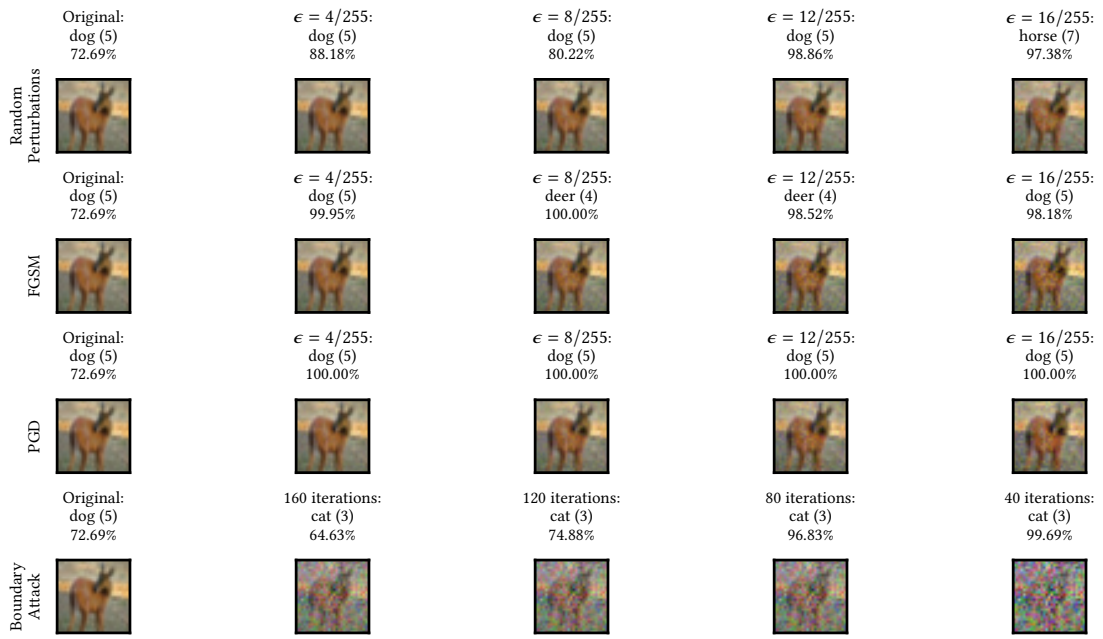
This section contains further adversarial examples that were created using the attacking threat models from section 3.6.2 on adversarially trained DNNs.

CIFAR-10

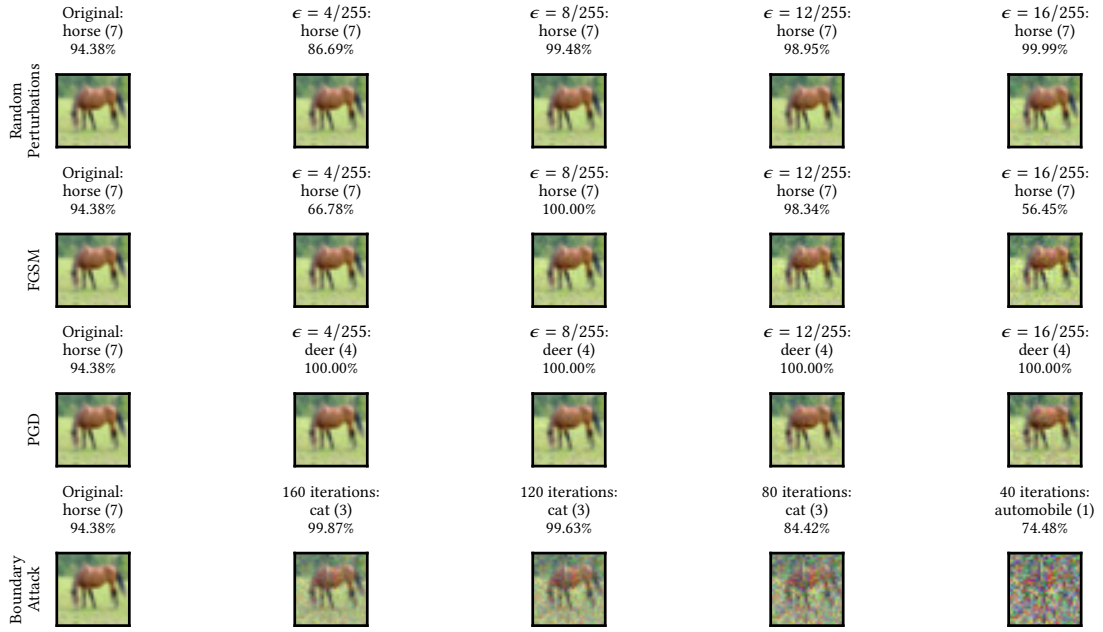
First, figs. C.1 to C.3 show the adversarial examples that are based on the CIFAR-10 images depicted in fig. 4.3 and created on the models being trained adversarially with threat models (A), (C) and (D), respectively. Second, figs. C.4 to C.7 show the adversarial examples of the images illustrated in fig. B.1, but originating from adversarially trained PreAct-ResNet-50 this time.

GTSRB

Analogously, figs. C.8 to C.10 show the adversarial examples that are based on the GTSRB images depicted in fig. 4.6 and created on the models being trained adversarially with threat models (A), (C) and (D), respectively. Furthermore, the adversarial examples of the images illustrated in fig. B.2 which originate from adversarially trained networks are shown in figs. C.11 to C.14.



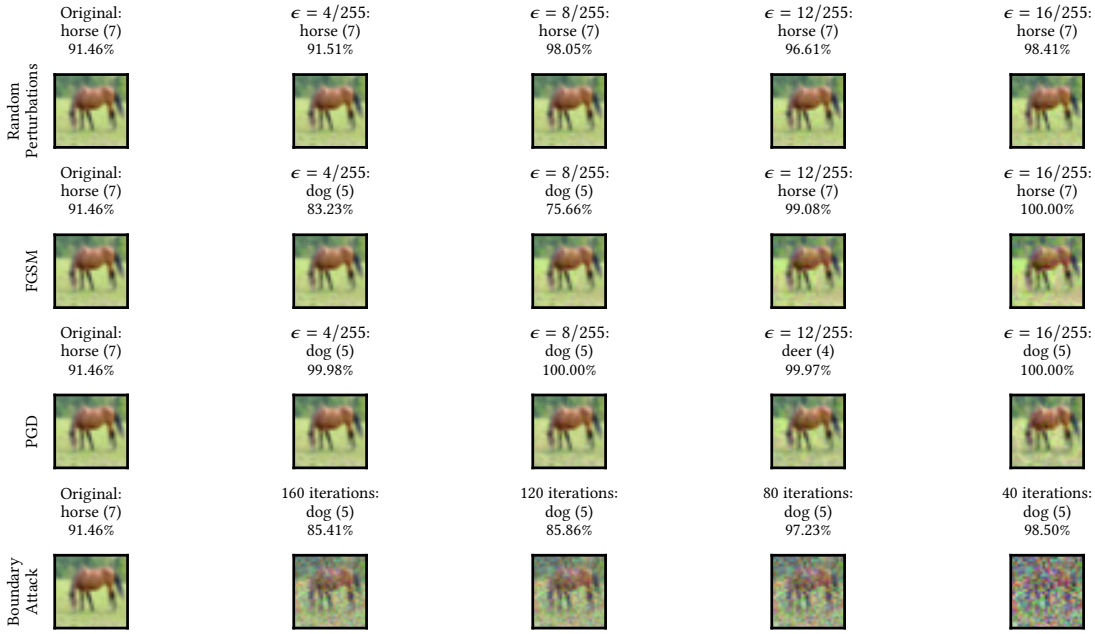
(a) Deer



(b) Horse

Figure C.1.: Adversarial examples of CIFAR-10, created on adversarially (FGSM, maximum ℓ_∞ distance $\epsilon^* = 8/255$) trained PreAct-ResNet-50 (Deer, Horse). Each row shows perturbed versions of a natural image, created with one of the adversarial attacks: random perturbations, FGSM, PGD and Boundary Attack. The original image is shown in the left column. Remaining columns show perturbed versions with maximum ℓ_∞ distance $\epsilon = 4/255, 8/255, 12/255, 16/255$ (from left to right, for random perturbations, FGSM and PGD) and after 40, 80, 120, 160 iterations of Boundary Attack (from right to left), respectively. Classification information is provided above the corresponding image.

(a) Deer



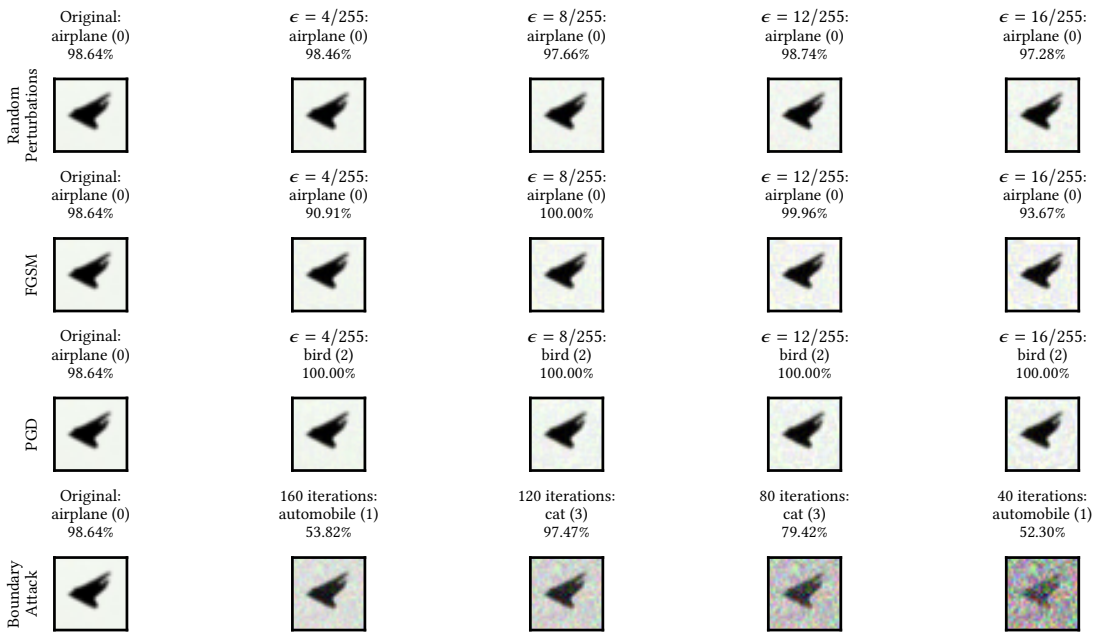
(b) Horse

Figure C.2.: Adversarial examples of CIFAR-10, created on adversarially (FGSM, maximum ℓ_∞ distance $\epsilon^* = 16/255$) trained PreAct-ResNet-50 (Deer, Horse). Each row shows perturbed versions of a natural image, created with one of the adversarial attacks: random perturbations, FGSM, PGD and Boundary Attack. The original image is shown in the left column. Remaining columns show perturbed versions with maximum ℓ_∞ distance $\epsilon = 4/255, 8/255, 12/255, 16/255$ (from left to right, for random perturbations, FGSM and PGD) and after 40, 80, 120, 160 iterations of Boundary Attack (from right to left), respectively. Classification information is provided above the corresponding image.

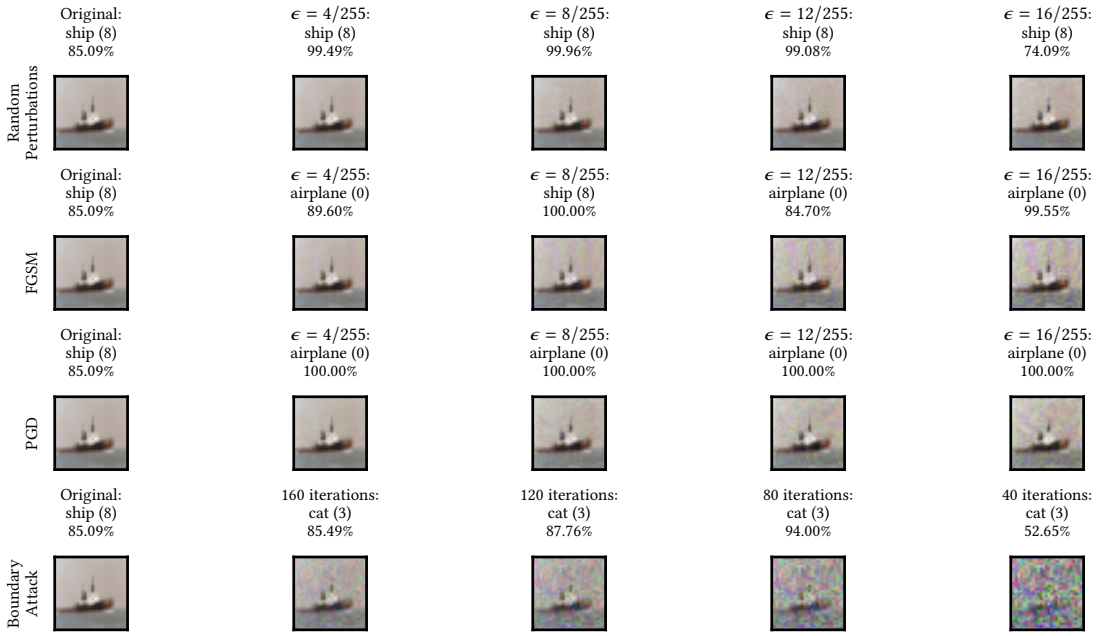
(a) Deer

(b) Horse

Figure C.3.: Adversarial examples of CIFAR-10, created on adversarially (SSS, maximum ℓ_∞ distance $\epsilon^* = 16/255$) trained PreAct-ResNet-50 (Deer, Horse). Each row shows perturbed versions of a natural image, created with one of the adversarial attacks: random perturbations, FGSM, PGD and Boundary Attack. The original image is shown in the left column. Remaining columns show perturbed versions with maximum ℓ_∞ distance $\epsilon = 4/255, 8/255, 12/255, 16/255$ (from left to right, for random perturbations, FGSM and PGD) and after 40, 80, 120, 160 iterations of Boundary Attack (from right to left), respectively. Classification information is provided above the corresponding image.

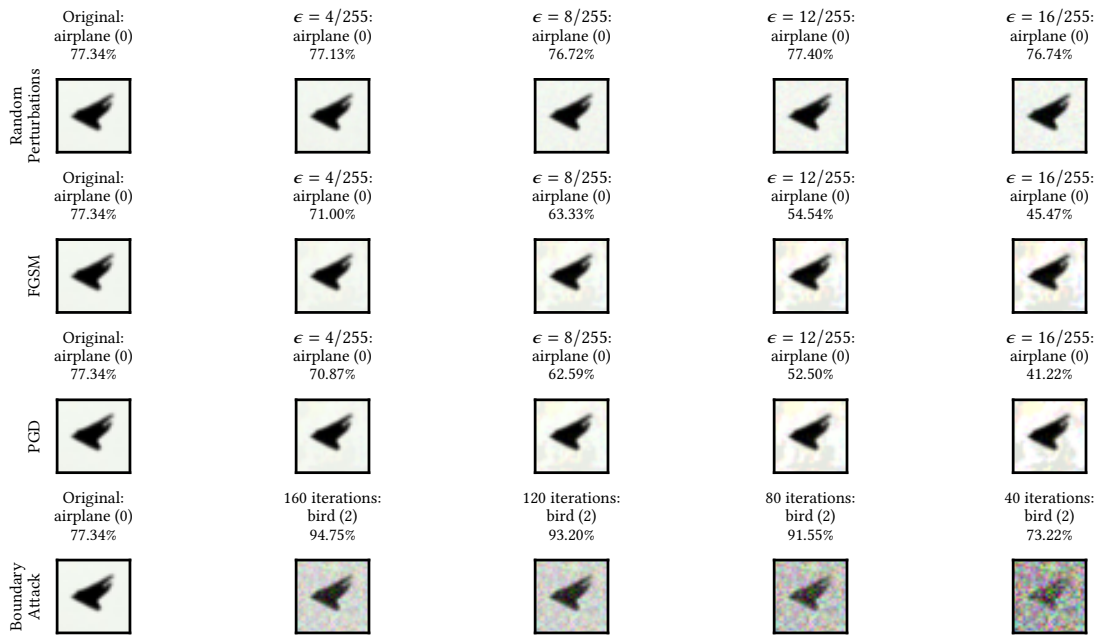


(a) Airplane

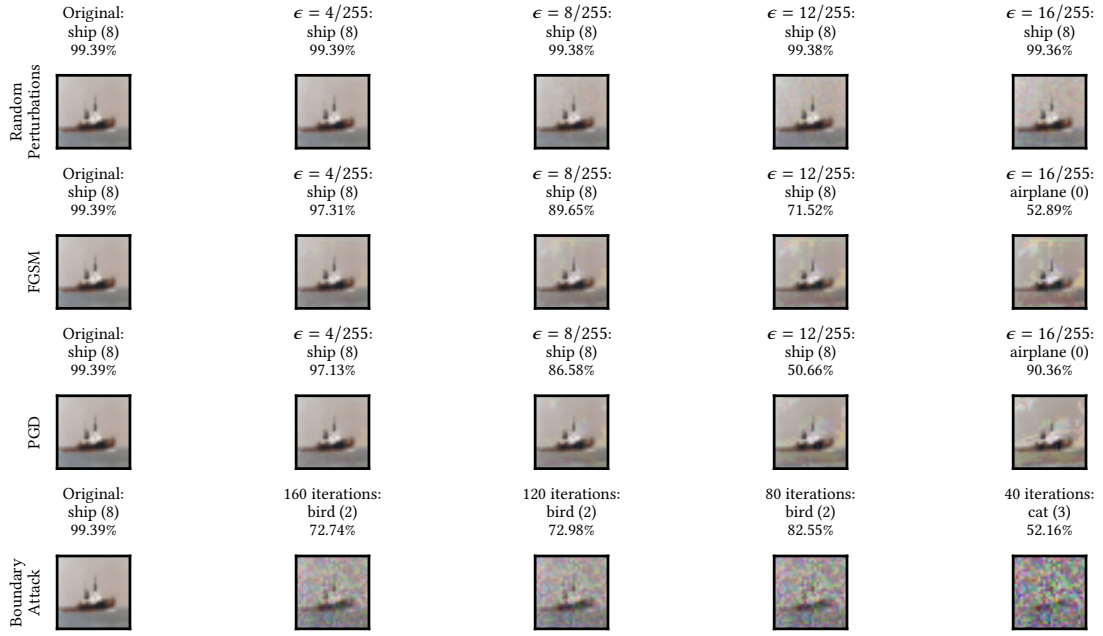


(b) Ship

Figure C.4.: Adversarial examples of CIFAR-10, created on adversarially (FGSM, maximum ℓ_∞ distance $\epsilon^* = 8/255$) trained PreAct-ResNet-50 (Airplane, Ship). Each row shows perturbed versions of a natural image, created with one of the adversarial attacks: random perturbations, FGSM, PGD and Boundary Attack. The original image is shown in the left column. Remaining columns show perturbed versions with maximum ℓ_∞ distance $\epsilon = 4/255, 8/255, 12/255, 16/255$ (from left to right, for random perturbations, FGSM and PGD) and after 40, 80, 120, 160 iterations of Boundary Attack (from right to left), respectively. Classification information is provided above the corresponding image.

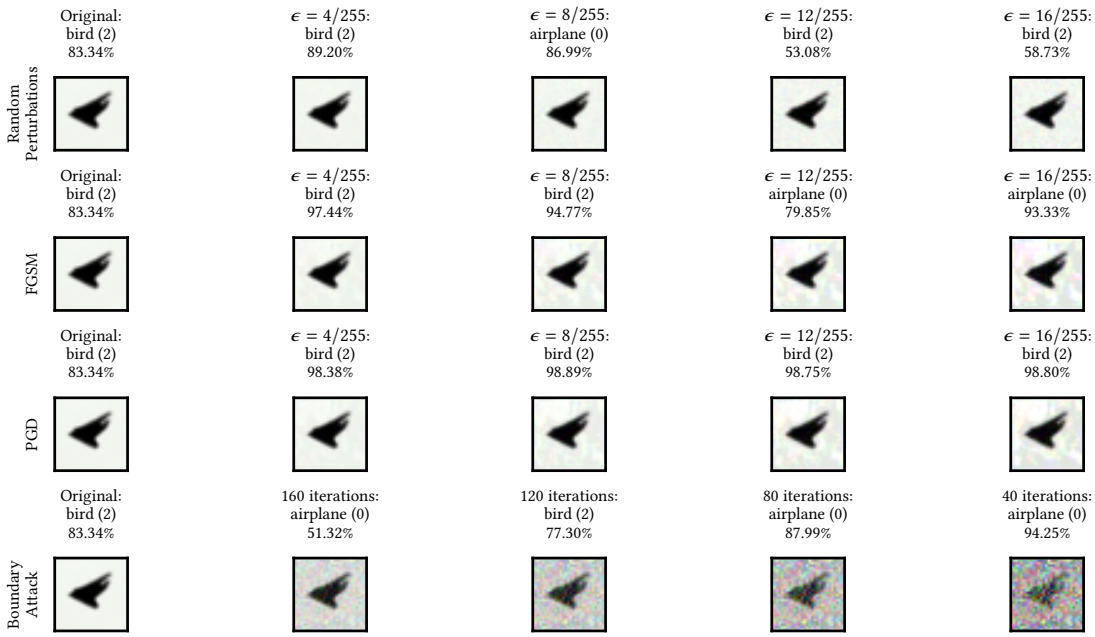


(a) Airplane

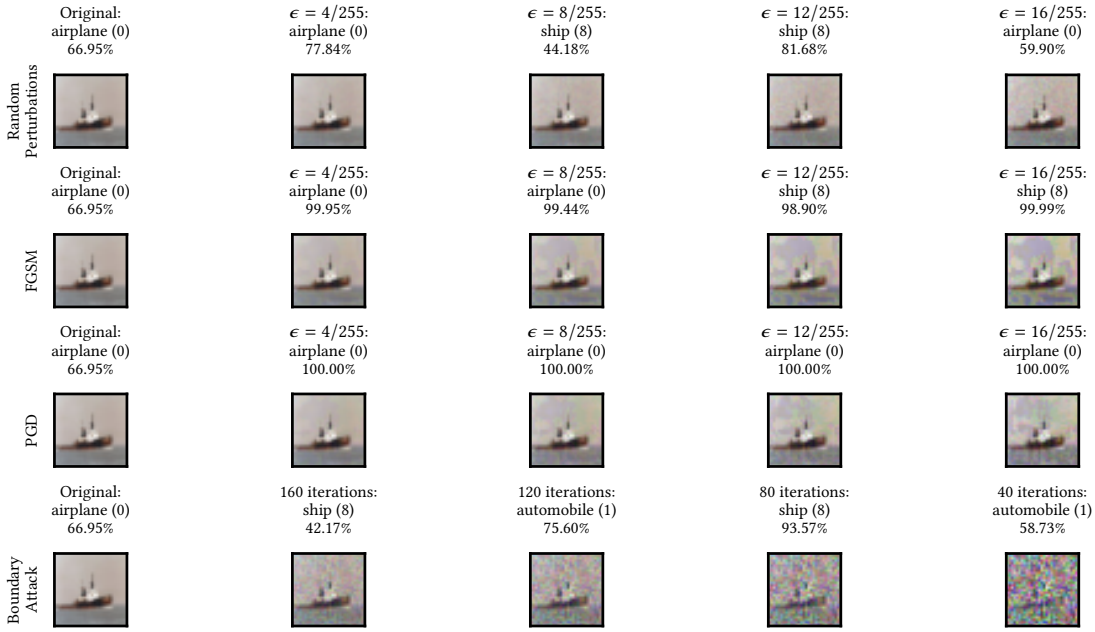


(b) Ship

Figure C.5.: Adversarial examples of CIFAR-10, created on adversarially (SSS, maximum ℓ_∞ distance $\epsilon^* = 8/255$) trained PreAct-ResNet-50 (Airplane, Ship). Each row shows perturbed versions of a natural image, created with one of the adversarial attacks: random perturbations, FGSM, PGD and Boundary Attack. The original image is shown in the left column. Remaining columns show perturbed versions with maximum ℓ_∞ distance $\epsilon = 4/255, 8/255, 12/255, 16/255$ (from left to right, for random perturbations, FGSM and PGD) and after 40, 80, 120, 160 iterations of Boundary Attack (from right to left), respectively. Classification information is provided above the corresponding image.

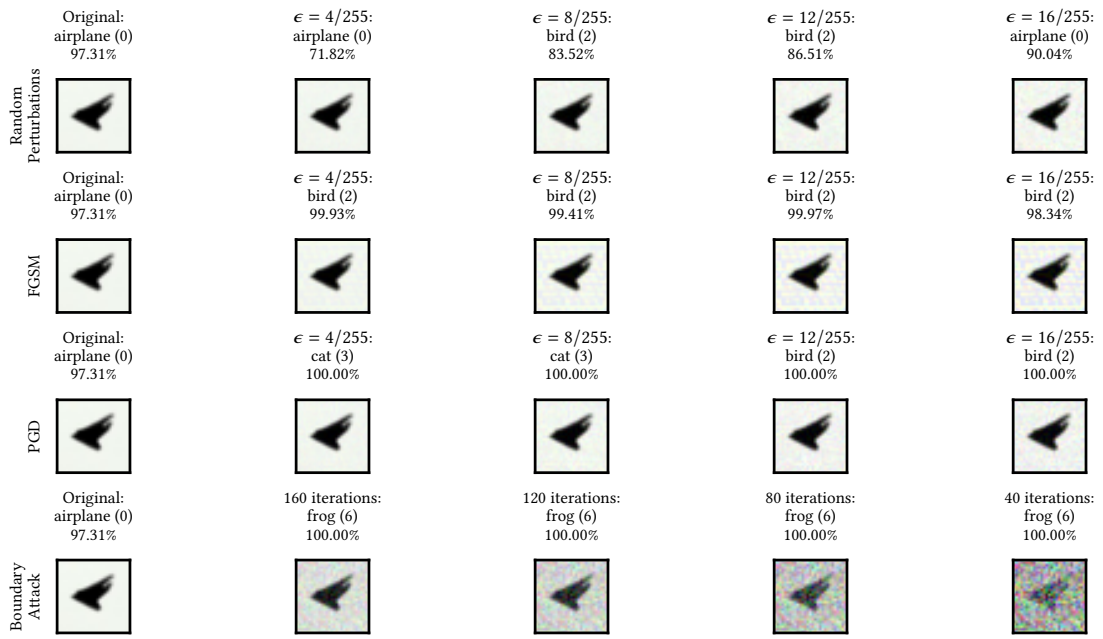


(a) Airplane

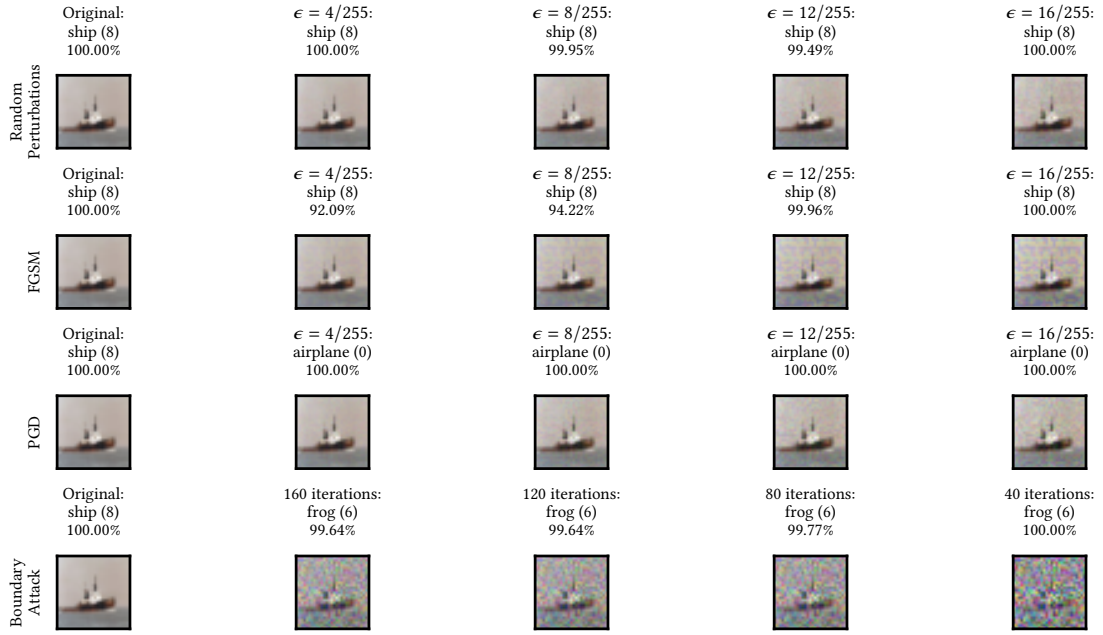


(b) Ship

Figure C.6.: Adversarial examples of CIFAR-10, created on adversarially (FGSM, maximum ℓ_∞ distance $\epsilon^* = 16/255$) trained PreAct-ResNet-50 (Airplane, Ship). Each row shows perturbed versions of a natural image, created with one of the adversarial attacks: random perturbations, FGSM, PGD and Boundary Attack. The original image is shown in the left column. Remaining columns show perturbed versions with maximum ℓ_∞ distance $\epsilon = 4/255, 8/255, 12/255, 16/255$ (from left to right, for random perturbations, FGSM and PGD) and after 40, 80, 120, 160 iterations of Boundary Attack (from right to left), respectively. Classification information is provided above the corresponding image.

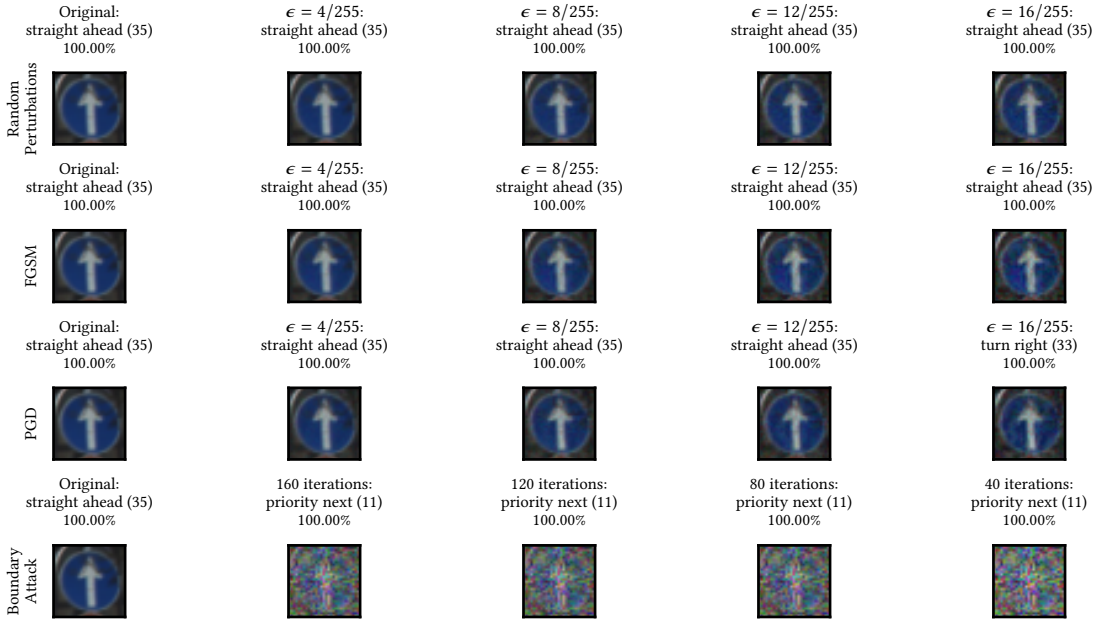


(a) Airplane

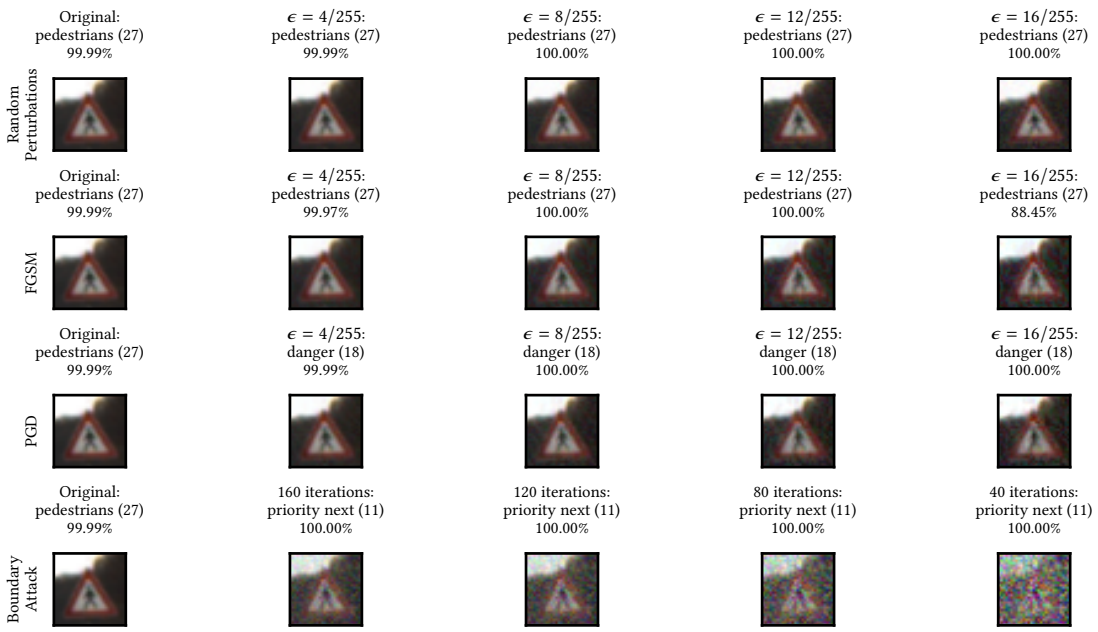


(b) Ship

Figure C.7.: Adversarial examples of CIFAR-10, created on adversarially (SSS, maximum ℓ_∞ distance $\epsilon^* = 16/255$) trained PreAct-ResNet-50 (Airplane, Ship). Each row shows perturbed versions of a natural image, created with one of the adversarial attacks: random perturbations, FGSM, PGD and Boundary Attack. The original image is shown in the left column. Remaining columns show perturbed versions with maximum ℓ_∞ distance $\epsilon = 4/255, 8/255, 12/255, 16/255$ (from left to right, for random perturbations, FGSM and PGD) and after 40, 80, 120, 160 iterations of Boundary Attack (from right to left), respectively. Classification information is provided above the corresponding image.

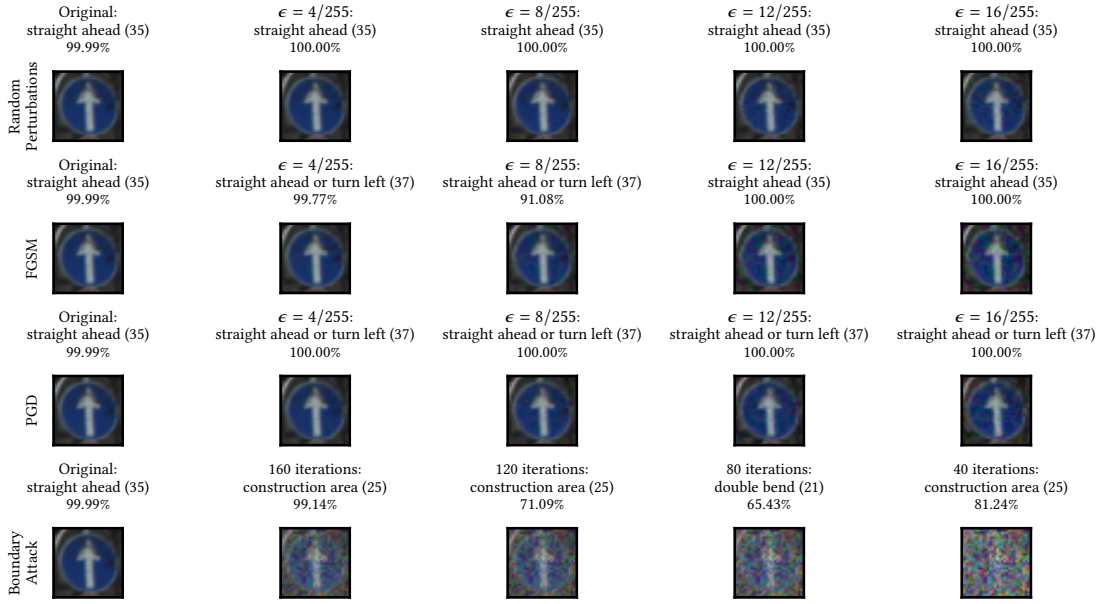


(a) "Straight ahead" sign

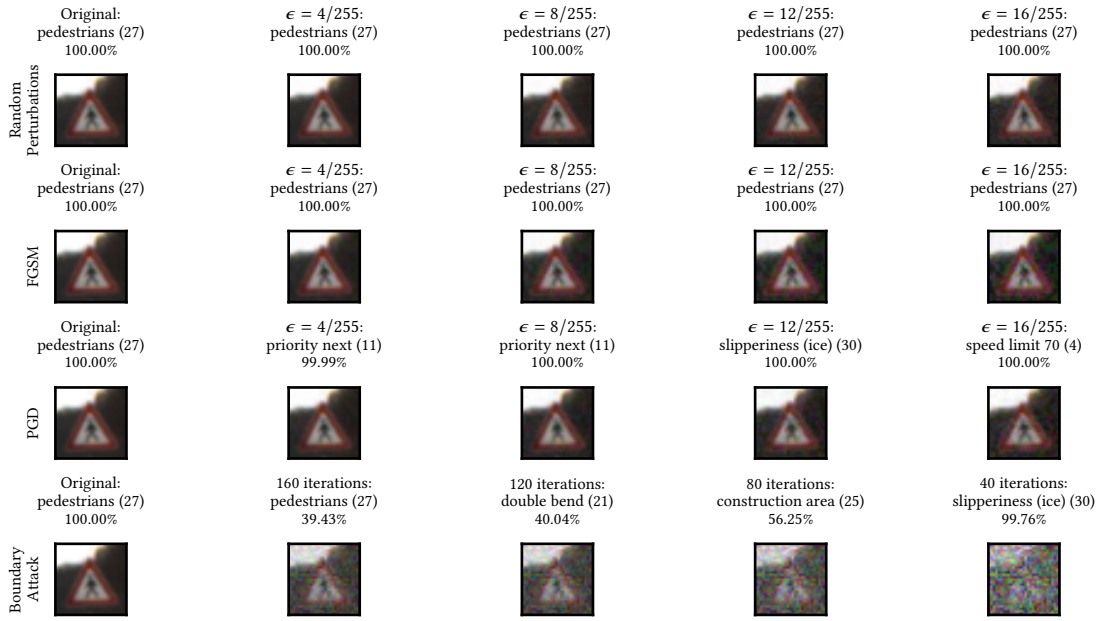


(b) "Pedestrians" sign

Figure C.8.: Adversarial examples of GTSRB, created on adversarially (FGSM, maximum ℓ_∞ distance $\epsilon^* = 8/255$) trained PreAct-ResNet-50 (Straight ahead, Pedestrians). Each row shows perturbed versions of a natural image, created with one of the adversarial attacks: random perturbations, FGSM, PGD and Boundary Attack. The original image is shown in the left column. Remaining columns show perturbed versions with maximum ℓ_∞ distance $\epsilon = 4/255, 8/255, 12/255, 16/255$ (from left to right, for random perturbations, FGSM and PGD) and after 40, 80, 120, 160 iterations of Boundary Attack (from right to left), respectively. Classification information is provided above the corresponding image.

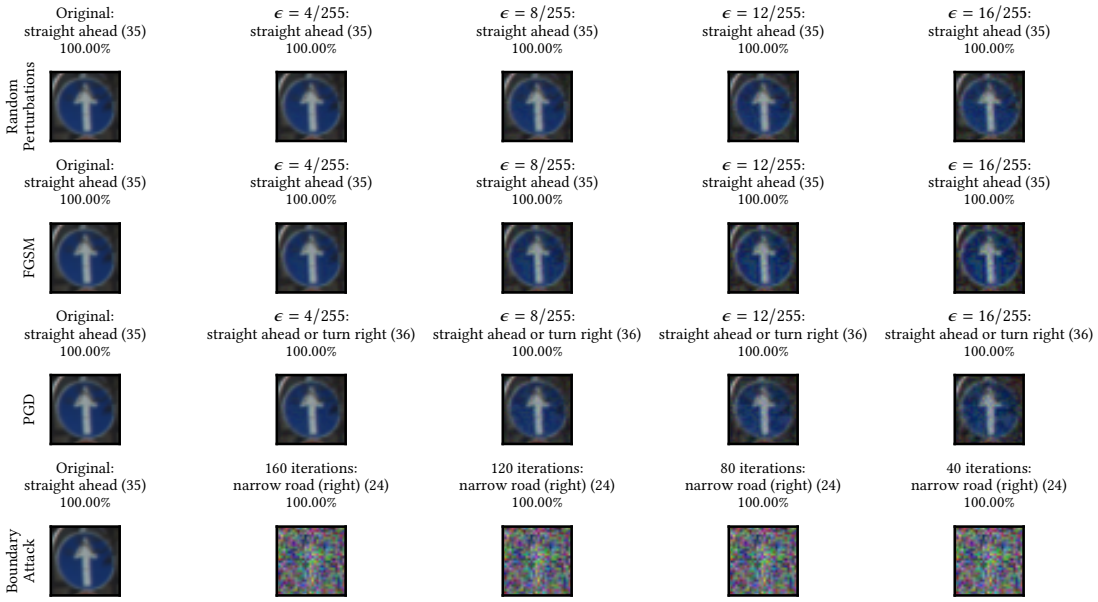


(a) "Straight ahead" sign

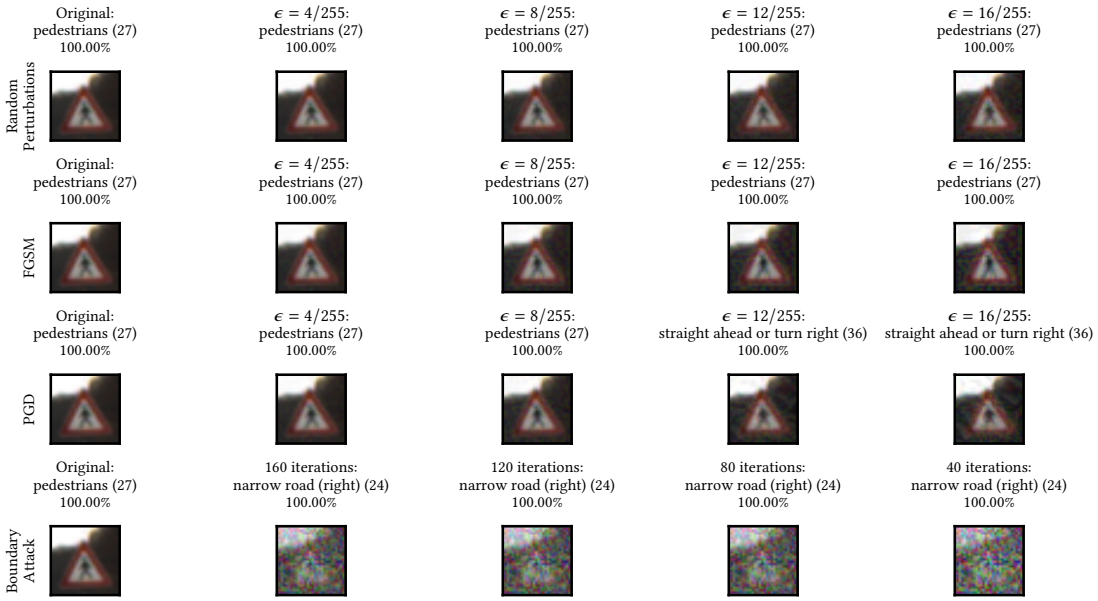


(b) "Pedestrians" sign

Figure C.9.: Adversarial examples of GTSRB, created on adversarially (FGSM, maximum ℓ_∞ distance $\epsilon^* = 16/255$) trained PreAct-ResNet-50 (Straight ahead, Pedestrians). Each row shows perturbed versions of a natural image, created with one of the adversarial attacks: random perturbations, FGSM, PGD and Boundary Attack. The original image is shown in the left column. Remaining columns show perturbed versions with maximum ℓ_∞ distance $\epsilon = 4/255, 8/255, 12/255, 16/255$ (from left to right, for random perturbations, FGSM and PGD) and after 40, 80, 120, 160 iterations of Boundary Attack (from right to left), respectively. Classification information is provided above the corresponding image.

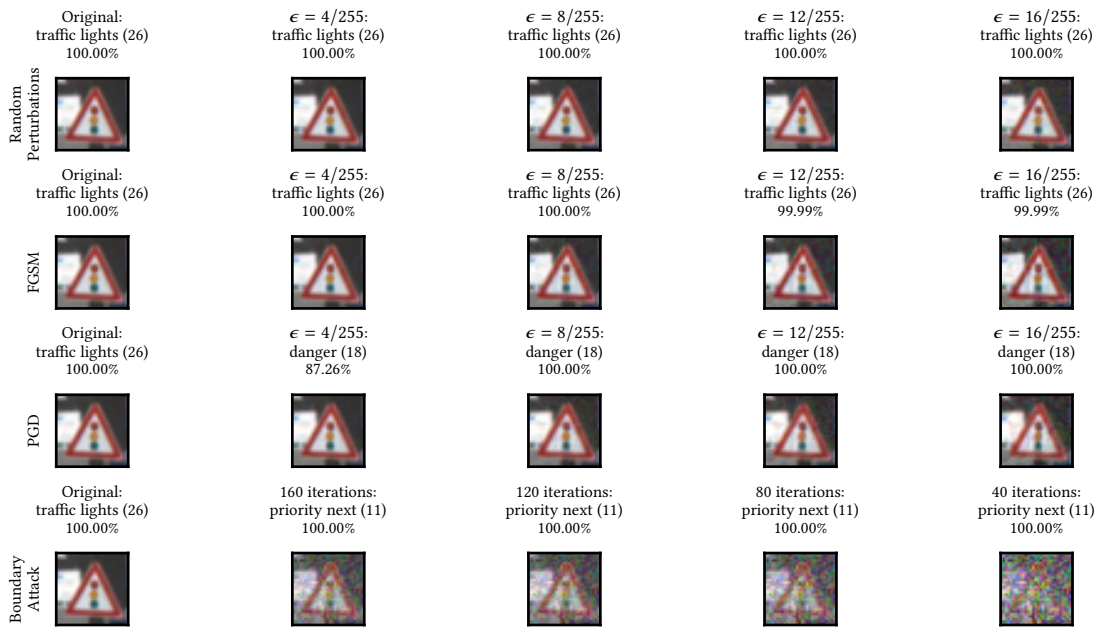


(a) "Straight ahead" sign

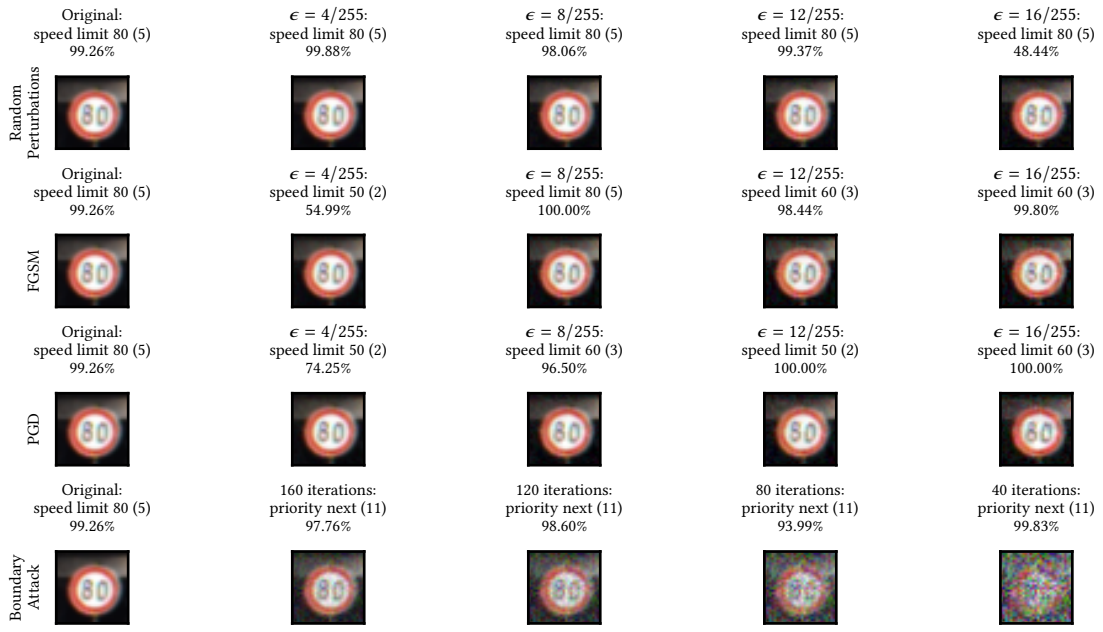


(b) "Pedestrians" sign

Figure C.10.: Adversarial examples of GTSRB, created on adversarially (SSS, maximum ℓ_∞ distance $\epsilon^* = 16/255$) trained PreAct-ResNet-50 (Straight ahead, Pedestrians). Each row shows perturbed versions of a natural image, created with one of the adversarial attacks: random perturbations, FGSM, PGD and Boundary Attack. The original image is shown in the left column. Remaining columns show perturbed versions with maximum ℓ_∞ distance $\epsilon = 4/255, 8/255, 12/255, 16/255$ (from left to right, for random perturbations, FGSM and PGD) and after 40, 80, 120, 160 iterations of Boundary Attack (from right to left), respectively. Classification information is provided above the corresponding image.



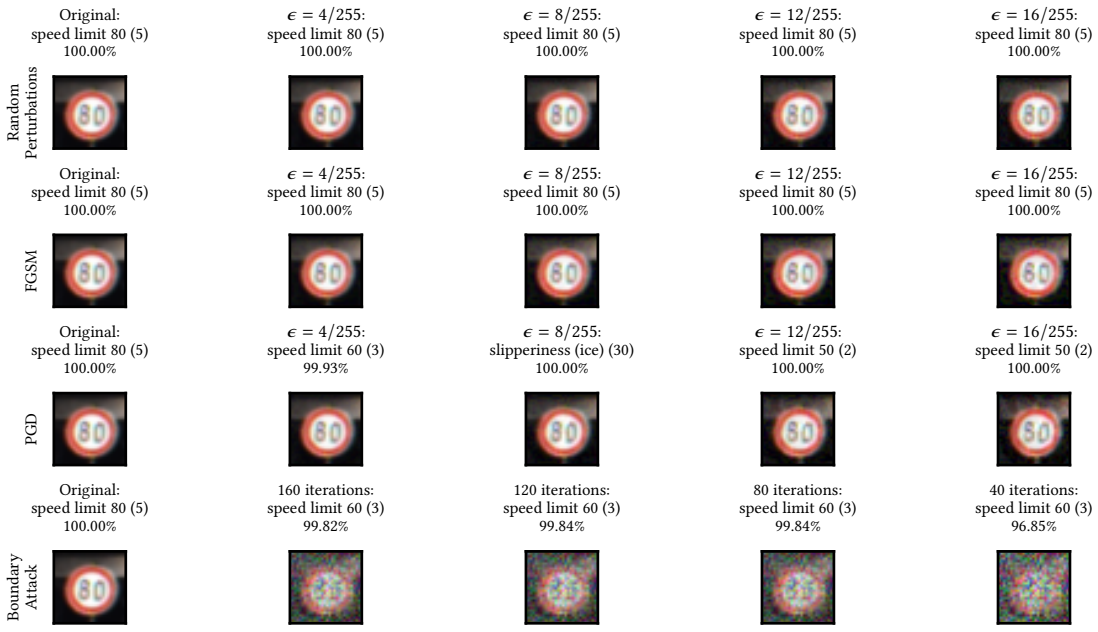
(a) "Traffic lights" sign



(b) "Speed limit 80" sign

Figure C.11.: Adversarial examples of GTSRB, created on adversarially (FGSM, maximum ℓ_∞ distance $\epsilon^* = 8/255$) trained PreAct-ResNet-50 (Traffic Lights, Speed Limit 80). Each row shows perturbed versions of a natural image, created with one of the adversarial attacks: random perturbations, FGSM, PGD and Boundary Attack. The original image is shown in the left column. Remaining columns show perturbed versions with maximum ℓ_∞ distance $\epsilon = 4/255, 8/255, 12/255, 16/255$ (from left to right, for random perturbations, FGSM and PGD) and after 40, 80, 120, 160 iterations of Boundary Attack (from right to left), respectively. Classification information is provided above the corresponding image.

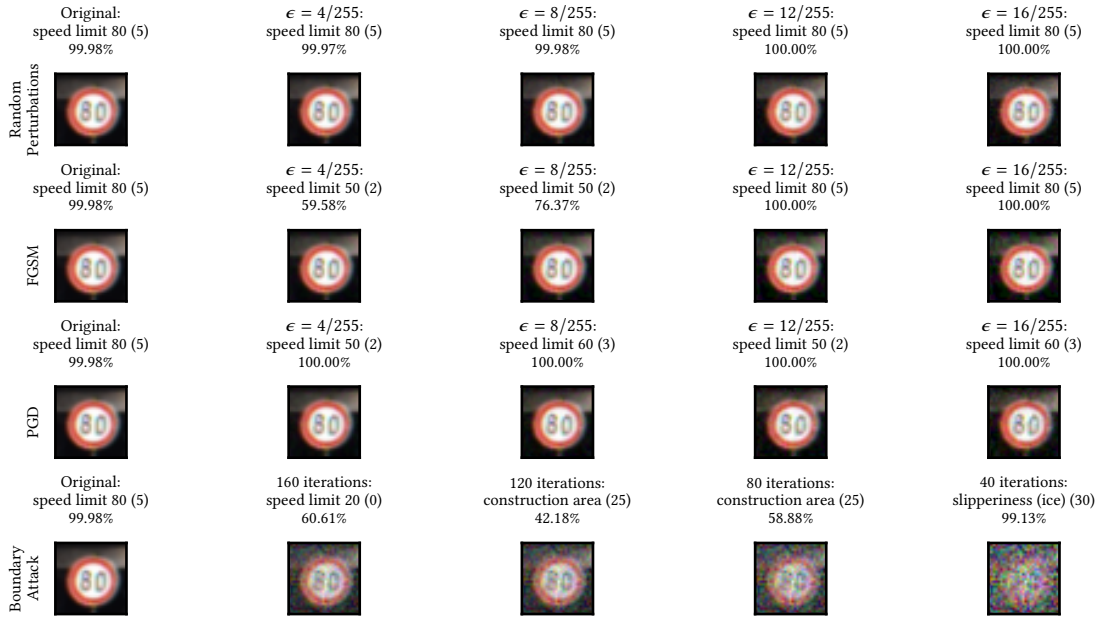
(a) "Traffic lights" sign



(b) "Speed limit 80" sign

Figure C.12.: Adversarial examples of GTSRB, created on adversarially (SSS, maximum ℓ_∞ distance $\epsilon^* = 8/255$) trained PreAct-ResNet-50 (Traffic Lights, Speed Limit 80). Each row shows perturbed versions of a natural image, created with one of the adversarial attacks: random perturbations, FGSM, PGD and Boundary Attack. The original image is shown in the left column. Remaining columns show perturbed versions with maximum ℓ_∞ distance $\epsilon = 4/255, 8/255, 12/255, 16/255$ (from left to right, for random perturbations, FGSM and PGD) and after 40, 80, 120, 160 iterations of Boundary Attack (from right to left), respectively. Classification information is provided above the corresponding image.

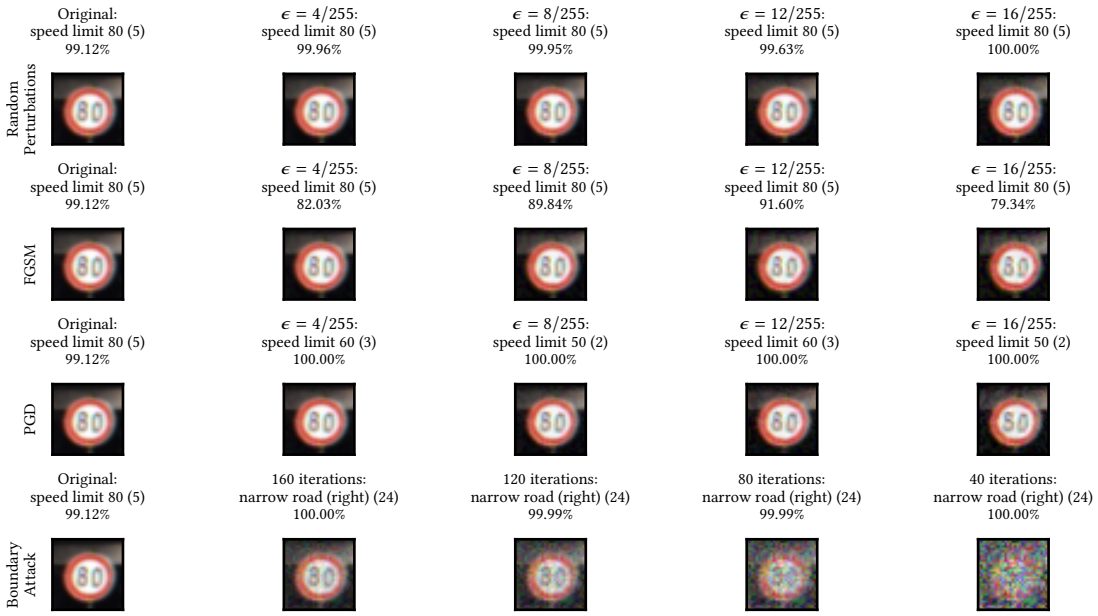
(a) "Traffic lights" sign



(b) "Speed limit 80" sign

Figure C.13.: Adversarial examples of GTSRB, created on adversarially (FGSM, maximum ℓ_∞ distance $\epsilon^* = 16/255$) trained PreAct-ResNet-50 (Traffic Lights, Speed Limit 80). Each row shows perturbed versions of a natural image, created with one of the adversarial attacks: random perturbations, FGSM, PGD and Boundary Attack. The original image is shown in the left column. Remaining columns show perturbed versions with maximum ℓ_∞ distance $\epsilon = 4/255, 8/255, 12/255, 16/255$ (from left to right, for random perturbations, FGSM and PGD) and after 40, 80, 120, 160 iterations of Boundary Attack (from right to left), respectively. Classification information is provided above the corresponding image.

(a) "Traffic lights" sign



(b) "Speed limit 80" sign

Figure C.14.: Adversarial examples of GTSRB, created on adversarially (SSS, maximum ℓ_∞ distance $\epsilon^* = 16/255$) trained PreAct-ResNet-50 (Traffic Lights, Speed Limit 80). Each row shows perturbed versions of a natural image, created with one of the adversarial attacks: random perturbations, FGSM, PGD and Boundary Attack. The original image is shown in the left column. Remaining columns show perturbed versions with maximum ℓ_∞ distance $\epsilon = 4/255, 8/255, 12/255, 16/255$ (from left to right, for random perturbations, FGSM and PGD) and after 40, 80, 120, 160 iterations of Boundary Attack (from right to left), respectively. Classification information is provided above the corresponding image.

D FURTHER LOSS LANDSCAPES

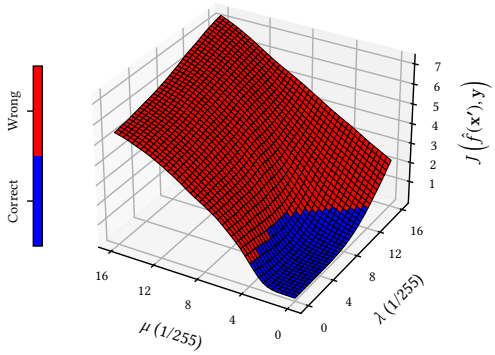
In the following, we also visualize the loss landscapes of standardly and adversarially trained networks w.r.t. the additional images of appendix B.

CIFAR-10

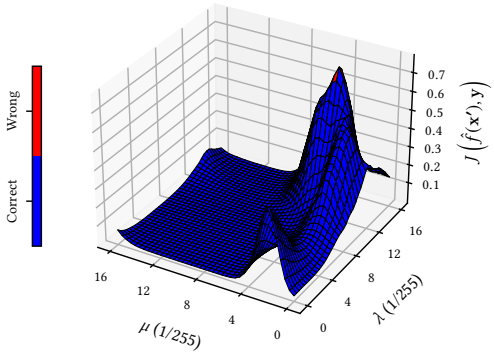
First of all, fig. D.1 shows the loss landscapes w.r.t. the image illustrated in fig. B.1a. Next, the loss landscapes w.r.t. the second additional CIFAR-10 image, depicted in fig. B.1b, are shown in fig. D.2.

GTSRB

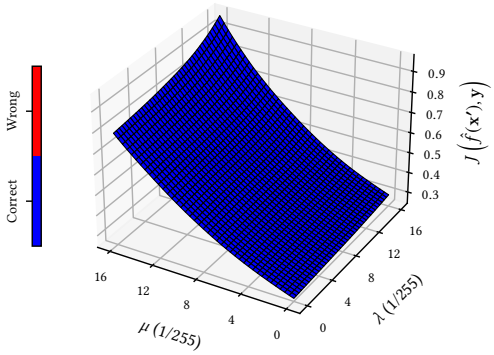
In an analogous way, figs. D.3 and D.4 show the loss landscapes w.r.t. the images illustrated in figs. B.2a and B.2b.



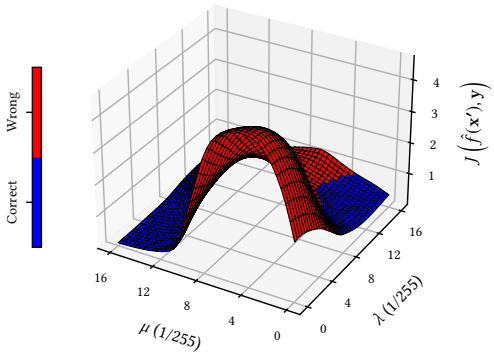
(a) Standard Training



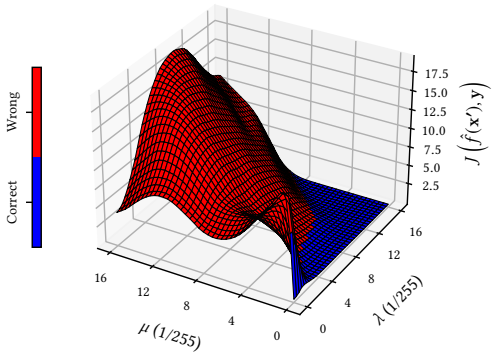
(b) Adversarial training: FGSM, $\epsilon^* = 8/255$



(c) Adversarial training: SSS, $\epsilon^* = 8/255$

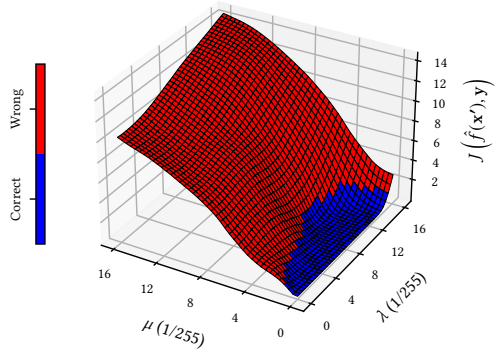


(d) Adversarial training: FGSM, $\epsilon^* = 16/255$

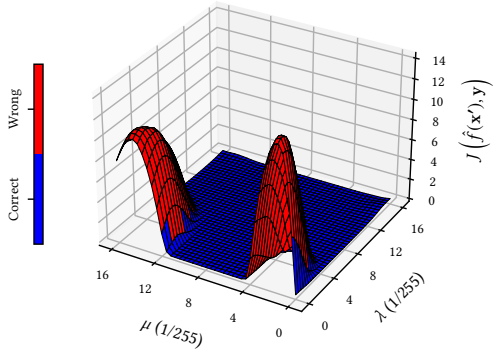


(e) Adversarial training: SSS, $\epsilon^* = 16/255$

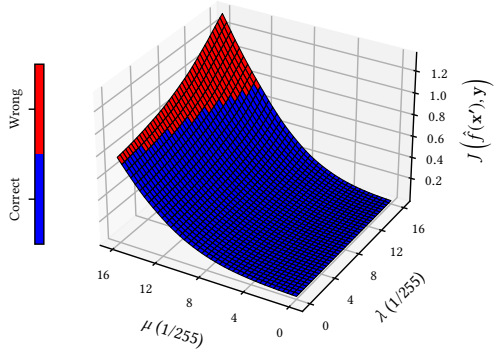
Figure D.1.: Loss landscapes from PreAct-ResNet-50, trained standardly (top) and adversarially using FGSM (left) and SSS (right) at maximum ℓ_∞ distances $\epsilon^* = 8/255$ (middle) and $\epsilon^* = 16/255$ (bottom) on CIFAR-10 (Airplane). In all cases, the natural image from the validation dataset (see fig. B.1a) marks the origin. Moreover, the left axis points in the adversarial direction and the right one in a random one. Both are scaled in multiples of 1/255 according to the ∞-norm. Last, correctly classified perturbations are colored blue, wrongly ones red.



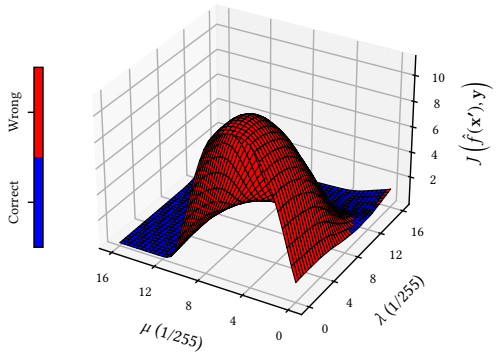
(a) Standard Training



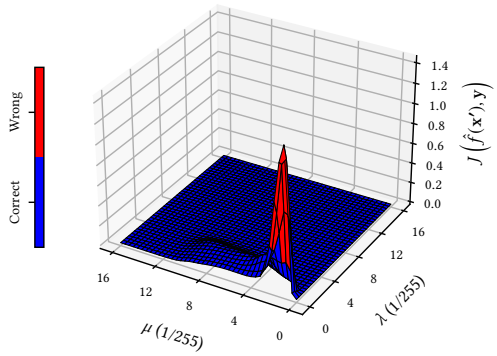
(b) Adversarial training: FGSM, $\epsilon^* = 8/255$



(c) Adversarial training: SSS, $\epsilon^* = 8/255$

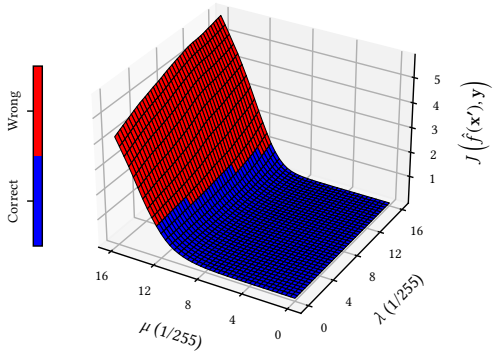


(d) Adversarial training: FGSM, $\epsilon^* = 16/255$

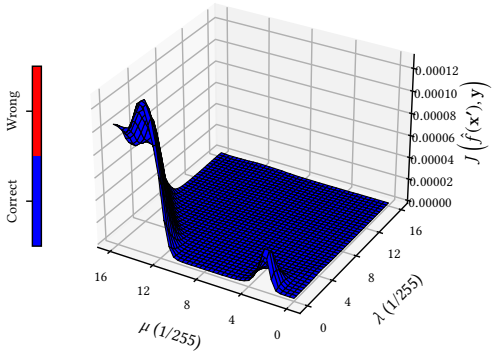


(e) Adversarial training: SSS, $\epsilon^* = 16/255$

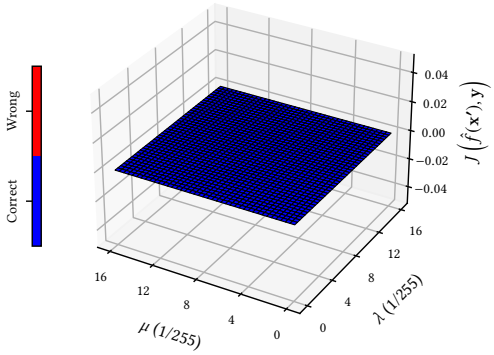
Figure D.2.: Loss landscapes from PreAct-ResNet-50, trained standardly (top) and adversarially using FGSM (left) and SSS (right) at maximum ℓ_∞ distances $\epsilon^* = 8/255$ (middle) and $\epsilon^* = 16/255$ (bottom) on CIFAR-10 (Ship). In all cases, the natural image from the validation dataset (see fig. B.1b) marks the origin. Moreover, the left axis points in the adversarial direction and the right one in a random one. Both are scaled in multiples of 1/255 according to the ∞-norm. Last, correctly classified perturbations are colored blue, wrongly ones red.



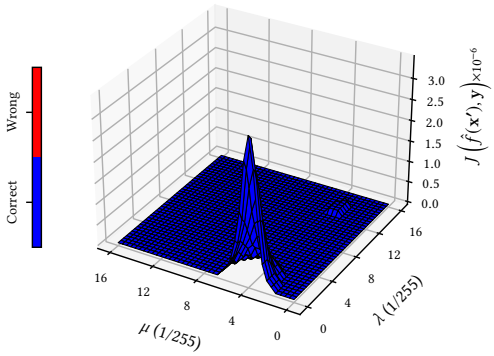
(a) Standard training



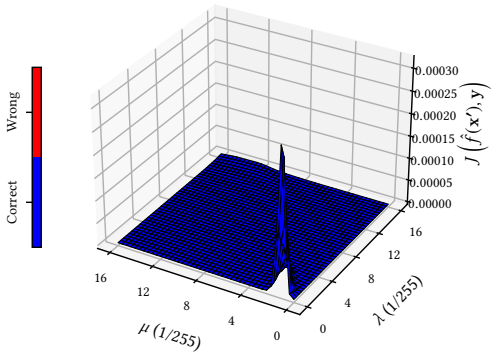
(b) Adversarial training: FGSM, $\epsilon^* = 8/255$



(c) Adversarial training: SSS, $\epsilon^* = 8/255$

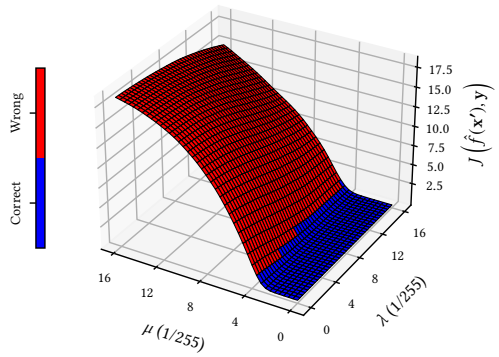


(d) Adversarial training: FGSM, $\epsilon^* = 16/255$

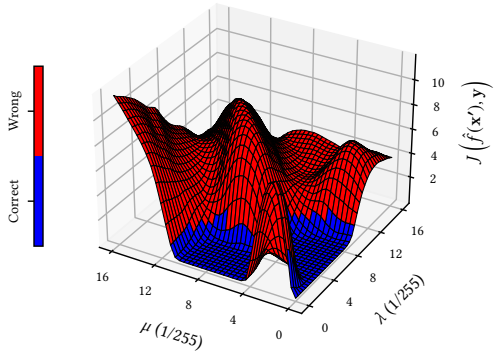


(e) Adversarial training: SSS, $\epsilon^* = 16/255$

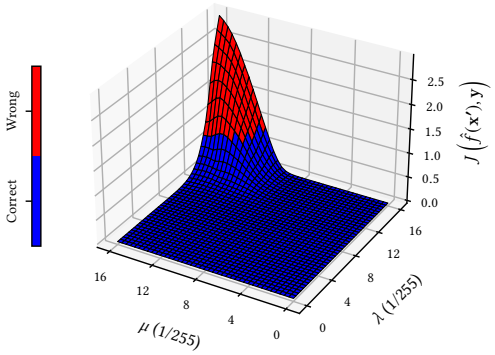
Figure D.3.: Loss landscapes from PreAct-ResNet-50, trained standardly (top) and adversarially using FGSM (left) and SSS (right) at maximum ℓ_∞ distances $\epsilon^* = 8/255$ (middle) and $\epsilon^* = 16/255$ (bottom) on GTSRB (Traffic Lights). In all cases, the natural image from the validation dataset (see fig. B.2a) marks the origin. Moreover, the left axis points in the adversarial direction and the right one in a random one. Both are scaled in multiples of $1/255$ according to the ∞-norm. Last, correctly classified perturbations are colored blue, wrongly ones red.



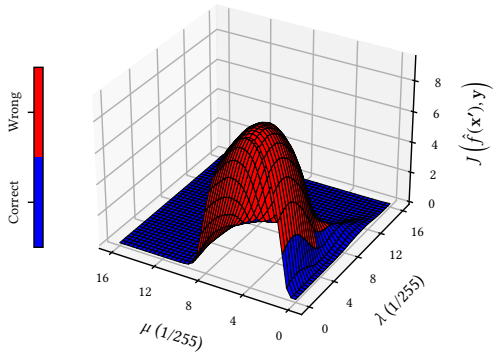
(a) Standard training



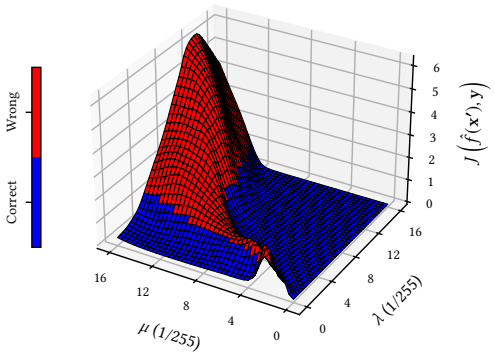
(b) Adversarial training: FGSM, $\epsilon^* = 8/255$



(c) Adversarial training: SSS, $\epsilon^* = 8/255$



(d) Adversarial training: FGSM, $\epsilon^* = 16/255$



(e) Adversarial training: SSS, $\epsilon^* = 16/255$

Figure D.4.: Loss landscapes from PreAct-ResNet-50, trained standardly (top) and adversarially using FGSM (left) and SSS (right) at maximum ℓ_∞ distances $\epsilon^* = 8/255$ (middle) and $\epsilon^* = 16/255$ (bottom) on GTSRB (Speed limit 80). In all cases, the natural image from the validation dataset (see fig. B.2b) marks the origin. Moreover, the left axis points in the adversarial direction and the right one in a random one. Both are scaled in multiples of 1/255 according to the ∞-norm. Last, correctly classified perturbations are colored blue, wrongly ones red.

BIBLIOGRAPHY

[1] Ronan Collobert and Jason Weston. „A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning“. In: *Proceedings of the 25th International Conference on Machine Learning*. Association for Computing Machinery, July 5, 2008, pp. 160–167. ISBN: 978-1-60558-205-4. DOI: 10.1145/1390156.1390177.

[2] Tomas Mikolov et al. „Distributed Representations of Words and Phrases and Their Compositionality“. In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. Curran Associates Inc., Dec. 5, 2013, pp. 3111–3119. URL: https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.

[3] A. Graves, A. Mohamed, and G. Hinton. „Speech Recognition with Deep Recurrent Neural Networks“. In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. May 2013, pp. 6645–6649. DOI: 10.1109/ICASSP.2013.6638947.

[4] G. Hinton et al. „Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups“. In: *IEEE Signal Processing Magazine* 29.6 (Nov. 2012), pp. 82–97. DOI: 10.1109/MSP.2012.2205597.

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

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. „ImageNet Classification with Deep Convolutional Neural Networks“. In: *Communications of the ACM* 60.6 (May 24, 2017), pp. 84–90. DOI: 10.1145/3065386.

[7] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. Apr. 10, 2015. arXiv: 1409.1556 [cs].

[8] C. Szegedy et al. „Going Deeper with Convolutions“. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015, pp. 1–9. DOI: 10.1109/CVPR.2015.7298594.

[9] Mariusz Bojarski et al. *End to End Learning for Self-Driving Cars*. Apr. 25, 2016. arXiv: 1604.07316 [cs].

[10] C. Chen et al. „DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving“. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. Dec. 2015, pp. 2722–2730. DOI: 10.1109/ICCV.2015.312.

[11] J. Ker et al. „Deep Learning Applications in Medical Image Analysis“. In: *IEEE Access* 6 (2018), pp. 9375–9389. DOI: 10.1109/ACCESS.2017.2788044.

[12] Muhammad Imran Razzak, Saeeda Naz, and Ahmad Zaib. „Deep Learning for Medical Image Processing: Overview, Challenges and the Future“. In: *Classification in BioApps: Automation of Decision Making*. Springer International Publishing, 2018, pp. 323–350. ISBN: 978-3-319-65981-7. DOI: 10.1007/978-3-319-65981-7_12.

[13] Christian Szegedy et al. *Intriguing Properties of Neural Networks*. Feb. 19, 2014. arXiv: 1312.6199 [cs].

[14] Nicholas Carlini. *A Complete List of All Adversarial Example Papers*. June 9, 2021. URL: https://nicholas.carlini.com/writing/2019/all-adversarial-example-papers.html (visited on 06/09/2021).

[15] N. Carlini and D. Wagner. „Towards Evaluating the Robustness of Neural Networks“. In: *2017 IEEE Symposium on Security and Privacy (SP)*. May 2017, pp. 39–57. DOI: 10.1109/SP.2017.49.

[16] Anish Athalye, Nicholas Carlini, and David Wagner. „Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples“. In: *Proceedings of the 35th International Conference on Machine Learning*. Vol. 80. PMLR, July 3, 2018, pp. 274–283. URL: http://proceedings.mlr.press/v80/athalye18a/athalye18a.pdf.

[17] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. *Transferability in Machine Learning: From Phenomena to Black-Box Attacks Using Adversarial Samples*. May 23, 2016. arXiv: 1605.07277 [cs].

[18] Nicolas Papernot et al. „Practical Black-Box Attacks against Machine Learning“. In: *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. Association for Computing Machinery, Apr. 2, 2017, pp. 506–519. ISBN: 978-1-4503-4944-4. DOI: 10.1145/3052973.3053009.

[19] S. Moosavi-Dezfooli et al. „Universal Adversarial Perturbations“. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017, pp. 86–94. DOI: 10.1109/CVPR.2017.17.

[20] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and Harnessing Adversarial Examples*. Mar. 20, 2015. arXiv: 1412.6572 [cs, stat].

[21] Aleksander Madry et al. *Towards Deep Learning Models Resistant to Adversarial Attacks*. June 19, 2017. arXiv: 1706.06083 [cs, stat].

[22] Connor Shorten and Taghi M. Khoshgoftaar. „A Survey on Image Data Augmentation for Deep Learning". In: *Journal of Big Data* 6.1 (July 6, 2019), p. 60. DOI: 10.1186/s40537-019-0197-0.

[23] Eric Wong and Zico Kolter. „Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope". In: *Proceedings of the 35th International Conference on Machine Learning*. Vol. 80. PMLR, July 10–15, 2018, pp. 5286–5295. URL: http://proceedings.mlr.press/v80/wong18a.html.

[24] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. *Certified Defenses against Adversarial Examples*. Oct. 31, 2020. arXiv: 1801.09344 [cs].

[25] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. „Certified Adversarial Robustness via Randomized Smoothing". In: *Proceedings of the 36th International Conference on Machine Learning*. Vol. 97. PMLR, June 9–15, 2019, pp. 1310–1320. URL: http://proceedings.mlr.press/v97/cohen19c.html.

[26] Jamie Hayes. „Extensions and Limitations of Randomized Smoothing for Robustness Guarantees". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. June 2020, pp. 3413–3421. DOI: 10.1109/CVPRW50498.2020.00401.

[27] Yann LeCun, Corinna Cortes, and Christopher J. C. Burges. *The MNIST Database of Handwritten Digits*. 1998. URL: http://yann.lecun.com/exdb/mnist/ (visited on 06/15/2021).

[28] Richard E. Bellman. *Adaptive Control Processes*. Princeton University Press, Dec. 8, 2015 [1961]. ISBN: 978-1-4008-7466-8. DOI: 10.1515/9781400874668.

[29] Kaiming He et al. „Identity Mappings in Deep Residual Networks". In: *Computer Vision – ECCV 2016*. Springer International Publishing, 2016, pp. 630–645. ISBN: 978-3-319-46493-0. DOI: 10.1007/978-3-319-46493-0_38.

[30] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. Technical report. Apr. 8, 2009. URL: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf (visited on 06/15/2021).

[31] Johannes Stallkamp et al. „The German Traffic Sign Recognition Benchmark: A Multi-Class Classification Competition". In: *The 2011 International Joint Conference on Neural Networks*. July 2011, pp. 1453–1460. DOI: 10.1109/IJCNN.2011.6033395.

[32] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org.

[33] G. Cybenko. „Approximation by Superpositions of a Sigmoidal Function“. In: *Mathematics of Control, Signals and Systems* 2.4 (Dec. 1, 1989), pp. 303–314. DOI: 10.1007/BF02551274.

[34] Y. LeCun et al. „Backpropagation Applied to Handwritten Zip Code Recognition“. In: *Neural Computation* 1.4 (Dec. 1, 1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

[35] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, Sept. 7, 2012. 1104 pp.

[36] Dimitri P. Bertsekas. *Nonlinear Programming*. 3. Edition. Athena Scientific, June 27, 2016. 861 pp. ISBN: 978-1-886529-05-2.

[37] Anna Choromanska et al. „The Loss Surfaces of Multilayer Networks“. In: *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*. Vol. 38. PMLR, Feb. 21, 2015, pp. 192–204. URL: http://proceedings.mlr.press/v38/choromanska15.pdf.

[38] Yann LeCun et al. „Efficient BackProp“. In: *Neural Networks: Tricks of the Trade, This Book Is an Outgrowth of a 1996 NIPS Workshop*. Springer-Verlag, Jan. 1, 1998, pp. 9–50. ISBN: 978-3-540-65311-0. DOI: 10.1007/978-3-642-35289-8.

[39] Marat V. Markin. *Elementary Functional Analysis*. De Gruyter, Oct. 8, 2018. ISBN: 978-3-11-061403-9. DOI: 10.1515/9783110614039.

[40] C. D. (Cyrus D.) Cantrell. *Modern Mathematical Methods for Physicists and Engineers*. Cambridge, UK ; New York : Cambridge University Press, 2000. 790 pp. ISBN: 978-0-521-59827-9.

[41] N. Papernot et al. „The Limitations of Deep Learning in Adversarial Settings“. In: *2016 IEEE European Symposium on Security and Privacy (EuroS P)*. Mar. 2016, pp. 372–387. DOI: 10.1109/EuroSP.2016.36.

[42] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. „DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks“. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016, pp. 2574–2582. DOI: 10.1109/CVPR.2016.282.

[43] Xavier Glorot and Yoshua Bengio. „Understanding the Difficulty of Training Deep Feedforward Neural Networks“. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Vol. 9. PMLR, May 13–15, 2010, pp. 249–256. URL: http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf.

[44] Sergey Ioffe and Christian Szegedy. „Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift“. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning*. Vol. 37. PMLR, July 6, 2015, pp. 448–456. URL: http://proceedings.mlr.press/v37/ioffe15.pdf.

[45] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. Jan. 4, 2019. arXiv: 1711.05101 [cs, math].

[46] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. Dec. 22, 2014. arXiv: 1412.6980 [cs].

[47] Ilya Loshchilov and Frank Hutter. *SGDR: Stochastic Gradient Descent with Warm Restarts*. May 3, 2017. arXiv: 1608.03983 [cs, math].

[48] Yinpeng Dong et al. „Boosting Adversarial Attacks with Momentum“. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. June 2018, pp. 9185–9193. DOI: 10.1109/CVPR.2018.00957.

[49] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. *Adversarial Examples in the Physical World*. July 8, 2016. arXiv: 1607.02533 [cs, stat].

[50] Sébastien Bubeck. „Convex Optimization: Algorithms and Complexity“. In: *Foundations and Trends® in Machine Learning* 8.3-4 (Nov. 11, 2015), pp. 231–357. DOI: 10.1561/2200000050.

[51] Wieland Brendel, Jonas Rauber, and Matthias Bethge. *Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models*. Feb. 16, 2018. arXiv: 1712.04248 [cs, stat].

[52] 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.

[53] Hoki Kim, Woojin Lee, and Jaewook Lee. „Understanding Catastrophic Overfitting in Single-Step Adversarial Training“. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.9 (May 2021), pp. 8119–8127. arXiv: 2010.01799. URL: https://ojs.aaai.org/index.php/AAAI/article/view/16989/16796.

[54] Eric Wong, Leslie Rice, and J. Zico Kolter. *Fast Is Better than Free: Revisiting Adversarial Training*. Jan. 12, 2020. arXiv: 2001.03994 [cs, stat].

[55] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, 2009. ISBN: 1-4414-1269-7.

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

[57] Charles R. Harris et al. „Array Programming with NumPy“. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2.

[58] J. D. Hunter. „Matplotlib: A 2D Graphics Environment“. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[59] Florian Tramèr et al. *The Space of Transferable Adversarial Examples*. May 23, 2017. arXiv: 1704.03453 [cs, stat].

[60] Charles Fefferman, Sanjoy Mitter, and Hariharan Narayanan. „Testing the Manifold Hypothesis“. In: *Journal of the American Mathematical Society* 29.4 (Oct. 2016), pp. 983–1049. DOI: 10.1090/jams/852.

[61] Ali Shafahi et al. *Are Adversarial Examples Inevitable?* Sept. 6, 2018. arXiv: 1809.02104 [cs, stat].

[62] Jia Deng et al. „ImageNet: A Large-Scale Hierarchical Image Database“. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. June 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[63] Kevin Eykholt et al. „Robust Physical-World Attacks on Deep Learning Visual Classification“. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. June 2018, pp. 1625–1634. DOI: 10.1109/CVPR.2018.00175.