GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Fakultät für
Physik

Master's Thesis

Robust Perception in Humans and Machines

Robuste Perzeption in Menschen und Maschinen

prepared by

Roland Simon Zimmermann

from Recklinghausen

at the Institute for Theoretical Physics, Tübingen University

Thesis period:	October 1, 2019 until February 5, 2020
Supervisor:	Dr. Wieland Brendel
First referee:	Prof. Dr. Matthias Bethge
Second referee:	Prof. Dr. Florentin Wörgötter

Abstract

Under optimal conditions, modern computer vision algorithms based on neural networks can provide almost perfect performance on various perceptual tasks, such as image classification. However, these classifiers are often not robust and their performance deteriorates severely when faced with corrupted input images. Research on the robustness of image classifiers focuses on two types of image corruptions: adversarial examples and common corruptions. While the former are image perturbations created by an adversary to cause wrong behavior of the algorithm, the latter describe image distortions that naturally occur outside of perfect lab conditions. In this work, new methods to evaluate the robustness of image classifiers against both types of corruption as well as approaches to increase the robustness are presented.

In a first step, the robustness of a defense strategy against adversarial perturbations, based on Bayesian neural networks and adversarial training, is analyzed. It is shown that by adjusting an existing adversarial attack to this defense, the defense becomes ineffective.

Next, two methods are derived to assess the robustness of arbitrary image classifiers against image corruptions. This is motivated by the goal of performing a fair comparison between current image classification networks and human visual perception in terms of their robustness. The first method is a decision-based adversarial attack and searches image-dependent adversarial perturbations using Markov Chain Monte Carlo sampling. It can be implemented using a two-alternatives-forced-choice task in a psychophysical experiment. The attack reaches competitive results compared to previous white-box attacks. In the second method, an image-independent and dataset-dependent noise distribution is learned by an adversarial noise generator. The experiments indicate that image classifiers based on convolutional neural networks are more susceptible to this noise than to various other distributions such as Gaussian noise.

Finally, it is demonstrated that the robustness of a common image classification network (ResNet-50) against common corruptions can be improved strongly by using a simple but properly tuned Gaussian data augmentation. This simple baseline easily reaches previous state-of-the-art performance on the popular corruption benchmark ImageNet-C. Built upon this strong baseline and the previously proposed adversarial noise generator, it is shown that an adversarial training of the recognition model against uncorrelated worst-case noise distributions leads to an additional increase in performance.

Keywords: Machine Learning, Artificial Neural Networks, Adversarial Examples, Robustness, Psychophysics

Contents

Nomenclature

Abbreviations

Abbreviation	Meaning
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
ES	Evolution Strategy
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
MCMC	Markov Chain Monte Carlo
MCMC-P	Markov Chain Monte Carlo with People
CMP	Constrained Markov Chain Monte Carlo with People
PC	Principal Component

Letters and Symbols

Style	Mathematical Object
a, α	Scalars
$\mathbf{b}$, $\boldsymbol{\beta}$	Vectors
$\mathbf{C}$, $\boldsymbol{\Gamma}$	Matrices

Symbol	Meaning
N	Image classifier
$\mathrm{N_p}$	Probability distribution over classes induced by classifier N
$\mathrm{N_l}$	Log-Probabilities induced by classifier N
$\mathbf{x}$	Clean image
$\mathbf{x}'$	Adversarial image
$\mathcal{L}_{\mathrm{CE}}$	Cross-Entropy loss function

1. Introduction

For a long time, humanity has tried to create machines that are capable of solving specific tasks or imitating human behavior [68]. While this endeavor has been rather unsuccessful for many decades, progress has picked up speed, especially in the recent years [43, 58, 69, 78, 80]. With the wide-spread availability of high-performance computing systems, the research interest in this area increased and many methods proposed years ago could finally be applied and scaled up to more complex tasks. In the last few years alone, machine learning algorithms such as Deep Neural Networks (DNNs) and Convolutional Neural networks (CNNs) have shown a strong performance improvement. This resulted in models that are able to perform on par with human performance or even surpass it in several tasks such as image classification, speech recognition and gaming [13, 42, 43, 78, 89, 98].

Despite the great progress over the last years, many questions regarding the behavior of DNNs still remain unanswered. By answering those questions, one can hope to further increase the performance of artificial intelligence. Therefore, cases in which one can observe unexpected or even undesired behavior of an artificial agent can be seen as opportunities to gain a better understanding of how these agents operate. This also includes failure cases, e.g. situations in which the agent fails entirely to solve a task. Thus, by identifying and analyzing failure cases, one can look for two things: gaining a better understanding of the general behavior and decision process and finding challenging situations that can be used as benchmarks to direct further development of artificial intelligence.

One such failure case is the susceptibility of DNNs to adversarial examples [94]. An adversarial example is a small perturbation of the input datum that changes the decision of a DNN in arbitrary directions, but which is very subtle and usually imperceptible for humans. In the case of an image classification task, an adversarial example is a small modification of the input image, imperceptible for a human observer, that changes the predicted class of the network [35, 94]. These perturba-

tions are of interest to researchers because of two reasons. First, as long as these examples exist, adversaries can fool DNNs. Therefore, they cannot be applied to security-relevant tasks without redundancy or supervision [9, 10]. Second, these examples can be used to both analyze the decision boundaries of DNNs as well as to understand which visual features their decisions are based on [30, 32, 48, 74, 95]. Thus, the research on adversarial examples is currently two-fold: on the one hand, researchers try to reduce the risk posed by adversarial examples by developing defense mechanisms against these [52, 62, 92] and on the other hand researchers improve the understanding of DNNs in the light of adversarial examples [48].

Modern solutions such as DNNs do not only fail in the presence of minimal adversarial perturbations but can also fail when confronted with input data that was affected by a naturally occurring corruption. These common corruptions are not constrained to be imperceptible to a human observer but rather include all types of corruptions that can occur in real-world scenarios outside of perfect lab conditions. They include simple Gaussian or Salt and Pepper noise; natural variations like rain, snow or fog; and compression artifacts such as those caused by JPEG encoding [45]. All of these corruptions do not change the semantic content of the input, and thus, machine learning models should not change their decision-making behavior in their presence. In many practical real-world applications, robustness to common corruptions is often more relevant than robustness to artificially designed adversarial perturbations. Autonomous cars should not change their behavior in the face of unusual weather conditions such as hail, sand storms or small pixel defects in their sensors. Even though the effects of these common corruptions are less severe than those of adversarial examples, facing them in real-world scenarios is far more likely and they therefore present a higher risk. Thus, many researchers have also worked on improving the robustness against such common corruptions in addition to robustness against adversarial attacks [45, 63, 99].

To steer future research in computer vision, it is reasonable to assess the current state in this field. On the one hand, this includes benchmarking results across different computer vision algorithms (e.g. [84]) and on the other hand comparing the performance of those algorithms and of humans (e.g. [27–29, 51]). While the former helps to find benefits and downsides of specific algorithmic choices, the latter helps to put the current state-of-the-art in perspective: a common hypothesis is that the

more similar computer vision approaches become to human vision, the better they will also perform in real-world applications. Therefore, by conducting such comparative studies, one cannot only measure the performance gap between humans and machines in general but also for specific sub-tasks or in specific situations, e.g. image classification performance under common corruptions [27]. Once this gap is measured, one can hope to make conclusions about the weaknesses and strengths of the algorithms and to define future research questions (e.g. [76]).

To improve the robustness of computer vision algorithms against both adversarial perturbations and common corruptions, it is reasonable to assess the currently achieved level of robustness by comparing the performance with humans [23, 24, 27, 51, 91, 102]. In the last two years, there have been multiple studies that tried to answer how the robustness of computer vision algorithms compares to the robustness of humans. Specifically for adversarial examples, there is still no widely accepted answer, as some studies showed that adversarial examples do not only affect CNNs but also humans [24, 102] while others showed the opposite to be true [23]. All of these studies applied a similar scheme: crafting adversarial examples for a CNN and testing their influence on a human. In this work, I argue that such an approach is not the most appropriate way to compare the two, as it is a priori not clear that humans and CNNs should be sensitive to the same type of adversarial examples. Therefore, by finding a way to craft adversarial examples in a comparative way and at the same time specific to humans and CNNs, one obtains a tool that is better suited for comparing the two.

This work is divided into four chapters. In the beginning, an introduction and a short summary of concepts and notations used in this thesis are given (Chapter 2). This includes Artificial Neural Networks (ANNs), adversarial examples and different optimization techniques. The summary can be skipped by those familiar with such concepts.

Next, a proposed defense mechanism against adversarial attacks (Adv-BNN), which is based on the concept of Bayesian Neural Networks (BNNs), is analyzed (Chapter 3). A thorough evaluation of the method shows that it fails to significantly increase the robustness.

In the following two chapters, two different approaches to compare CNNs and humans in terms of their robustness against both adversarial examples as well as

image corruptions are developed. These can be used to put the performance of CNNs in relation to human performance in the face of image corruptions. For this, two new adversarial attacks that can be performed on both CNNs and humans are introduced. The former attack uses spatially uncorrelated and image-independent noise (Chapter 4) and is optimized by an evolution strategy (ES). The latter uses highly correlated noise (Chapter 5) and is optimized by a modified version of Markov-Chain-Monte-Carlo optimization (MCMC).

In the final chapter, the application of the previously introduced uncorrelated noise to improve the robustness of CNNs against various types of correlations is analyzed (Chapter 6). It is demonstrated that by using uncorrelated noise, one can already achieve a surprisingly high level of robustness against common image corruptions.

2. Background

2.1. Datasets

In this work, tools to analyze the robustness of algorithms in the domain of computer vision are developed. A special focus is set to image classification tasks. For this, one needs, on the one hand, a dataset which defines the task, and on the other hand, an algorithm to solve it. In the following, the different datasets that are used shall be introduced briefly.

2.1.1. MNIST

In 1998 LeCun et al. [59] introduced the MNIST dataset containing images of handwritten digits between 0 and 9. The images in this dataset have a spatial resolulion of 28×28 pixels and are grayscale. In the following, the dataset is always preprocessed such that the pixels have values in the range $[0, 1]$. The dataset is split into a train and a test subset which contain 50,000 and 10,000 images, respectively. Exemplary images are shown in Figure 2.1. Because of its low spatial resolution and the rather low complexity compared to other image datasets, MNIST is a popular toy dataset to test novel machine learning algorithms.

Figure 2.1.: Exemplary images of the MNIST dataset.

2.1.2. CIFAR-10

With the aim of testing and developing novel neural networks that extract image features and perform image classification on a different task than digit recognition on MNIST, Krizhevsky et al. [56] created the CIRAR-10 dataset. This is a set of color images with a spatial resolution of 32×32 pixels. There are, just like in MNIST, 60,000 images which are split into a train and a test subset of 50,000 and 10,000 images, respectively. There are 10 different classes of images in this dataset: airplanes, birds, cars, cats, deer, dogs, frogs, horses, ships, and trucks. Every image is assigned to one of these classes according to the object shown. Exemplary images for each of these classes are shown in Figure 2.2. One caveat of the CIFAR-10 dataset is its low spatial resolution and the resulting poor image quality.

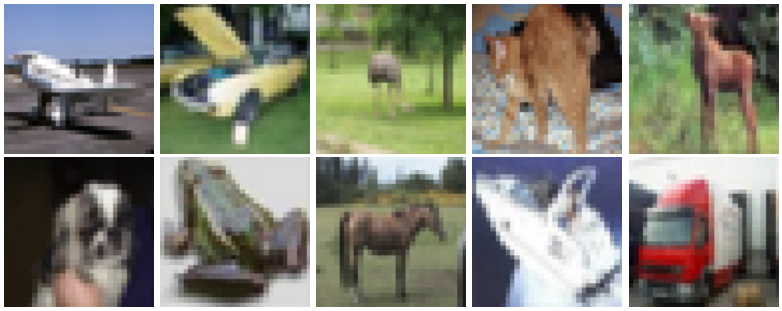


Figure 2.2.: Exemplary images of the CIFAR-10 dataset.

2.1.3. STL-10

The STL-10 dataset is a relatively small dataset which only contains 13,000 labeled images split into 5,000 train and 8,000 test samples, and an additional larger set of unlabeled images [17]. Originally, the dataset has been created to advance research on unsupervised image recognition algorithms, which explains the low number of labeled images. Nevertheless, it is sometimes also used as a benchmark dataset for supervised image classification. Like CIFAR-10, this dataset contains color images of 10 different classes. Also, the classes are almost identical to CIFAR-10, with the exception of one: instead of images of frogs STL-10 contains images of monkeys. An illustration of some exemplary images for each of these classes is shown in Figure 2.3. The spatial resolution of the images is larger than in both MNIST and CIFAR-10 and amounts to 96×96 pixels.

Figure 2.3.: Exemplary images of the STL-10 dataset.

2.1.4. ImageNet-16

As the performance of computer vision based image classification algorithms increased, researchers desired to work on more complex datasets that are closer to real-world scenarios. In the course of this, Deng et al. [21] created the ImageNet dataset in 2009: a collection of more than 14 million images of 21,841 different object classes which were crawled from the internet. The images were assigned to classes that were taken from WordNet [72], which is a hierarchical model of English words. This means, that every class corresponds to one English noun which itself is part of different hierarchical groups. For example, a Golden Retriever is a member of the groups 'hunting dog', 'dog' and 'animal'. Due to the large number of classes, this dataset is highly imbalanced. Therefore, the authors also released the ILSVRC2012 subset, which is smaller and more balanced. It contains color images of objects corresponding to 1,000 different classes and is an often-used benchmark for image classification. Nowadays, the term ImageNet is normally used to refer to the ILSVRC2012 subset and not the complete original dataset. In the remainder of this work, this practice is also adopted.

The differences between the different classes in ImageNet varies: some of the classes are very similar to each other (e.g. different breeds of dogs) and some are more distinct (e.g. cats compared to knives). As such fine-grained classification can only be performed by a specifically trained human, it is very cumbersome to perform psychophysical experiments on the ImageNet dataset. Therefore, Geirhos et al. [29] created a new subset of ImageNet with only 16 classes. They searched for 16 meta-groups that can be used to describe multiple classes at the same time such that each of these 16 classes is very distinct for a human. For this, they mapped the hierarchical groups of the ImageNet/WordNet structure to the less finely grained

	Geirhos et al.	Balanced ImageNet-16	
Class	[29]	Train	Test
Airplane	1013	1000	50
Bear	4344	1000	50
Bicycle	2391	1000	50
Bird	57529	1000	50
Boat	5301	1000	50
Bottle	5659	1000	50
Car	3507	1000	50
Cat	6964	1000	50
Chair	3359	1000	50
Clock	2122	1000	50
Dog	112023	1000	50
Elephant	2355	1000	50
Keyboard	1771	1000	50
Truck	2839	1000	50
Knife	586	586	50
Oven	892	892	50
Σ	213555	15478	800

Table 2.1.: Comparison of number of samples per class for the dataset introduced by Geirhos et al. [29] and the balanced version called ImageNet-16. While the original dataset does not have a distinct test set, this is introduced in the balanced version. Furthermore, the new version contains approximately the same number of samples per class.

object structure of the COCO dataset, which is an often used object detection dataset with 80 distinct classes [60]. The result was a new dataset with the 16 classes shown in the first column of Table 2.1. To ensure that all images have the same size they resized the images and used the center-crop with a size of 224×224 pixels. Exemplary images are shown in Figure 2.4.

In their work, Geirhos et al. [29] were only interested in an upper bound of the performance of machine learning models, which means that they did not have to create a separate test set to validate the performance. Additionally, their data was highly imbalanced as the class 'dog' contained more than 100,000 images while the class 'knife' contained less than 1000 images (second column of Table 2.1). The meaningfulness of the most often used metric for image classification - the classification accuracy - is reduced by a strong class imbalance. Therefore, this work introduces a new dataset called ImageNet-16 based on Geirhos et al. [29], that is both balanced across classes and is split into a train and a test dataset (2.1).

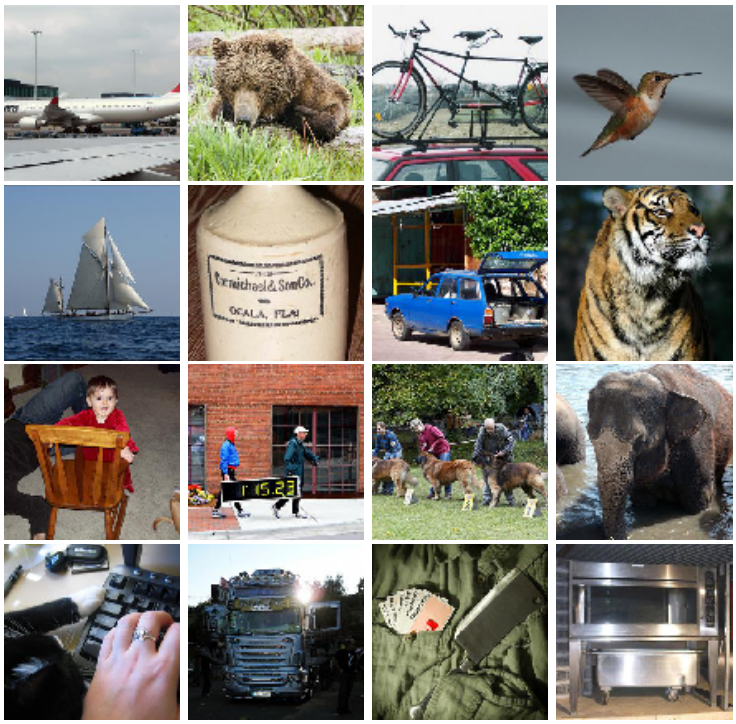


Figure 2.4.: Exemplary images of the ImageNet-16 dataset. The image classes correspond to the classes listed in Table 2.1 in direction of reading.

Its training set contains approximately 1,000 images per class while there are 50 samples per class in the validation set.

2.1.5. ImageNet-C

The ImageNet-C dataset was proposed by Hendrycks and Dietterich [45] as a benchmark to test the robustness of image classifiers against a diverse set of 15 image corruptions. An example of how these corruptions look like can be found in Figure 2.5. These image corruptions resemble distortions that can either occur because of natural causes (e.g. weather effects) or because of flawed image sensors (e.g. noise). The 15 corruptions of the dataset are organized in four main categories (Noise, Blur, Weather and Digital) and have five levels of severity each to reflect varying intensities of the corruptions in real world applications. In summary, this results in 75 different corruptions. As the name of the ImageNet-C dataset suggests, it is based on the ImageNet dataset. However, the authors only applied the 75 corruptions to the validation subset of ImageNet. Because of their diverse corruption types, one

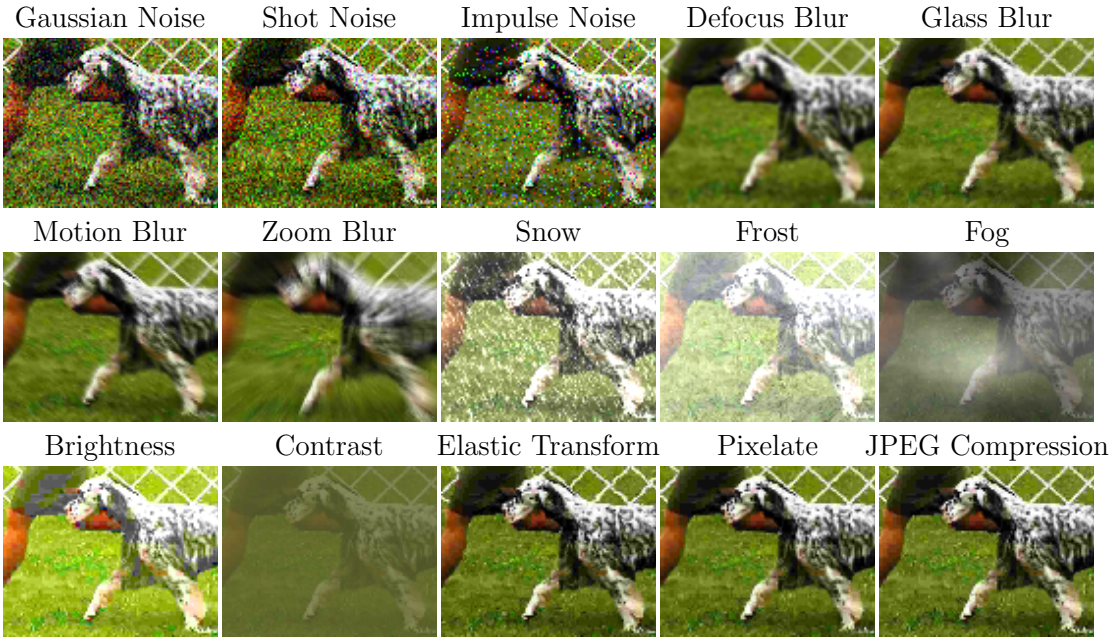


Figure 2.5.: Exemplary images of the ImageNet-C dataset. The same image is shown for all 15 corruption types using severity level 3 out of 5, where corruption is stronger for higher levels.

can see the ImageNet-C dataset as a proxy for the most common corruptions in the real world. This property is only guaranteed as long as one does not directly train on ImageNet-C. Therefore, the authors suggest to use this dataset only to benchmark and validate already trained image classifiers and to not directly train on the dataset.

2.2. Artificial Neural Networks

After the previous section introduced the different datasets used in this work, it now remains to introduce the machine learning algorithms used to solve the image classification tasks. These algorithms are all based on artificial neural networks.

In the middle of the past century, researchers started to build artificial devices simulating the previously observed behavior of biological neurons [69, 80]. Essentially, these artificial neurons are functions of the form $g\colon \mathbb{R}^n \mapsto \mathbb{R}$, that map n input signals to one output signal. To calculate their output, they internally sum up the signals and process it with a non-linear activation function - this mimics in an abstract way the most basic functions of an actual biological neuron [69]. In the

1960s, complex networks of artificial neurons - Artificial Neural Networks (ANNs) - were also studied for the first time [50], but failed to gain long-lasting popularity. In recent years, ANNs have gained popularity again with the increased availability of computing power [78] but developed further away from the original biological inspiration. This section gives an introduction to ANNs and their notation; a more detailed description can be found in [34].

An ANN $N\colon I \mapsto T$ is a non-linear function, which maps an input space I to a target space T. In most cases it can be expressed as a concatenation $N = f_1 \circ f_2 \circ \ldots \circ f_n$ of non-linear functions f_i. Because of the sequential structure, these functions are often called 'layers' of the network. Every layer f_i is described using a set of parameters $\boldsymbol{\theta_i}$ controlling the function's behavior. The concatenation $\boldsymbol{\theta}$ of all these parameters controls the network's behavior and has to be adjusted such that the network is able to fulfill specific tasks. Therefore, the parameters $\boldsymbol{\theta}$ are often referred to as the learnable parameters of a network. To simplify the notation, in the following the parameters of layers or networks are not explicitly mentioned anymore unless required. In the next section, some of the most important types of network layers will be introduced [35].

In the case of an image classification task, the target of the network is to assign a class label to each input image. The input space $I = \mathbb{R}^{w\times h\times c}$ is the space of images with a width w, height h and number of color channels c (1 for grayscale and 3 for color images). The target space T the discrete set of k different classes $T = \{1, 2, \ldots, k\}$ [59]. To simplify notation, in the remainder of this work, it is always assumed that the output of the image classification network N can be rewritten as $N(\mathbf{x}) = \arg\max_y N_p(\mathbf{x})_y$. Here, the function $N_p(\mathbf{x})\colon I \mapsto [0,1]^k$ refers to the learned multinomial probability distribution $p(y|\mathbf{x})$ which assigns to each image probability values indicating the class membership for all classes $y \in T$. Furthermore, it is assumed that the multinomial distribution N_p can be expressed as

$$N_p(\mathbf{x}) = \text{softmax}(N_l(\mathbf{x})) = \left(\frac{e^{N_l(\mathbf{x})_i}}{\sum_j e^{N_l(\mathbf{x})_j}}\right)_{i=1,\ldots k}, \tag{2.1}$$

where the expression $N_l(\mathbf{x})_i$ refers to the i-th element of the logit values, which in practice are often the output of the penultimate layer of the neural network.

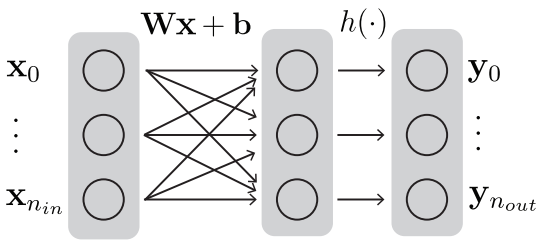


Figure 2.6: Schematics of a fully connected layer. The input $\mathbf{x} \in \mathbb{R}^{n_{in}}$ on the left side is processed by an affine transformation $\mathbf{Wx} + \mathbf{b}$ before a non-linear activation function $h(\cdot)$ is applied to calculate the final output $\mathbf{y} \in \mathbb{R}^{n_{out}}$.

2.2.1. Building Blocks

Fully Connected Layer A fully connected layer is the most fundamental part of an ANN and very close to the first description of ANNs [80]. Such a layer can be seen as a set of artificial neurons which processes a multi-dimensional input signal and produces a multi-dimensional output. Mathematically, it can be expressed as

$$\begin{aligned} f &\colon \mathbb{R}^{n_{in}} \mapsto \mathbb{R}^{n_{out}} \\ f(\mathbf{x}) &= h\left(\mathbf{Wx} + \mathbf{b}\right). \end{aligned} \tag{2.2}$$

Here, $\mathbf{W} \in \mathbb{R}^{n_{out} \times n_{in}}$ denotes the weight matrix that controls which input signals have an effect on which output signal and $\mathbf{b}$ is a constant bias added to the internal state independent of the input signal. This procedure is schematically shown in Figure 2.6. The internal state is finally used to calculate the output of the fully connected layer f using a (non-)linear activation function h. Both, $\mathbf{W}$ and $\mathbf{b}$ are learnable parameters of this layer type. The activation function is always applied elementwise, i.e. $h(\mathbf{x}) = \left(\hat{h}(\mathbf{x}_0), \ldots, \hat{h}(\mathbf{x}_n)\right)$. The two most often used functions are the identity $\hat{h}(x) = x$ and the ReLU function $\hat{h}(x) = \max\{x, 0\}$ [34]. Fully connected layers process vectors of real number; in many situations, one wants to apply such a layer to data with a multi-dimensional structure such as gray-scale images (2-D) or color-images (3-D). To apply a fully connected layer on these, one reshapes the data and reinterprets it as a multi-dimensional vector.

Convolutional Layer Even though fully connected layers already have a high predictive performance [20, 47], they also have some drawbacks. Namely, the number of parameters is larger than $n_{in} \cdot n_{out}$, which can easily result in millions of parameters for large input data, e.g. images [59]. To solve this issue, Rumelhart et al. [82] came up with the concept of parameter sharing. Here, weights inside a layer are shared across multiple connections, i.e. different connections between input and output are described by the same parameters.

LeCun et al. [58] noticed that weight sharing does not only decrease the number of parameters in a layer but also induces information about the geometry of the task: by sharing weights the same feature can be detected at multiple locations of the input data. This location invariance is especially important for image classification tasks as the class label does not depend on the location of an object but just on its presence. They used a type of weight sharing in which small patches of the input

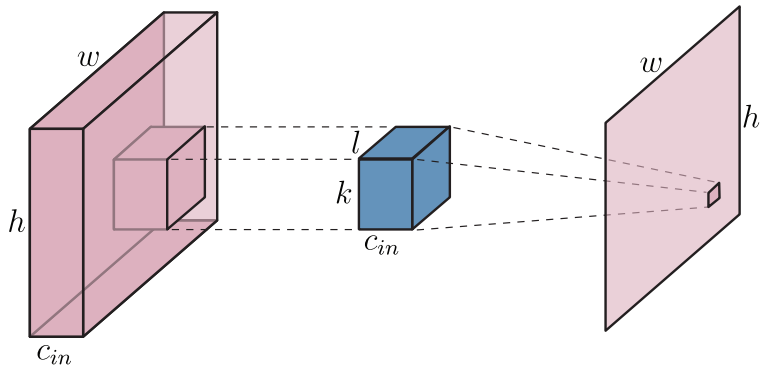


Figure 2.7.: Illustration of how the output of a convolutional layer is calculated. For each of the output channels (right plane) one 3-D kernel (blue) $\mathbf{K} \in \mathbb{R}^{c_{in} \times k \times l}$ is used. This kernel is moved across the input (left box) $\mathbf{x} \in \mathbb{R}^{w \times h \times c_{in}}$ and multiplicated elementwise with different patches of the input data. The result is given by the sum over the elementwise product. The procedure displayed is repeated for each of the output channels c_{in}.

image are used to compute the output of a layer. This kind of weight sharing is nowadays known as a convolutional layer; its function is shown in Figure 2.7. The usage of local patches of the image can be expressed using the notation of the 2-D convolution operation as

$$f \colon \mathbb{R}^{w \times h \times c_{in}} \mapsto \mathbb{R}^{w \times h \times c_{out}}$$
$$f(\mathbf{X}) = \left(\sum_{l,m,n} \mathbf{K}_{i,l,m,n} \cdot \mathbf{X}_{j+m-1,k+n-1,l} \right)_{i,j,k} = h(\mathbf{K} * \mathbf{X} + \mathbf{b}), \tag{2.3}$$

where $\mathbf{K}$ is the so-called convolutional kernel, $\mathbf{b}$ a bias, h a non-linear activation function and $*$ denotes a 2-D convolution. The kernel $\mathbf{K}$ is a 4-D tensor, whose element $\mathbf{K}_{i,j,k,l}$ describes the connection strength between a unit in a channel i of the input to a unit in channel j of the output which have an offset of up to k rows and l columns [35]. ANNs using convolutional layers are called Convolutional Neural

Networks (CNNs).

Pooling Layer In a pooling layer, the activations of the previous layer at some location are replaced by summary statistics of the activations in their local neighborhood [34]. Examples for such summary statistics are max pooling and average pooling, which use the maximum activation [101] and the average activation [11] in a rectangular neighborhood respectively. Including a pooling layer in a network serves two purposes: this can be seen as another type of parameter sharing, as in the following layers the exact same operations will be applied to all items which were included in the local neighborhood. This effectively reduces the number of learnable parameters in the network and, therefore, also reduces the computational cost. Additionally, using such local summary statistics enforces the learned representations to become approximately invariant to small translations of the input signal, which is also an often desired property of CNNs as described in the previous paragraph [34].

Batch Normalization Layer Modern ANNs are often composed of numerous layers of functions. This causes unintentional effects during the training procedure (Section 2.2.2). In particular, most algorithms for training an ANN calculate how one parameter of the network should be adjusted such that the performance of the network is increased, assuming that all other parameters are held constant [34]. However, in practice, one does not update the parameters in turns as this would increase the training time drastically and make training a deep network unfeasible. Instead, all parameters are updated at the same time, which can cause side-effects in which parameters are updated too much or too little. This effect can also be amplified by the different magnitudes of the activations of different layers. To solve this issue and to stabilize the training of deep networks Ioffe and Szegedy [49] proposed to normalize the activations of each layer before they are passed to the next layer such that all have a similar standard deviation and mean across image batches [34]. For this, they suggest inserting a batch normalization layer before every non-linear activation function in a network. The output of such a layer for n input samples $\mathbf{x}_{i=1,\dots n}$ is then computed as

$$f(\mathbf{x}) = \gamma\hat{\mathbf{x}} + \beta, \tag{2.4}$$

where γ and β are learnable parameters [49] and $\hat{\mathbf{x}}$ is a normalized version of the input, given by

$$\begin{aligned}\hat{\mathbf{x}} &= \frac{\mathbf{x} - \boldsymbol{\mu}}{\sqrt{\boldsymbol{\sigma}^2 + \epsilon}} \\ \boldsymbol{\mu} &= \frac{1}{n}\sum_{i}^{n} \mathbf{x}_i \\ \boldsymbol{\sigma}^2 &= \frac{1}{n}\sum_{i}^{n} (\mathbf{x}_i - \boldsymbol{\mu}^2).\end{aligned}$$

Inserting batch normalization layers into a network can reduce the risk of a stagnating training as it makes sure that the activation function is mostly evaluated in regimes where the values are not constant, i.e. the gradients do not vanish [49].

2.2.2. Training

While there exist different ways of training an ANN [34], in this work only the technique of supervised learning will be used. Supervised learning describes how to modify the learnable parameters of the network such that the output of the network models a target function f^* as well as possible. This strategy can be divided into two stages: first, one has to collect a dataset $\mathcal{D}$ with samples $(\mathbf{x}, f^*(\mathbf{x}))$. If f^* is an image classification task this refers to collecting images and annotating them with the correct label. Second, the parameters of the network are modified to increase the quality of the prediction, i.e. $N(\mathbf{x})$ should be as close as possible to the ground truth $f^*(\mathbf{x})$ for all images in the dataset. This scheme is also known as the maximum likelihood principle, which aims to decrease the dissimilarity between the true data distribution $p_{\mathcal{D}}$ and the data distribution approximated by the network N_p. This dissimilarity is measured by a loss function $\mathcal{L}$. For a classification task, in which the network predicts a multinomial distribution (Section 2.2), one often uses the multinomial cross-entropy loss function [34]

$$\mathcal{L} = \mathop{\mathbb{E}}_{\mathbf{x},y\sim\mathcal{D}} \left[\left(\log \left(N_p(\mathbf{x}) \right)_y \right) \right]. \tag{2.5}$$

To increase the ability of the network to generalize better to unseen data, one often includes an additional term which regularizes the parameters of the network [34],

e.g. the l_2 norm of the parameters:

$$\mathcal{L} = \mathop{\mathbb{E}}_{\mathbf{x},y\sim\mathcal{D}}\left[\left(\log\left(N_p(\mathbf{x})\right)_y\right)\right] + \sum_i ||\boldsymbol{\theta}_i||_2^2. \tag{2.6}$$

To actually find the parameter values which minimize the loss function,

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \ \mathcal{L} \tag{2.7}$$

one mostly uses gradient-based optimization techniques to solve the optimization problem [35]. These methods iteratively update the parameters using information about the gradient of the loss function that is calculated using backpropagation of the error-signal [81, 82].

To illustrate the general functionality of those optimizers, the Stochastic Gradient Descent (SGD) optimizer will be presented shortly. Gradient Descent changes the parameters of a network in an iterative fashion in the negative direction of the gradient of the loss function with respect to the parameters [34, 81, 82], i.e.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \lambda \nabla_{\boldsymbol{\theta}} \mathcal{L}, \tag{2.8}$$

with λ being the learning rate of the optimizer. SGD extends this by calculating the loss function not on the entire dataset but only on small mini-batches of the data [34]. These mini-batches only contain m randomly selected samples of the dataset. For a classification task, in which the cross-entropy introduced in Equation (2.5) is used, this means

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \lambda \nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_i^m (\log N_p(\mathbf{x}_i))_{y_i} \right). \tag{2.9}$$

2.2.3. Bayesian Neural Networks

In the previously described formulation of ANNs the network was expressed by a set of parameters which were adjusted to approximate a target function. An extension of this formulation are Bayesian Neural Networks (BNNs) [4, 8, 75]. In a BNN all parameters are represented by a probability distribution over possible values of the parameters rather than using a single fixed value as it is practice in normal ANNs [8]. This modification is said to encourage the network to learn more robust features [8].

Mathematically, in a BNN the posterior distribution over the parameters $p(\boldsymbol{\theta}|\mathcal{D})$ given the training dataset $\mathcal{D}$ is modeled. To make predictions about an input, the expectation over this distribution is used; the prediction of the network N for an input $\mathbf{x}$ is given by $\mathbb{E}_{\boldsymbol{\theta}\sim p(\boldsymbol{\theta}|\mathcal{D})}\left[N_{\boldsymbol{\theta}}(\mathbf{x})\right]$. In practice, this expectation value is often approximated with the average value across a finite number of predictions based on sampled realizations $\boldsymbol{\theta} \sim p(\boldsymbol{\theta}|\mathcal{D})$ [8, 61].

2.2.4. Common Network Architectures

The focus of this work is not to create new architectures or to improve existing approaches for image classification tasks, but rather to develop novel methods to analyze the robustness of these. This will both be done for different datasets and different networks. In this section a brief overview over the four analyzed classification architectures will be given. The first two have been created to perform well on the MNIST dataset, while the other two have been developed for more complicated datasets, such as CIFAR-10 or ImageNet.

C & W-MNIST The first network architecture will be called C & W-MNIST (Figure 2.8) and has been used in previous studies by Carlini and Wagner [15] to classify MNIST images. The network is a simple CNN, which first extracts image features using convolutional layers before three fully connected layers predict the class probability for each of the 10 classes in the dataset. On clean MNIST images the network used here has a performance of 93.4 %.

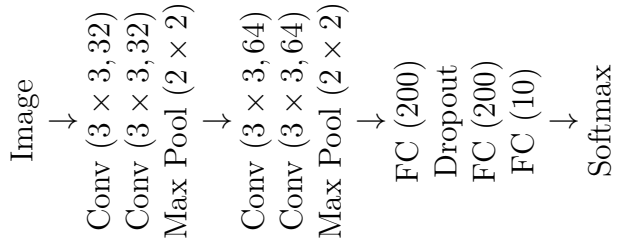


Figure 2.8.: Illustration of the structure of the C & W MNIST model introduced in [15]. Every convolutional (Conv) and fully connected (FC) layer (except the last one) is followed by a ReLU activation function. For fully connected layers the number in braces corresponds to the number of neurons in the layer, while for convolutional layers they correspond to the size of the convolutional kernel and the number of filters.

Madry-MNIST The second network used for the MNIST dataset will be called Madry-MNIST. It has been created by Madry et al. [65] (Figure 2.9). Even though the architecture is rather simple, the network is still able to perform well on MNIST with an accuracy of 99.2 %. While the architecture of the Madry-MNIST network is rather standard, the way it was trained is very interesting: during training the authors included adversarial perturbations (Section 2.3) in the training dataset which yielded a network that has a significantly increased robustness against perturbations of the images during test time.

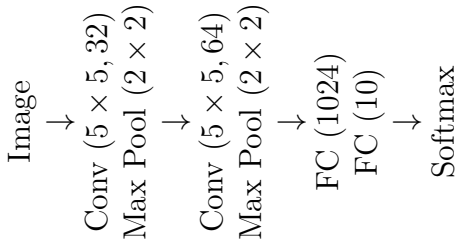


Figure 2.9.: Illustration of the structure of the Madry MNIST model used in [65], using the same notation as in Figure 2.9.

VGG The convolutional networks proposed by the Visual Geometry Group (VGG) were one of the first architectures which replaced convolutional layers that have large kernel sizes with multiple stacked layers that use smaller kernels [90]. This makes the network thinner and deeper compared to other popular networks for large scale image datasets at that time (cf. [57]). One of the networks they proposed, the VGG-16, is illustrated in Figure 2.10. This network consists of 13 convolutional and

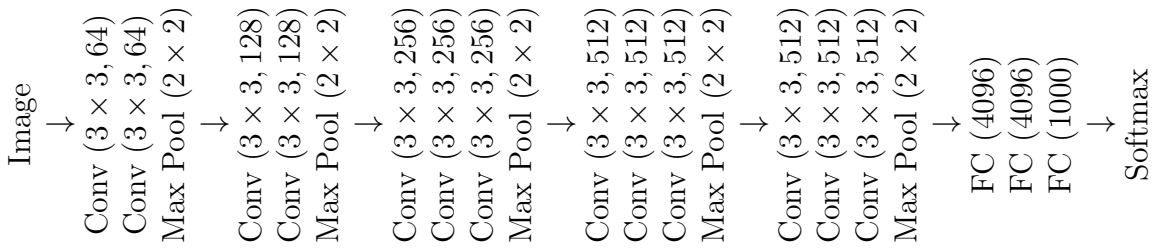


Figure 2.10.: Illustration of the structure of the VGG-16 network introduced in [90], using the same notation as in Figure 2.9.

3 fully connected layers. There is also a larger version of the network, the VGG-19,

which contains three additional convolutional layers. The VGG-16 network used in the context of this work has an accuracy of 71.6 % on ImageNet, while the larger variant of the network, the VGG-19, achieves an accuracy of 72.4 %.

ResNet The last type of networks used in this work is the residual network (ResNet) by He et al. [43]. The core motivation behind these network architectures is preventing the vanishing gradient effect. This effect can be observed in very deep networks: the deeper a network becomes, the weaker the gradient signal becomes. As the signal is used to update the network's weights, the vanishing gradient effect makes the training inefficient. To solve the problem, He et al. [44] proposed to add so-called residual connections to the network. An example of how such a residual connection can be used is shown in Figure 2.11. The residual connections allow the input to skip some layers of the network, which effectively reduces the depth of the network while maintaining its predictive power. To ease notation, one often summarizes multiple layers of a network which are connected by such a residual connection as one building block. In Figure 2.11 the so-called Bottleneck building block is shown, which is used in ResNets. The idea behind the residual connections is that it is

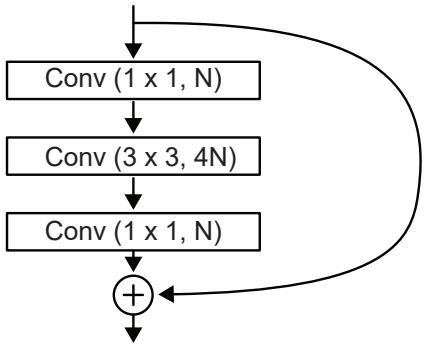


Figure 2.11: Illustration of the Bottleneck building block with N channels used in ResNet architectures [43]. The same notation as in Figure 2.9 has been used, except that after each convolutional layer an additional batch normalization layer is used. Modified according to [43].

easier for one layer to predict the residual (i.e. how an input should change) than directly predicting the output. This means, that one re-writes the transformation done by a layer as $f(\mathbf{x}) = \mathbf{x} + g(\mathbf{x})$, whereby the latter part corresponds to the residual.

Based on the Bottleneck block, He et al. [43] developed the family of ResNets which were the first networks to have many dozens of layers while still staying trainable. They proposed different networks (e.g. ResNet-18, Resnet-50, ResNet-152) which only differ in the number of layers, i.e. the number of Bottleneck blocks used. In Figure 2.12 the structure of the ResNet-50 is displayed; the other types

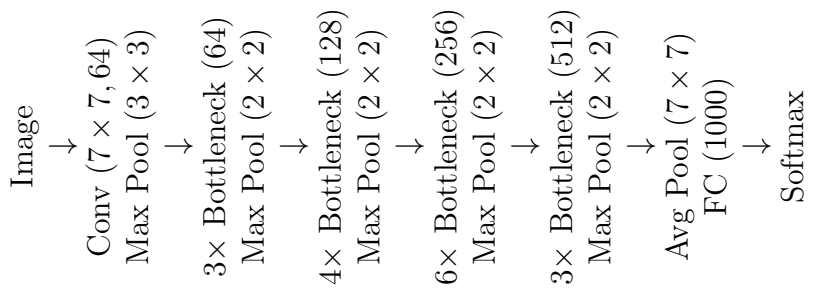


Figure 2.12.: Illustration of the structure of the ResNet-50 model introduced in [43, 44], using the same notation as in Figure 2.9. For Bottleneck blocks the number in brackets corresponds to the number of filters used in the first layer of them. Repetitions of the same building block (which do not share their parameters) are indicated by prepending an $x\times$ to them.

of ResNets look very similar but contain a different number of building blocks. The networks are often used in various computer vision tasks, as they yield good performance on numerous datasets. On ImageNet a ResNet-50 achieves an accuracy of 76.2 %, while a ResNet-152 performs better with an accuracy of 78.3 %. On CIFAR-10 a variant of the ResNet-18 used in this work has a performance of 93.7 %.

2.3. Adversarial Perturbations

With the increased popularity of deep neural networks, researchers started to become not only interested in their performance but also to analyze their security [6, 94]. Here, the phenomenon of adversarial perturbations was observed: a small change to the input of a network can change its output completely. In the following, only the case of an image classification problem will be analyzed. This means that a small modification of the input image changes the predicted class. There are two sub-types of adversarial attacks - targeted and untargeted attacks. Given an image classification network $N\colon I \mapsto C$ (Section 2.2), an original image $\mathbf{x}$ and its ground-truth label y, the untargeted attack looks for visually similar examples $\mathbf{x}'$ such that $y = N(\mathbf{x}) \neq N(\mathbf{x}')$. This can also expressed as

$$\mathbf{x}' = \arg\min_{\mathbf{x}'} d(\mathbf{x}, \mathbf{x}') \quad \text{s.t.} \quad N(\mathbf{x}') \neq N(\mathbf{x}), \tag{2.10}$$

where $d\colon I \times I \mapsto \mathbb{R}$ is a metric to measure the visual difference between the original and adversarial image, e.g. an ℓ_p norm. In the case of a targeted attack, the aim of the attacker is to change the network's prediction not to a random but to a specific target class t: find an image $\mathbf{x}'$ such that $\mathbf{x}$ and $\mathbf{x}'$ are close and $N(\mathbf{x}') = t \neq N(\mathbf{x})$. Using the previously introduced distance function d, this can be expressed as

$$\mathbf{x}' = \arg\min_{\mathbf{x}'} d(\mathbf{x}, \mathbf{x}') \quad \text{s.t.} \quad N(\mathbf{x}') = t. \tag{2.11}$$

In the context of adversarial examples, one can define the robustness of a neural network as the minimal required distance $d(\mathbf{x}, \mathbf{x}')$ between original and adversarial example to manipulate the network's decision. Higher robustness means, the network becomes less sensitive to changes of the input. To quantify the robustness of an image classifier across an entire dataset $\mathcal{D}$, one often measures the median distance between clean images and minimal adversarial perturbations [87],

$$\epsilon^* = \operatorname*{median}_{\mathbf{x}\sim\mathcal{D}} \{\min d(\mathbf{x}, \mathbf{x}') \quad \text{s.t. } \mathbf{x}' \text{ is adversarial}\}. \tag{2.12}$$

To improve the robustness against adversarial examples, recently many approaches have been suggested [61, 62, 66, 87]. While most of them turned out to not increase the adversarial robustness significantly [14, 52], there is one simple but very efficient approach that proved itself over time: adversarial training [35, 65]. In this defense strategy, the training mechanism of the network is adjusted: during training, one repeatedly calculates adversarial examples for the current state of the network and trains the network on these, such that the network learns to recognize them, i.e. becomes more robust [35, 65].

In the remainder of this section, two popular types of adversarial attacks will be introduced. For both of the attacks there exists a targeted and an untargeted version which have a very similar form; for the sake of brevity only the untargeted version will be shown here.

2.3.1. Projected Gradient Descent Attack

The untargeted Projected Gradient Descent (PGD) was one of the first adversarial attacks [65]. To find the example with the minimal distance to the original image $d(\mathbf{x}, \mathbf{x}')$ using PGD, one replaces the exact minimization of the distance in Equation (2.10) with an approximated binary search: find the minimal value ϵ such

that

$$\exists \mathbf{x}' : N(\mathbf{x}') \neq N(\mathbf{x}) \wedge d(\mathbf{x}, \mathbf{x}') < \epsilon \tag{2.13}$$

is fulfilled. While during the training of an ANN the loss is minimized with respect to the network's parameters $\boldsymbol{\theta}$, PGD performs a gradient ascent to maximize the loss with respect to the input $\mathbf{x}$ for fixed parameters $\boldsymbol{\theta}$. By increasing the loss function the attack changes the predicted probability values N_p until the prediction $N(\mathbf{x}') = \arg\max_y N_p(\mathbf{x}')_y$ is changed such that the adversarial condition is fulfilled. To make sure the second condition of Equation (2.13) is met, after each gradient ascent step the image is projected back to the feasible set $\mathbf{x}' \in I \quad \wedge \quad d(\mathbf{x}, \mathbf{x}') < \epsilon$ using the projection operator $\prod_{\ell_2(\mathbf{x},\mathbf{x}')\leq\epsilon}$. Combining these insights, the resulting iterative attack procedure has the form

$$\mathbf{x}'_{t+1} = \prod_{\ell_2(\mathbf{x},\mathbf{x}')\leq\epsilon} \left[\mathbf{x}_t + \alpha \nabla_{\mathbf{x}} \mathcal{L}\left(N_p(\mathbf{x}), y\right)\Big|_{\mathbf{x}=\mathbf{x}'_t} \right], \tag{2.14}$$

in the case of the distance to minimize being the ℓ_2 norm. If instead the ℓ_∞ norm is used, the procedure changes to

$$\mathbf{x}'_{t+1} = \prod_{\ell_\infty(\mathbf{x},\mathbf{x}')\leq\epsilon} \left[\mathbf{x}_t + \alpha \operatorname{sgn}\left(\nabla_{\mathbf{x}} \mathcal{L}(N_p(\mathbf{x}), y)\Big|_{\mathbf{x}=\mathbf{x}'_t} \right) \right], \tag{2.15}$$

where sgn corresponds to the signum operation.

2.3.2. Carlini & Wagner Attack

The Carlini & Wagner attack (C & W) is an adversarial attack trying to include the minimization of the adversarial image's ℓ_2 norm directly in the objective. Originally Carlini and Wagner [14] proposed it as a targeted attack but with a small modification, one can also use it as an untargeted attack. In the following, only the modified untargeted version will be shown and used. In this attack, the adversarial example is not directly searched in the image space but rather in a transformed search space, as this improves the convergence of the numerical optimization. The transformation is given by the non-linear function $h(\mathbf{s}) = 0.5 \tanh(\mathbf{s} + 1)$ [14]. Instead of solving the optimization problem given in Equation (2.10), the attack optimizes the dual

problem

$$\underset{\mathbf{s}}{\text{minimize}} \, \|h(\mathbf{s})\|_2^2 + c \cdot f\left(h(\mathbf{s}) + \mathbf{x}\right) \tag{2.16}$$

$$\text{with} \quad f\left(\mathbf{x}'\right) = \max\left(N_l\left(\mathbf{x}'\right)_y - \max\left\{N_l\left(\mathbf{x}'\right)_i : i \neq y\right\}, -\kappa\right), \tag{2.17}$$

where κ and c are hyperparameters of the attack and the function f measures the difference between the predicted logit values for the ground-truth label y and the highest other predicted logit. The goal becomes to minimize this difference. Once it is a non-positive value, an untargeted adversarial example has been found, as then there exists at least one class for which the predicted probability is larger than for the ground-truth class [14].

2.4. Evolution Strategies

The mathematical area of optimization studies methods to find optimal solutions to a problem. These problems are often expressed using an objective function

$$f \colon \mathbb{R}^n \to \mathbb{R} \tag{2.18}$$

$$\mathbf{x} \mapsto f(\mathbf{x}), \tag{2.19}$$

which maps the optimization parameters x from a high-dimensional space to a scalar representing their 'value' [12]. For the ease of clarity, in the following it is always assumed that one is interested in finding the minimum of the objective:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} f(\mathbf{x}). \tag{2.20}$$

For this, the most popular class of algorithms are gradient-based optimization methods [12]. These algorithms require the objective f to be at least once differentiable and use information about the gradient to find the minima of the function. If one has no knowledge about the gradients of a function or if the function is not differentiable, the problem is called a black-box optimization problem. Here, the only information accessible about the objective are its function values [38].

One popular class of black-box optimization algorithms are Evolution Strategies (ES), which are search procedures inspired by biological evolution. They are iterative procedures and follow a similar pattern [85]: In every iteration g ('generation') a new population of λ different parameters $x_{i=1,\dots,\lambda}$ ('genotypes') is generated based

on the previous generation by perturbing ('mutating') their parameters. Then, for every member of the population the objective function's value ('fitness') is evaluated. The better the score of a member of the population is, the more it will influence the members of the next generation [85]. The parameter λ controls the size of the population. This procedure is displayed in Algorithm 2.1. Here, the members of the next generation are created by sampling from a parameterized distribution $p(x|\boldsymbol{\theta}^i)$. The parameters θ^i of the distribution are adjusted according to the fitness values for every member of the population using an update function F [38].

Algorithm 2.1: Prototypical Evolution Strategies [38]

initialize distribution parameters $\boldsymbol{\theta}^0$;
for *generation* $i = 0, 1, 2, \ldots$ **do**
    sample population from distribution $\mathbf{x}_1, \ldots, \mathbf{x}_\lambda \sim p(\mathbf{x}|\theta^i)$;
    evaluate f for every member $\mathbf{x}_1, \ldots, \mathbf{x}_\lambda$;
    update distribution parameters $\boldsymbol{\theta}^{i+1} \leftarrow F\left(\boldsymbol{\theta}^i, (\mathbf{x}_1, f(\mathbf{x}_1)), \ldots, (\mathbf{x}_\lambda, f(\mathbf{x}_\lambda)\right)$;

The majority of ES implementing the generic Algorithm 2.1 model the population distribution $p(\mathbf{x}|\boldsymbol{\theta})$ as a Gaussian distribution $\mathcal{N}(\boldsymbol{m}, \boldsymbol{\Sigma}^2)$. For this class of algorithms, the update function F updates the mean $\boldsymbol{m}$ and variance $\boldsymbol{\Sigma}^2$ of the distribution [39, 40]. In this work, the most successful variant of such an algorithm will be used; it will be introduced in the following section.

2.4.1. Covariance Matrix Adaptation Evolution Strategy

The Covariance Matrix Adaptation ES (CMA-ES) is a popular algorithm because of its strong performance and versatility [36–40]. It was developed with the aim of improving the convergence speed of the optimization process compared to previous ES. This improvement of convergence comes with a slightly increased computational cost. Most other ES algorithms use a simplified description for the variance of the population distribution and approximate it using the unit matrix or a diagonal matrix [36, 85]. CMA-ES improves previous search algorithms by adjusting not only the variance of the Gaussian search distribution but also its covariance. This allows the algorithm to use different scales to search in different dimensions. Therefore, CMA-ES is known to better scale to various coordinate systems, which do not need to be equidistant for all dimensions [37–40]. The following introduction to CMA-ES is based on the work of Hansen [38].

The search distribution $p(\mathbf{x}|\boldsymbol{\theta})$ utilized in CMA-ES is a Gaussian distribution

$$p(x_j|\mathbf{m}^i) = \mathcal{N}\left(\mathbf{m}^i, (\sigma^i)^2\hat{\mathbf{C}}^i\right), \tag{2.21}$$

where $\mathbf{m}^i$ is the mean value of the search distribution and $(\sigma^i)^2\hat{\mathbf{C}}^i$ is its covariance.

As the search distribution has two parameter vectors, the update function F in Algorithm 2.1 can be split into two updates: one for the mean $\mathbf{m}^i$ and one for the covariance matrix $\hat{\mathbf{C}}^i$.

The mean of the search distribution is updated according to a weighted average over the μ fittest population members:

$$\boldsymbol{m}^{i+1} = \boldsymbol{m}^i + c_\mathrm{m}\sum_{j=1}^{\mu} w_j\left(\boldsymbol{x}_{j:\lambda}^{i+1} - \boldsymbol{m}^i\right). \tag{2.22}$$

Here, λ again represents the population size and μ controls how many of members of the current generation are used to calculate the parameters update. The notation $\mathbf{x}_{j:\lambda}^i$ means the j-th best individual of the population in terms of its fitness and w_j are weighting factors which are often uniformly set to $w_j = 1/\mu$.

In the second step, the covariance matrix $\hat{\mathbf{C}}^i$ is updated according to

$$\hat{\mathbf{C}}^{i+1} = \left(1 - c_1 - c_\mu\sum w_j\right)\hat{\mathbf{C}} + c_1\mathbf{p}_c^{i+1}\left(\mathbf{p}_c^{i+1}\right)^T + c_\mu\sum_{j}^{\lambda} w_j\mathbf{y}_{j:\lambda}^{i+1}\left(\mathbf{y}_{j:\,\lambda}^{i+1}\right)^T, \tag{2.23}$$

where $\mathbf{p}_c^i \in \mathbb{R}^n$ is the evolution path, i.e. an exponentially smoothed trace of the distribution's mean $\mathbf{m}^i$. The expression $\mathbf{y}_{j:\lambda}^i$ corresponds to the scaled difference between the population member $\mathbf{x}_{j:\lambda}^i$ and the distributions mean:

$$\mathbf{y}_{j:\lambda}^i = \frac{\mathbf{x}_{j:\lambda}^i - \mathbf{m}^i}{\sigma^i}.$$

Because of the adjustment of the covariance matrix, CMA-ES is known to be very stable and sample efficient across different domains and objectives [3, 37].

2.5. Markov Chain Monte Carlo

In the fields of statistical mechanics and material science, researchers are often interested in macroscopic properties of a system, such as the density distribution. To compute them numerically, one has to compute an expected value over all possible

microscopic realizations of the macroscopic state [5, 70]:

$$F = \mathop{\mathbb{E}}_{s \in S}[f(s)] = \int_S \pi(s) f(s) ds, \tag{2.24}$$

whereby s is a microscopic realization, S the macroscopic state, $f(s)$ the property one wants to calculate and $\pi(s)$ the probability distribution of the realizations. As the analytical solution of the integral might not be trivial, one can use a Monte Carlo method to compute it. For this, the integral is replaced with a finite sum computed over states sampled from the probability distribution:

$$F \approx \frac{1}{N} \sum_i^N f(s_i), \quad s_i \sim \pi(s). \tag{2.25}$$

In general, the true distribution $\pi(s)$ might have a complicated form which makes sampling from it hard. One way to bypass this, is by generating samples from a uniform distribution over the allowed space of states and replacing the sum with a weighted sum, whereby the weight of each sample corresponds to its probability. Even though this method is able to approximate the exact value, it is cumbersome as one often generates samples with a low probability which only contribute little to the sum.

An alternative way to solve this issue was proposed by Metropolis et al. [70] in 1953. They suggested a way how to sample efficiently from $\pi(s)$ by utilizing a Markov Chain that has the true distribution $\pi(s)$ as its stationary distribution. Thus, by sampling from the easier to compute Markov Chain, one can sample from π and therefore calculate the expectation value efficiently [53]. A Markov Chain is a sequence of random variables $s_0, s_1, s_2, \ldots$ so that for every index i, the variable s_i only depends on the previous variable s_{i-1}. This means, that s_i can be sampled from a probability distribution $p\,(s_i|s_{i-1})$. The probability distribution p therefore fully describes the transition probability from s_{i-1} to s_i and is called the transition kernel. If the values s_i of the chain do not depend anymore on the initial state s_0, the chain has reached its stationary distribution [53].

Now the question arises, how one can sample transitions from the transition kernel. For this, a so-called acceptance function $\alpha(s_i, r)$ and a proposal distribution $q(r|s_i)$ are introduced. Whenever a new state shall be sampled from the transition kernel $p(s_i|s_{i-1})$, a two-staged method is used: first, a proposal r is sampled from the proposal distribution $q(r|s_i)$ depending on the current state s_i. Next, the proposal is

accepted to become the new state s_{i+1} with a probability defined by the acceptance function $\alpha(s_i, r)$. If the proposal is rejected, the chain does not change, i.e. the current state stays and becomes the new state: $s_{i+1} = s_i$ [53, 86].

Whether the chain actually reaches its stationary case depends on the choice of the transition kernel, i.e. the choice of the proposal distribution and the acceptance function. There exist multiple ways how to choose α and q which guarantee convergence. In the following, only the method proposed by Barker [5] shall be summarized. The acceptance function introduced by Barker [5] has the form

$$\alpha(r, s) = \frac{\pi(r)}{\pi(r) + \pi(s)}, \tag{2.26}$$

whereby π is the probability distribution one wants to sample from. This acceptance function guarantees convergence as long as a symmetric proposal distribution q is used, i.e. $q(s|r) = q(r|s)$.

By inspecting the Barker acceptance function, one sees that once the chain has reached a state with high probability $\pi(s)$ it becomes very unlikely that a transition to another state r is performed. Therefore, one can also use the MCMC algorithm to maximize the probability, i.e. find $\arg\max \pi(s)$, by sampling long enough from the chain. Thus, MCMC can also be used as a black-box optimization method.

2.5.1. Markov Chain Monte Carlo with People

In psychophysics, an often used type of experiment is the so-called Two-Alternative Forced Choice (2-AFC) task in which two stimuli (e.g. images) are presented to a human subject. The subject is told that one of the stimuli comes from a target distribution and it has to choose the correct one. These tasks can be used to analyze the subjective experience of a test person. A possible scenario of a 2-AFC task is the following: two images $\mathbf{x}_1$ and $\mathbf{x}_2$ are presented to a human. The subject is now told that one of them corresponds to a specific class c and he has to chose the correct one. Repeating this experiment with multiple subjects gives an insight into which sample is more likely to be perceived as an image of class c.

Sanborn and Griffiths [86] suggest analyzing a 2-AFC task from a Bayesian perspective. In this 2-AFC task, two stimuli are presented to a human subject. One of these is drawn from class-dependent probability distribution $p(\mathbf{x}|c)$ of the true class c, while the other one is samples from a different distribution. The two possible choices of the subject now correspond to the two hypotheses H_1 and H_2. The first

hypothesis means that the image $\mathbf{x}_1$ is sampled from the true distribution $p(\mathbf{x}|c)$ and $\mathbf{x}_2$ is drawn from the alternative distribution. The second hypothesis is the other way around. By applying Bayes' rule one can calculate the posterior distribution

$$p(H_1|\mathbf{x}_1, \mathbf{x}_2) = \frac{p(\mathbf{x}_1, \mathbf{x}_2|H_1)p(H_1)}{p(\mathbf{x}_1, \mathbf{x}_2|H_1)p(H_1) + p(\mathbf{x}_1, \mathbf{x}_2|H_2)p(H_2)}. \tag{2.27}$$

As $\mathbf{x}_1$ and $\mathbf{x}_2$ are independent one can rewrite this expression using the category distribution $p(\mathbf{x}|c)$ and the alternative distribution $g(\mathbf{x})$ to get

$$p(H_1|\mathbf{x}_1, \mathbf{x}_2) = \frac{p(\mathbf{x}_1|c)g(\mathbf{x}_2)p(H_1)}{p(\mathbf{x}_1|c)g(\mathbf{x}_2)p(H_1) + p(\mathbf{x}_2|c)g(\mathbf{x}_1)p(h_2)}. \tag{2.28}$$

Assuming the two hypotheses have the same prior probabilities, i.e. $p(H_1) = p(H_2)$, and that the two stimuli have (approximately) the same probability in the alternative distribution, i.e. $g(\mathbf{x}_1) \approx g(\mathbf{x}_2)$ one can simplify this expression further to

$$p(H_1|\mathbf{x}_1, \mathbf{x}_2) = \frac{p(\mathbf{x}_1|c)}{p(\mathbf{x}_1|c) + p(\mathbf{x}_2|c)}. \tag{2.29}$$

This equation is known as the Luce choice rule [64, 88]. By substituting the probability $p(\mathbf{x}|c)$ in Equation (2.26) with $\pi(\mathbf{x})$, one gets the Barker acceptance function Equation (2.29) [86]. This means, that sampling from the Barker acceptance function can be realized as a 2-AFC experiment. Thus, the MCMC algorithm can be used to maximize the conditional probability $p(\mathbf{x}|c)$ of a stimulus $\mathbf{x}$ that human subjects associate with the category c. This variation of the MCMC algorithm is called Markov Chain Monte Carlo with People (MCMC-P) [86]. Several studies have already used MCMC-P to find the maxima of the probability distribution induced by subjects, e.g. finding happy/sad faces [67], finding heavy/light-appearing objects [18], rectangles close to the golden ratio [7] or stick-figure representations of animals [86]. A description of how to apply the MCMC-P algorithm can be found in Algorithm 2.2. In this algorithm, the binary decision function $d(\cdot, \cdot)$ corresponds to the human decision in the 2-AFC task. The stimuli presented in this 2-AFC task are generated by a function h, which transforms a set of parameters - which effectively are the subject of the optimization - to a stimulus that can be presented to a human. These parameters are also called *state*, as they represent the state of the Markov chain. By including the function h, one can perform a dimensionality reduction and, therefore, reduce the size of the search space, which results in an

Algorithm 2.2: Original MCMC-P.

Input: Binary decision function $d(\mathbf{x}_1, \mathbf{x}_2) \to \{1, 2\}$, State of original stimulus $\mathbf{s}_0 \in \mathbb{R}^n$, Number of steps N, Stimulus generating function $h \colon \mathbb{R}^n \mapsto \mathbb{S}$, Proposal distribution q

Result: $\mathbf{x}_N = \arg\max p\left(h(\mathbf{x})|c\right)$

Initialize state of Markov Chain $\mathbf{s}_0$;
for $t \leftarrow 1$ ***to*** N **do**
    // Get perturbation of state to get new proposal
    $\mathbf{p}_t \sim q(\mathbf{s}_{t-1})$
    // Generate stimuli from states
    $\mathbf{x}_c = h\left(\mathbf{s}_{t-1}\right)$
    $\mathbf{x}_p = h\left(\mathbf{p}_t\right)$
    // Compare stimulus of current state and of the proposed state
    if $d(\mathbf{x}_c, \mathbf{x}_p) = 1$ **then**
        // Keep current state
        $\mathbf{s}_t \leftarrow \mathbf{s}_{t-1}$
    else
        // Replace current state with proposal
        $\mathbf{s}_t \leftarrow \mathbf{p}_t$
    end
end
return $\mathbf{s}_N$

improved efficiency of the algorithm [7].

3. Analyzing the Robustness of Bayesian Neural Networks

In this chapter, the robustness of a defense mechanism proposed by Liu et al. [62] will be examined. The results of this evaluation have already been made public (see [103]) and were shared with the authors of the proposed method. In the original study, the authors combine the idea of making a neural network robust through adversarial training with the concept of Bayesian neural networks (Section 2.2.3). They call this combination Adv-BNN.

The BNN used in their work does not learn a point estimate of the weights to fulfill a task, but instead the parameters of a probability distribution, from which the actual weights are sampled at inference. During training, the parameters of this distribution are adjusted such that on the one hand, the network can solve the task with the sampled weights and on the other hand, the distribution does not deviate too much from a fixed prior distribution. This is implemented by utilizing an additional loss function that measures the Kullback-Leibler (KL) divergence between the learned and the prior distribution. Their notion of BNNs assumes independence between the different weights of the network. This means, that the probability distribution of all weights of the network can be separated into a product of individual one-dimensional distributions. For these one-dimensional distributions, Gaussian distributions are used to parameterize the learnable distribution and a Gaussian prior with a fixed standard deviation is assumed. This choice has been motivated by the analytical solution of the KL divergence which exists in this setting [46].

The authors claim that by applying the concept of adversarial training to such a BNN, one can efficiently lower the Lipschitz constant of the network [62]. The Lipschitz constant describes how much the value of a function, i.e. the output of a neural network, changes in proportion to small changes of the input. Such a reduced Lipschitz constant is associated with more robust networks that are less susceptible to adversarial examples [61, 62].

In the used BNN formalism, the network's predictions are not deterministic, but stochastic. This stochasticity can be expressed using a random vector $\boldsymbol{\xi}$ as an additional input to the network and then treating the network itself as being completely deterministic by applying the reparametrization trick [55]. Therefore, to obtain a prediction at test time Liu et al. [62] propose to use the expected value of the network's prediction over all the possible values for the random vector. As there exists no easy way to compute an analytical expression for this expected value, they approximate it as the mean over the finite set of n predictions obtained for different random vectors, i.e. multiple values of $\boldsymbol{\xi}$:

$$\mathop{\mathbb{E}}_{\boldsymbol{\xi}}\left[N(\mathbf{x};\boldsymbol{\xi}_i)\right] \approx \frac{1}{n}\sum_{i}^{N} N(\mathbf{x};\boldsymbol{\xi}_i). \tag{3.1}$$

In a popular definition of adversarial examples (Section 2.3), one looks for perturbed images $\mathbf{x}'$ which maximize the prediction loss $\mathcal{L}$ (e.g. cross-entropy) while the perturbation is constrained to be small $d(\mathbf{x},\mathbf{x}') < \epsilon$, where d is again a distance measure like an ℓ_p norm. This definition is for example used in the PGD attack (Section 2.3.1). For the BNN formalism used here, this definition needs to be adjusted, as the prediction and therefore the loss is not deterministic anymore. By including this into the definition Liu et al. [62] obtain the expression

$$\mathbf{x}' = \mathop{\arg\max}_{\substack{\mathbf{x}' \\ d(\mathbf{x},\mathbf{x}')<\epsilon}} \mathop{\mathbb{E}}_{\boldsymbol{\xi}}\left[\mathcal{L}(N(\mathbf{x};\boldsymbol{\xi}), y)\right]. \tag{3.2}$$

Next, they notice that one cannot use the usual PGD algorithm to find adversarial examples according to this definition, as one also has to take the stochasticity of the BNN into account. For this, the authors suggest to use a stochastic gradient ascent formulation of the usual PGD and come up with the modified update method

$$\mathbf{x}'_{t+1} \leftarrow \mathop{\Pi}_{d(\mathbf{x},\mathbf{x}')<\epsilon}\left[\mathbf{x}'_t + \eta\nabla_{\mathbf{x}} L\left(N(\boldsymbol{x};\boldsymbol{\xi}), y\right)\Big|_{\substack{\mathbf{x}=\boldsymbol{x}'_t \\ \boldsymbol{\xi}=\boldsymbol{\xi}_t}}\right], \tag{3.3}$$

for which they sample a new random vector $\boldsymbol{\xi}_t$ in each update step. Here, Π represents the projection onto the set of allowed perturbations, e.g. an ℓ_p-ball around $\mathbf{x}$. In the following, this attack method will be called naive PGD.

3.1. Adaptation of Adversarial Attacks

Similar to the predictions of the network its gradients are stochastic, too. Thus, the gradient obtained from a single inference can point into an unrepresentative direction which makes it harder for the gradient ascent to perform well and find a stable optimum.

To solve this issue, I propose to use a modified version of their naive PGD algorithm which comes closer to the original definition of the PGD attack. By estimating the real gradient of the network as the average of the gradients over multiple random vectors $\boldsymbol{\xi}$, one can obtain a more stable and therefore efficient attack

$$\mathbf{x}'_{t+1} \leftarrow \prod_{d(\mathbf{x},\mathbf{x}')<\epsilon} \left[\mathbf{x}'_t + \eta \mathop{\mathbb{E}}_{\boldsymbol{\xi}} \left(\nabla_{\mathbf{x}} L\left(f(\boldsymbol{x};\boldsymbol{\xi}), y\right)\Big|_{\mathbf{x}=\boldsymbol{x}'_t} \right) \right], \tag{3.4}$$

which will be called averaged PGD (A-PGD) in the following. The reasoning behind this attack is similar to the motivation behind the Expectation over Transformation (EOT) attack by Athalye et al. [2]: by averaging over the output of the network before the direction in which the adversarial example should be searched is calculated, one can reduce the risk of making unnecessary or inconsistent updates, thus, improving the overall efficiency of the attack. When the A-PGD attack is applied to a deterministic classifier, it behaves just like PGD.

3.2. Experiments & Results

To validate the claimed results about the robustness of the adversarially trained BNN proposed by Liu et al. [62], in this section the naive PGD will be applied to the BNN again to reproduce their results. Furthermore, the improved A-PGD attack is also applied to the model and its efficiency is compared to the naive PGD. For this, the network's weights released by the authors are used.[1] These are weights for the BNN variant of a VGG-16 network [90], which was adversarially trained on the STL-10 dataset (Section 2.1.3) with the native PGD attack. These weights have been trained to create a model that is robust against ℓ_∞ attacks; therefore, for a fair comparison, in this chapter only the ℓ_∞ variant of the attacks are used and evaluated. Even though the authors also released weights for the network trained on the CIFAR-10 and ImageNet dataset, those are not used as the authors admitted

[1] `https://github.com/xuanqing94/BayesianDefense`

they trained them incorrectly.[2] Furthermore, the robustness will be compared to a BNN which has been trained adversarially with the improved A-PGD variant as well as to a non-Bayesian adversarially trained VGG-16 network. The results of this comparison are displayed in Table 3.1.

		$\epsilon = 0.035$		$\epsilon = 0.07$	
Defense	Clean	Naive PGD	A-PGD	Naive PGD	A-PGD
Adv. Training	53.9 %	30.6 %	-	10.8 %	-
Adv-BNN	**59.9** %	(37.6 %)	30.3 %	(21.1 %)	8.6 %
Adv-BNN w/ A-PGD	53.2%	(38.2 %)	**31.2** %	(31.2 %)	**13.4** %

Table 3.1.: Accuracy for adversarial examples for three different defense methods on the STL-10 dataset: an adversarially trained standard VGG-16 (Adv. Training), the adversarially trained Bayesian variant proposed in [62] (Adv-BNN) and the Bayesian variant adversarially trained on A-PGD examples (Adv-BNN w/ A-PGD). The defended networks are tested on clean data and adversarial examples which are bounded by an ℓ_∞ ball of radius ϵ for $\epsilon = 0.035$ and $\epsilon = 0.07$. Accuracies against too weak adversarial attacks are shown in brackets. The highest accuracy for each of the three test scenarios is marked in bold.

Here, the robustness under adversarial attacks is evaluated for two ϵ radii, which were the ones used for the original evaluation of the BNN by Liu et al. [62]. In this table, there are two observations of special interest which hold for both radii. First, under attack of the naive PGD, their adversarially trained model does, in fact, perform better than the non-Bayesian adversarially trained model. But this improvement vanishes completely if the improved A-PGD attack is used to attack the model. Now, the accuracy is not higher but even slightly lower than the one of the baseline model. Second, if one trains their BNN architecture adversarially on perturbations found by the improved A-PGD attack, the model yields higher accuracy scores against both the naive PGD and A-PGD compared to their training. Furthermore, this accuracy is slightly higher than the accuracy of the baseline against adversarial examples found by PGD. This shows, that by using a suitable and sufficiently powerful adversarial attack, like A-PGD, the experimental results come closer to the theoretical derivations presented by Liu et al. [62] in terms of robustness. However, the improved robustness against adversarial examples is accompanied by a reduction of clean accuracy, which comes close to the accuracy of the adversarially trained deterministic model. A possible explanation for this obser-

[2] `https://github.com/xuanqing94/BayesianDefense/issues/5`

vation might be a too weak regularization of the network's weights in combination with the small size of the dataset.

3.3. Conclusion

This chapter showed once again that defense mechanisms against adversarial attacks are difficult to evaluate (cf. [14]). A wrong evaluation of such methods can result in an overstated promise of robustness and security. Here, it was demonstrated that the Adv-BNN approach of Liu et al. [62] does not display any indications of increased robustness against adversarial attacks. Their defense shows a slightly increased robustness against the weak adversarial examples generated by the naive PGD method compared to the adversarial training of a standard CNN. This advantage completely vanishes if an appropriate attack algorithm is used: in the face of the modified A-PGD method the robustness is decreased once again to the level of the baseline. If their BNN architecture is trained on adversarial examples from the A-PGD attack, its robustness is increased again: the model yields slightly higher robustness than both the adversarially trained standard CNN and the Adv-BNN trained on weak adversarial examples. This improvement is small and one could argue that it is not significant but merely based on the choice of hyperparameters. All in all, the findings of this chapter highlight the importance of averaging the gradients in gradient-based attacks against stochastic networks (e.g. Bayesian neural networks), as otherwise their robustness will be overestimated.

4. Image Independent Adversarial Noise for Humans

Whenever scientists propose a new hypothesis, they need to design experiments and to create tools to assess its validity. This is especially important in the domain of robustness of neural networks [14], as the previous chapter demonstrated: To measure robustness improvements of a neural network, one needs to have access to powerful evaluation tools with sufficiently strong attacks.

In both the current and the upcoming chapter, novel ways to find adversarial examples for an image classification model will be presented. These methods are designed such that they can be applied to a broad range of image classifiers and do neither require the models to be differentiable nor to allow access to the predicted class probabilities but only to the final decision. This means, that these attacks are decision-based [16]. Consequently, the attacks can be used to compare the robustness of a diverse set of classifiers, which also includes the robustness of a human classifier. Therefore, they can be used to assess the robustness of a neural network and to put this robustness into perspective by comparing it to the robustness of a human subject against both adversarial perturbations and image distortions. Existing studies, which compare the robustness of humans to that of neural networks, either use attacks that are specifically tailored to humans respectively to neural networks or use comparably weak attack descriptions (e.g. very low-dimensional texture models) [23–25, 51, 102]. In contrast to these studies, this work intends to find more general descriptions that are neither custom-fitted to known vulnerabilities of neural networks nor the human visual system.

The method proposed in this chapter is a weak form of an adversarial attack: it generates perturbations that are image independent and change the prediction of a classifier to a random class (i.e. an untargeted attack). In the following, this type of perturbation will be called *adversarial noise* to distinguish them clearly from perturbations created by normal adversarial attacks (e.g. PGD, C & W).

4.1. Generating Adversarial Noise

To generate adversarial noise a parameterized function in the form of a neural network is used. In general, the most efficient way to train such a network is to use gradient-based methods. However, this can only be done when a differentiable CNN is the target of the attack but not when the target is a black-box classifier such as a human subject. To circumvent this problem, a gradient-free optimization technique is used in both cases. For this, the well-tested Covariance Matrix Adaptation Evolution Strategy (CMA-ES, Section 2.4.1) is deployed. The goal of the optimization is to find the weights $\boldsymbol{\Theta}$ of a generative model $G_{\boldsymbol{\Theta}}$, such the median ℓ_2 distance between the original images and the adversarially perturbed ones

$$\epsilon^* = \underset{\mathbf{x},y\sim\mathcal{D}}{\text{median}} \{||\boldsymbol{\delta}||_2 \mid N(\mathbf{x}+\boldsymbol{\delta}) \neq y \quad \text{for} \quad \boldsymbol{\delta} \sim G_{\boldsymbol{\Theta}} \quad \wedge \quad \mathbf{x}+\boldsymbol{\delta} \in I\} \tag{4.1}$$

is minimal, whereby $\mathcal{D}$ represents the dataset, N the classifier under attack and I the image space (e.g. $[0,1]^{w\times h\times c}$). This is an often-used metric to measure robustness against adversarial perturbations (cf. Equation (2.12)). The latter constraint - that the perturbed images should be in the image space - is implemented by clipping the perturbed images accordingly for all experiments in this chapter.

To make the objective more practicable, instead the weakened objective

$$\arg\min_{\boldsymbol{\Theta}} \sum_i (N(\mathbf{x}_i+\boldsymbol{\delta}_i) = y_i) \quad \text{for} \quad \boldsymbol{\delta}_i \sim G_{\boldsymbol{\Theta}} \quad \wedge \quad ||\boldsymbol{\delta}||_2 < \epsilon \quad \wedge \quad \mathbf{x}+\boldsymbol{\delta} \in I \tag{4.2}$$

that measures the accuracy of a classifier under an attack is used to train the generator. Here, the perturbations found by the attack are constrained to be in an ϵ-ball. This objective enforces the accuracy of the classifier on the perturbed images to be as low as possible. In practice, a stochastic version of this objective is used, which measures the accuracy not on the entire dataset but only on mini batches. Using the cross entropy loss function $\mathcal{L}_{\mathbf{CE}}$, which is normally used to train image recognition networks, one can also rewrite this expression as

$$\arg\max_{\boldsymbol{\Theta}} \mathop{\mathbb{E}}_{\mathbf{x},y\sim\mathcal{D}} \mathop{\mathbb{E}}_{\boldsymbol{\delta}\sim G_{\boldsymbol{\theta}}} [\mathcal{L}_{\text{CE}}(N(\mathbf{x}+\boldsymbol{\delta}), y)] \quad \wedge \quad ||\boldsymbol{\delta}||_2 < \epsilon \quad \wedge \quad \mathbf{x}+\boldsymbol{\delta} \in I. \tag{4.3}$$

This objective can be used to find the optimal weights if both the generative as well as the recognition model are differentiable (Sections 4.2.4 and 6.1.1).

To evaluate the performance of the learned noise distributions $G_{\mathbf{\Theta}}$, the previously introduced ϵ value is used. During training, the size of the allowed perturbations ϵ^* is dynamically adjusted such that the average accuracy is always near 50 %. This is done to increase the information given to the optimization algorithm: a too large ϵ would result in a constant objective value near zero and a too small value to a constant value close to one, which are both not particularly sensitive to changes in the weights $\mathbf{\Theta}$.

While one is always interested in finding an optimization algorithm which needs as few queries to the network (i.e. predictions of the network for a given input sample) as possible, this is of special interest in the case of attacking a human classifier, as here every query consumes time of an experimental subject which causes costs. The process of recruiting and instructing experimental subjects also takes a considerable amount of additional time. To make such experiments feasible, one is, therefore, interested in keeping both the number of different experimental subjects required and the total experimental time as low as possible.

In psychophysical experiments, there is also an additional obstacle: the memory effect of humans [93, 97]. When a series of correlated stimuli is presented to a human, the subject can become aware of this correlation; this will influence its judgment. Imagine you have a set of images which all show the same scene but are perturbed by different levels of (Gaussian) noise; the task for the experimental subject is to classify the scene displayed. If one presents the images in ascending order of the noise level to a subject one gets different results than if the images are shown in descending order: once the subject has recognized the scene, even when stronger noise is added to the image, the subject might still recognize the scene, as it has already recognized it before.

For the task of generating adversarial noise, this means that one cannot use static noise such as the Universal Perturbations (UP) proposed by Moosavi-Dezfooli et al. [73], as a human subject can get used to the patterns which influences the experiment. Instead, it is more suited to use generative models that act as a probability distribution from which one can sample adversarial noises, like the UAN [41] and GAP [79] networks. Both of these models consist of a generative network [33] that transforms random variables drawn from a Normal distribution to an adversarial perturbation. This transformation is image independent in both networks; thus, the networks learn to generate UPs. UANs are closer related to GANs [33], as the initial random values are drawn from a low-dimensional distribution, transformed

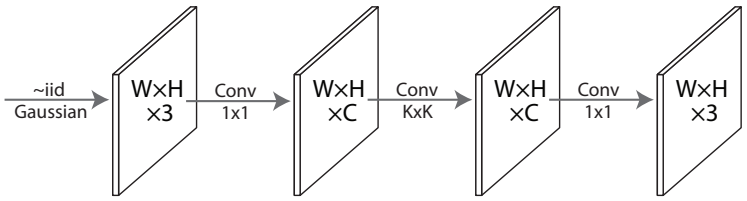


Figure 4.1.: Network architecture of the Spatial-K generators. The network transforms Gaussian noise using a local non-linearity modeled by 3 convolutional layers.

and finally scaled up to match the image dimensions. GAPs on the other hand directly start with random values that already match the image shape and are just transformed. While these networks are powerful generators, they also contain too many parameters to be trainable using gradient-free optimization techniques such as CMA-ES efficiently - especially if the queries need to be answered by a human. Therefore, one has to use generative models with a small number of parameters to obtain the image independent adversarial noise.

Given the constraints above, two types of generators will be derived in the following two sections. The conceptually closest related work is the GAP network by Poursaeed et al. [79].

4.1.1. Spatial Generators

The first type of generator proposed generates samples by transforming a random variable drawn from a multi-dimensional Gaussian distribution using local and non-linear functions. Its general structure is displayed in Figure 4.1. Here, the random variables are processed by three layers: the first and the third layer of the network act as non-linear and local functions which allow correlations across color channels. They are implemented by convolutional layers with a kernel size of 1×1. The second layer of the network is another convolutional layer with a kernel size of $K \times K$. As this is the only layer with a kernel size larger than 1, this means that the total correlation length of the noise generated by the network equals K. The larger the spatial correlation K gets, the more powerful the adversarial noise can be. In the limit of K being the spatial resolution of the image, the network should be able to produce adversarial perturbations similar to the ones produced by the UAN [41]. This increase in the network's generative power comes with the cost of requiring many parameters. The more parameters are used, the harder it becomes to optimize them. Thus, more queries to the image classifier under attack are required.

Model	Parameters
Spatial-1	183
Spatial-3	983
Spatial-3	2583

Table 4.1.: Overview of the number of parameters the different sub-types of the Spatial-K generator have. The table contains entries spatial correlation lengths $K \in 1, 3, 5$.

As mentioned above, this work looks for approaches that require a minimal number of queries to make it feasible to apply these methods to humans. Therefore, it is crucial to find a sweet spot such that K is as low as possible such that the network is still able to produce adversarial noise with a high probability. For this reason, only the three smallest spatial correlation sizes are analyzed: $K \in \{1, 3, 5\}$. An overview of the number of parameters that these networks have can be found in Table 4.1. Here, it becomes clear that the number of parameters easily goes beyond the limit of what can be optimized in a fast way by CMA-ES, which is about hundred of parameters.

4.1.2. Fourier Generator

To improve the efficiency of the noise by increasing the maximal allowed spatial correlation length without increasing the number of parameters too much, a new type of network architecture is derived. Here, the spatial correlation length of the noise is not set as a hyperparameter but instead is a trainable parameter of the network. Thus, the network can create perturbations with arbitrary correlation lengths with the same number of parameters. For this, the spatial correlation function $\tau_{\boldsymbol{\delta}}$ of the noise $\boldsymbol{\delta}$ is modified. To do this efficiently, the Wiener-Khinchin theorem [54, 96] is applied. The theorem relates the spatial correlation function to the power spectral density (PSD) P as a function of the Fourier transform $\mathcal{F}[\boldsymbol{\delta}]$ as

$$P = |\mathcal{F}[\boldsymbol{\delta}]|^2 = \mathcal{F}[\tau_{\boldsymbol{\delta}}]. \tag{4.4}$$

In the following, it is always assumed that both the correlation function as well as the PSD are two-dimensional functions, just as the image perturbation is. Given a target correlation function τ, one can now create a random perturbation $\boldsymbol{\delta}$ with this

correlation by utilizing the expression

$$\boldsymbol{\delta} = \mathcal{F}^{-1}[\sqrt{\mathcal{F}[\tau]} \odot \mathbf{z}] \quad \text{with} \quad \mathbf{z}_{ij} \in \mathbb{C} \quad \wedge \quad |\mathbf{z}_{ij}| = 1, \tag{4.5}$$

where $\odot$ means the elementwise product between the two matrices. The randomness of the variable $\boldsymbol{\delta}$ is controlled by the phase of the complex-valued random variable $\mathbf{z}$. The components of $\mathbf{z}$ are chosen randomly from the complex unit circle. Using the definition of the PSD this expression can be rewritten as

$$\mathbf{x} = \mathcal{F}^{-1}[\sqrt{P} \odot \mathbf{z}] \quad \text{with} \quad \mathbf{z}_{ij} \in \mathbb{C} \quad \wedge \quad |\mathbf{z}_{ij}| = 1, \tag{4.6}$$

where P is the PSD corresponding to the target spatial correlation function τ. Therefore, it is more practical to not directly model the spatial correlation function but instead the PSD. Usually, the PSD is visualized in a double-logarithmic plot. In the 1-D case, this corresponds to $\log P(\log k)$, where k is the frequency. Inspired by this, instead of P, its logarithm $f = \log P$ will be modeled. In the following, the logarithm of variables will be abbreviated with a prime, e.g. $\log k = k'$, for reasons of clarity and comprehensibility.

The crucial question is how to parameterize the PSD function P (i.e. its logarithm f) such that the description is powerful enough but does not have too many degrees of freedom. Here, one needs to decide which general form the parametrization of the function should have and which symmetries can be used. The more symmetries are used, the lower the degrees of freedom can become. For this, two different general functions are used: first, one can model the correlation function using a radially symmetrical (RS) function

$$f^{\boldsymbol{\theta}}(k'_x, k'_y) = f_r^{\boldsymbol{\theta}}\left(\sqrt{{k'_x}^2 + {k'_y}^2}\right). \tag{4.7}$$

To describe the function $f_r^{\theta} : \mathbb{R} \mapsto \mathbb{R}$, a power function of the form

$$f_r^{\alpha,\beta,\gamma,\delta}(x) = \alpha x^{1-\beta}(1-x)^{1-\gamma} + \delta \tag{4.8}$$

is used. This function has a high variability as shown in Figure B.13 in Appendix B; it is unimodal for all choices of the parameters, meaning that there is only one local maximum.

Alternatively, the correlation function can be assumed to be not radially symmet-

rical (NRS) but that it can be separated in one angle and one distance-dependent part

$$f^{\boldsymbol{\theta}}(k'_x, k'_y) = f_r^{\boldsymbol{\theta}_r}\left(\sqrt{{k'_x}^2 + {k'_y}^2}\right) f_\phi^{\boldsymbol{\theta}_\phi}\left(\sphericalangle(k'_x, k'_y)\right). \tag{4.9}$$

Here, the distance-depending component is again described by the function shown above in Equation (4.8). For the angle-dependent component, a linear combination of such power functions,

$$f_\phi^{\boldsymbol{\alpha},\boldsymbol{\beta},\gamma,\delta}(x) = \sum_i \alpha_i x^{1-\beta_i}(1-x)^{1-\gamma_i} + \delta, \tag{4.10}$$

is used, as experiments showed that a single unimodal function is not powerful enough.

To allow local non-linearities and correlations across the color channels the output of a network implementing Equation (4.5) is processed by another network that works like the Spatial-1 network described in the previous section. The complete architecture is shown in Figure 4.2. In the experiments, the effect of including

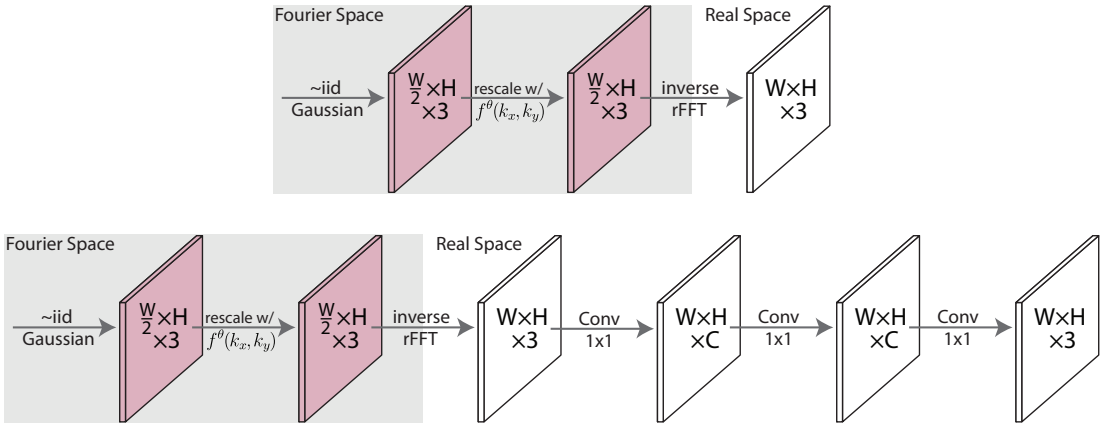


Figure 4.2.: Network architecture of the Fourier generators: the top shows linear generators (Fourier-RS, Fourier-NRS) and the bottom their non-linear counterparts (Fourier-RS-NL, Fourier-NRS-NL). Pink blocks refer to complex values in Fourier space, white blocks to real values. The network transforms complex-valued Gaussian noise in the Fourier space with a parameterized function f^θ before it is transformed to the real space. The parameters of f^θ are trained jointly with the rest of the network. The non-linearities in the bottom architecture are realized using the Spatial-1 architecture.

the additional non-linear processing to the output of the Fourier transformation is

examined. Note, that this non-linearity also allows for correlations across color channels. The number of parameters of the different Fourier-based generative networks is shown in Table 4.2. Here, it becomes evident that the description of spatial correlations in the Fourier domain using the Wiener-Khinchin theorem is very efficient in terms of parameters, compared to the naive version which only uses convolutional layers (cf. Table 4.1). This reduced number of parameters should improve the convergence speed of the training, such that fewer queries are required to train the network. It remains open how much this reduced number of parameters reduces the performance of the network in practice - this is evaluated in the following experiments.

Model	Parameters
Fourier-RS	4
Fourier-NRS	24
Fourier-RS-NL	187
Fourier-NRS-NL	207

Table 4.2.: Overview of the number of parameters the different sub-types of the Fourier-based noise generator have. The four generators compared are the Fourier generator with a radially symmetrical spectrum (Fourier-RS), the Fourier generator with a non-radially symmetrical spectrum (Fourier-NRS), and modifications of the aforementioned with an additional non-linearity (Fourier-RS-NL respectively Fourier-NRS-NL).

4.2. Experiments & Results

In the following sections, the efficiency and functionality of both, the different noise generators as well as the training scheme using CMA-ES are examined.

4.2.1. Analysis of the Baseline

Before the noise generated by the different proposed neural network architectures is analyzed, a baseline for the task is defined. In the following, this baseline is then used to benchmark the efficiency of the learned noise distributions in terms of how strong a noisy perturbation has to be until the prediction of a classifier is manipulated.

For this, a set of predefined and often encountered noise distributions in the real world or scientific experiments is tested. All the tested noise distributions are

pixel independent. Furthermore, they are purely local (e.g. iid.), which means that it is sufficient to describe how they change a single pixel of an entire image. To simplify notation, in the following clipping operations to fulfill the constraint that the perturbed image should lie in the image space will be left out for the sake of clarity. The set of noise distributions tested contains Gaussian-, Inverted Gaussian-, Uniform-, (Strictly Positive/Negative) Uniform-, Salt-, Pepper- and Salt & Pepper-Noise; furthermore, a constant positive/negative additive perturbation has been tested, too. A visualization of images with added noise can be found in Figure B.1 to B.3 in Appendix B.

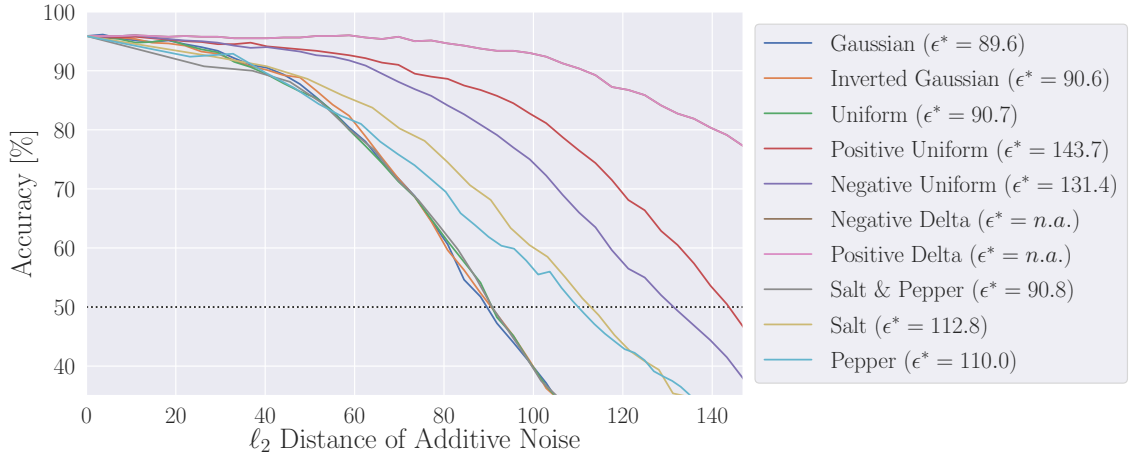


Figure 4.3.: Accuracy of the ResNet-50 model on noisy images from the ImageNet-16 dataset as a function of the ℓ_2 distance of the additive noise for different noise distributions. The dotted black line indicates an accuracy of 50%; intersections of curves with this line corresponds to the median ℓ_2 distance ϵ, which is shown in the legend.

In Figure 4.3 the reduced accuracy of a ResNet-50 classifier on images of the ImageNet-16 dataset perturbed by these noise distributions is shown as a function of the ℓ_2 size of the perturbation. The corresponding results for an MNIST and CIFAR-10 classifier can be found in Figures B.4 and B.5. A larger ℓ_2 size corresponds to stronger and more visible distortions of the original image. Here, one sees that the accuracy for all the noise sources stays almost unchanged for low ℓ_2 sizes and is then decreased almost linearly for stronger perturbations. Based on this figure, it is possible to measure the previously introduced ϵ^*, as it corresponds to the size of the perturbation which is sufficient to reduce the median accuracy to 50 %. The measured ϵ^* values for the different distributions are shown on the right side of the figure. For ImageNet-16 the minimal value is roughly $\epsilon^* \approx 89.6$ for Gaussian noise;

three other distributions have a very similar ϵ^* value. In the following, the value $\epsilon^* = 89.6$ is used as the baseline performance of a noise distribution one has to improve on. Improving this performance means to reduce ϵ^*, as this corresponds to smaller and less visible noise patterns that can confuse an image classifier.

4.2.2. Efficiency of Learned Noises

After the performance of the baseline distributions has been evaluated, now the performance of the proposed noise generator network architectures is analyzed. As described above, all models are trained with CMA-ES. The training is stopped once a fixed number of queries $(100,000)$ to the network has been used. This value is inspired by the planned application of the architectures and training method in psychophysical studies: here, one only has access to a limited amount of subjects, which constraints the number of possible queries. All hyperparameters used are listed in detail in Table C.2 in Appendix C. Again the performance is measured using the accuracy as a function of the ℓ_2 size of the perturbation. The results are displayed in Figure 4.4 for a ResNet-50 classifier on the ImageNet-16 dataset. For MNIST and CIFAR-10 the measured performance is displayed in Figures B.4 and B.5 in the appendix. In a first comparison with the results of the previous section, it becomes evident that all the proposed architectures are able to reduce the accuracy of the classifier already at a lower distortion level: while for the baselines distortion sizes of ≈ 40 are required to notice a first drop in the accuracy, for the noise generators this already happens at a distortion size of ≈ 20. The recorded functions of the noise generators have a steeper slope than those of the baselines, too. In the legend of Figure 4.4 again the median required distortion size ϵ^* to find adversarial images is shown. A visualization of images with added adversarial noise of this strength can be found in Figure B.6 to B.8 in Appendix B. The values can be compared to the ones reported for the baselines in Figure 4.3. The results are striking: even the most simple and worst noise generator (Spatial-1) achieves a result of $\epsilon^* = 66.1$, which beats the value $\epsilon^* = 89.6$ measured for the best baseline (Gaussian). The best performing generator is the Fourier-NRS-NL architecture. For this architecture, a value of $\epsilon^* = 33.2$ is measured, which is close to one-third of the best baseline. All in all, these results show the supremacy of the proposed generator architectures. Furthermore, it becomes evident that the parametrization in the Fourier space is a powerful description of the noise and has a better efficiency in the face of purely decision-based optimization and a limited number of queries.

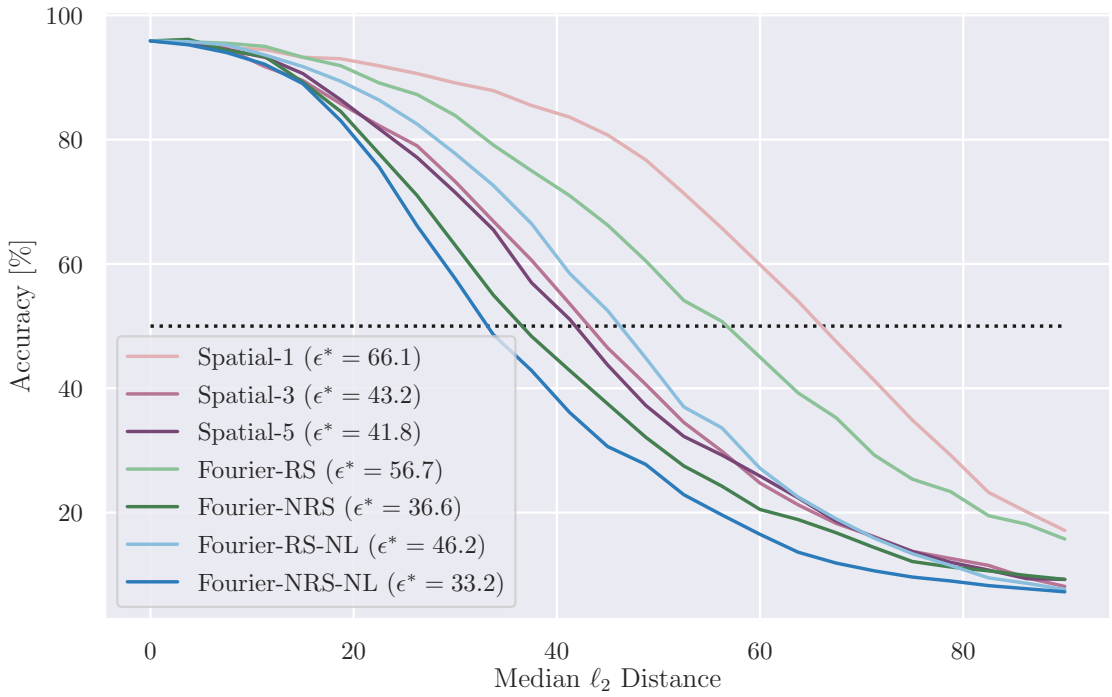


Figure 4.4.: Accuracy of the ResNet-50 model on noisy images from the ImageNet-16 dataset as a function of the ℓ_2 distance of the additive noise for different noise generators trained with CMA-ES.

4.2.3. Comparison with Related Work

After it has been demonstrated in the previous section that the learned noise distributions are more efficient than the baseline distributions (cf. Section 4.2.1), they are now compared to models proposed by previous work. For this, the best-performing model (Fourier-NRS-NL) is compared to the UAN [41] and GAP [79] networks. Following the default evaluation benchmark in the literature, the so-called fooling rate is reported. This rate measures for how many images the predicted class of an image changed when the adversarial perturbation has been added. This includes all images and not only those the network predicted correctly in the absence of noise. As stated above, the GAP networks are conceptually the closest related work to the proposed models in this work. In GAPs, spatial correlations across the image are enabled by multiple convolution layers with several hundred thousand parameters. In contrast to them, the Fourier-NRS-NL model uses an efficient low-dimensional parametrization to model non-local correlations, which in total only uses a few hundred parameters. Due to a vastly higher number of learnable parameters, GAPs can be expected to perform better than the Fourier-NRS-NL.

The results of the comparison are displayed in Table 4.3. Here, the performance is measured for three different networks (VGG-16, VGG-19, ResNet-152) which are trained and evaluated on the ImageNet dataset. These are compared to two versions of the Fourier-NRS-NL model: the first one is the same model used in the previous section and has been trained on the smaller ImageNet-16 dataset while the latter was trained on the ImageNet dataset. Following the commonly used benchmark in the literature, the adversarial perturbations are bound to have an ℓ_2 size of 7.84. For the GAP and UAN the values reported in their publication have been used here and were not calculated again. The results show that the Fourier-NRS-NL achieves

Generator	Parameters	VGG-16	VGG-19	ResNet-152
GAP [79]	757,379	93.9%	94.9%	79.5%
UAN [41]	157,112,585	-	86.0 %	91.4 %
Fourier-NRS-NL*	207	38.6 %	37.5 %	19.8 %
Fourier-NRS-NL	207	42.9 %	44.6 %	22.1 %

Table 4.3.: Comparison of the Fourier-NRS-NL generator, the Generative Adversarial Perturbation (GAP) approach [79] and the Universal Adversarial Network (UAN) [41]. The number of parameters of the generators (second column, the smaller the better) as well as the error rates caused by the adversarial noise generators (third to last column, the larger the better) are compared. The asterisks indicates that the model is trained to attack an ImageNet-16 and not an ImageNet classifier but was evaluated on the ImageNet dataset to allow for comparison.

roughly half of the fooling rate of the two baselines for the VGG models. For the generally better performing ResNet-152, this gap widens: while the two baselines can approximately maintain their fooling rate, the performance of the Fourier-NRS-NL has a relative drop of 50 %. The results substantiate the hypothesis, that both GAP and UAN perform better due to their larger network size. Furthermore, one has to keep in mind that the Fourier-NRS-NL is trained with a different objective than the UAN/GAP models: the latter were directly trained to have a maximal fooling rate for the ℓ_2 distance used in the evaluation, while the former is trained to have a minimal ϵ^* value, i.e. minimal median ℓ_2 distance required to fool the network for all images. Even though these objectives are highly related, these differences can still explain some of the differences between the results.

4.2.4. Performance of Evolution Strategies

In comparison to existing studies, this work proposes new architectures for the generative model and uses a different optimization scheme. In the previous sections, the choice of the architecture of the generative model has been evaluated and compared to alternative designs. Now it remains to also analyze the used optimization algorithm. This allows one to disentangle the two possible sources of low performance to direct future research. Therefore, in this section the performance of a model trained using CMA-ES is compared to the ones trained using gradient-based optimization for the proposed architectures. For this, the architectures are trained with SGD with momentum. The hyperparameters used are listed in detail in Table C.1. To compare the two, the median ℓ_2 distance ϵ^* between the clean and adversarially perturbed images for the validation dataset is measured during the training process. This gives insights into both the convergence behavior and the final performance of the algorithms. The results of the experiment for the ImageNet-16 dataset are displayed in Figure 4.5. More results for other datasets can be found in Figures B.9 and B.10. In the figure, one can see that for the Spatial-K generators both optimizers converge towards similar final performance levels even though SGD reaches this point faster. This observation also holds true for some of the Fourier-based generator architectures. However, for some of these, the SGD optimization seems to be inefficient and appears to get stuck in sub-optimal, local minima as the final performance is lower than the final performance of the networks trained with CMA-

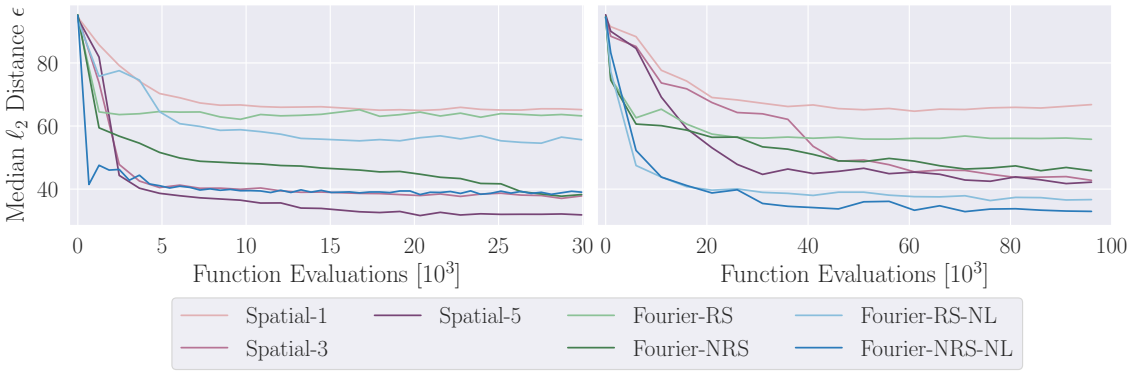


Figure 4.5.: Comparison between the training progress of noise generators trained with SGD (left) and CMA-ES (right) against the ResNet-50 model on ImageNet-16. The training progress is measured as the required median ℓ_2 distance to reduce the classifier's accuracy to 50% as a function of the function evaluations.

ES (cf. Fourier-NRS generator). This might be explained by two things: first, the hyperparameter search for CMA-ES has been more thorough than the one for SGD. Therefore, it is still possible that there are SGD parameters that lead to similar results as CMA-ES does. Second, the Fourier-based generators use double-exponential functions to parameterize the spatial correlations (Equation (4.6)). These functions can have sharp gradients that make gradient-based optimization techniques more likely to diverge and to show effects of numerical instabilities. In summary, these results confirm the choice of CMA-ES as the optimization algorithm. A future research direction could be to stabilize the parametrization in the Fourier space to improve convergence behavior of gradient-based optimizers. This could allow using these architectures as an efficient alternative to GAPs or UANs in scenarios in which one is not depending on the network to be trainable using gradient-free optimizers.

4.2.5. Conclusion

The chapter explored a way to assess the robustness of general image classifiers. A method to generate image independent perturbations that have a high likelihood of confusing an image classifier was introduced and evaluated. For this, two types of generator network architectures were evaluated. Both of them were able to surpass a baseline of simple noise distributions in terms of the required perturbation size to confuse a CNN. The network architecture, which generates perturbations with arbitrary spatial correlations using the Fourier space, showed the best performance. Furthermore, it was demonstrated that these networks can be trained without gradient-based methods but using an Evolution Strategy. This allows researchers to apply them also to non-differentiable and black-box image classifiers. In comparison to vastly larger networks used in the existing work, the proposed networks produce less efficient perturbations: this highlights the trade-off between a sufficiently small size of the network, such that an Evolution Strategy can be utilized, and the attack's performance. It remains an open challenge‘ to close the performance gap further while keeping the training feasible using non-gradient-based methods.

Another future step is to create a psychophysical experiment to utilize the proposed method in a human experiment. On the one hand, this will allow verifying that the proposed method can successfully attack a human classifier. On the other hand, it will allow for a comparison between humans and machines in terms of their robustness in an image classification task. As the design and execution of such an

experiment go beyond the scope of this work, it remains an open task for the future.

5. Image-Dependent Adversarial Attack for Humans

After the previous chapter has introduced and tested a method to generate image-independent adversarial noise, which has a high probability to reduce the predictive power of an image classifier, an alternative image-dependent attack is introduced in this chapter. In principle, one could increase the capacity of the previously described generative network and feed the original image and/or the ground-truth label to it to make the predicted perturbations image-dependent. However, this approach would cause the generator to contain not only a few hundred but possibly millions of parameters. As the applied ES-based algorithms are only efficient for optimization problems with a few hundred parameters, this approach is not feasible. Instead, in this chapter, another approach is followed, which is based on the so-called Markov Chain Monte Carlo with People (MCMC-P) algorithm (Section 2.5) [7]. A great advantage of it is that it allows performing an optimization of the same objective on both humans and machines in a very similar way. This makes it the perfect candidate for the task of comparing the robustness against adversarial examples in both humans and machines.

5.1. Constrained Markov Chain Monte Carlo with People

The MCMC-P algorithm can be used to optimize stimuli presented to a subject such that the class-dependent probability of the presented stimuli should be either maximized or minimized. In the context of generating adversarial examples this translates to finding an adversarially perturbed image $\mathbf{x}'$ for which the class probability of the ground-truth class c is minimized, i.e. $\min_{\mathbf{x}'} p(\mathbf{x}'|c)$. To find minimally perturbed adversarial examples, the distance between the original image $\mathbf{x}$ and the adversarial example has to be minimized. This can be expressed as a constrained optimization

problem: minimize the distance such that the probability of the ground-truth class becomes less than for at least one other class (Section 2.3). The constraint can also be replaced with a fixed constant ξ as an upper bound to simplify the optimization problem, if ξ is chosen high enough. All in all, this results in the optimization problem

$$\min_{\mathbf{x}'} f(\mathbf{x}') \quad \wedge \quad \log p_C(\mathbf{x}'|c) \leq \xi, \tag{5.1}$$

where $f(\mathbf{x}') = d(\mathbf{x}, \mathbf{x}')$ measures the distance to the original image and $p_C(\cdot|c)$ is the predicted class probability of an image classifier. In the remainder of this section, the distance d is always assumed to be the Euclidian ℓ_2 norm. However, all methods proposed can just as well be applied to any other continuous distance measuring function. Using the Lagrangian formalism, the objective can be rewritten as

$$\min_{\mathbf{x}} f(\mathbf{x}) \quad \wedge \quad \log p_C(\mathbf{x}|c) \leq \xi \tag{5.2}$$

$$= \min_{t} \max_{\mu \geq 0} t - \mu\xi + \max_{\mathbf{x} \text{ s.t. } f(\mathbf{x})=t} \mu \left[\log p_C(\mathbf{x}|c)\right]. \tag{5.3}$$

A detailed derivation of this expression is given in Appendix D.

To reduce the size of the space in which the stimulus is searched, one can introduce a stimulus generating function $h : \mathbb{R}^n \mapsto \mathbb{S}$ (see the description of MCMC-P in Section 2.5). This function has the purpose of mapping a low-dimensional, abstract state to a stimulus - which can be seen as an inverted dimensionality reduction. Using h, one can rewrite the objective again as

$$\min_{\mathbf{x}} f(\mathbf{x}) \quad \wedge \quad \log p_C(\mathbf{x}|c) \leq \xi \tag{5.4}$$

$$= \underbrace{\min_{t} \max_{\mu \geq 0} t - \mu\xi + \underbrace{\max_{\mathbf{s} \text{ s.t. } f(h(\mathbf{s}))=t} \mu \left[\log p_C(h(\mathbf{s})|c)\right]}_{\text{Inner Loop (MCMC-P)}}}_{\text{Outer Loop}}. \tag{5.5}$$

This objective can be seen as a bi-level optimization problem with an inner and an outer loop. The outer loop can be solved by a simple binary search over the solutions of the inner loop. The inner loop can be solved using a variant of the MCMC-P procedure (Algorithm 2.2), which finds the optimal low-dimensional state s. A description of an algorithm to solve the optimization problem with such a function h is given in Algorithm 5.1. An illustration of this algorithm is also given

in Figure 5.1.

Algorithm 5.1: Constrained Optimization MCMC-P (CMP).

Input: Initial value $\mathbf{x}_0$, objective function $f : \mathbb{S} \mapsto \mathbb{R}$, Stimulus generating function $h : \mathbb{R}^n \mapsto \mathbb{S}$, number of search steps s

Result: Constrained optimum $\mathbf{x}_s = \arg\min f$

for $i \leftarrow 0$ ***to*** s **do**
  $\epsilon_i := f\left(h(\mathbf{x}'_{i-1})\right)$
  //use MCMC-P (Algorithm 2.2)
  $\mathbf{x}'_i \leftarrow \underset{f(h(\tilde{\mathbf{x}}))=\epsilon_i}{\arg\max}\ p(h(\tilde{\mathbf{x}})|c)$
  Reduce ϵ_i to get closer to decision boundary (e.g. binary search)
end
return $\mathbf{x}'_s$

For the MCMC-P procedure, the choice of the proposal distribution is important: On the one hand, this distribution should be symmetrical (Section 2.5) and on the other hand, the support of this distribution must be constrained to the feasible set of the constraint. This set is defined as the preimage of the objective function f for a given value t. A possible candidate for a valid distribution satisfying these requirements is a transformed version of the often used Normal distribution. Here, the distribution is transformed such that all samples drawn from it lie in the feasible set of the constraint. This is done by projecting every sample of the distribution to the nearest point in the set. It is not necessary to explicitly know the analytical expression for the distribution, but sufficient to know how to sample from it. A general sampling mechanism that works for all continuous objective functions f and stimulus generating functions h is given in Algorithm A.1 in Appendix A. Because of its generality, this scheme can be inefficient. If h and f are known, one can derive more efficient methods. As the aim of this chapter is to find adversarial examples, in the remainder f will be the ℓ_2 distance between the adversarially perturbed and the original image. Furthermore, as it will be introduced below, a semi-orthogonal projection will be used for h. Such projections are distance-conserving for the ℓ_2 norm. In this special case, there even exists a simple analytical expression for a proposal distribution:

$$\mathbf{s}_{t+1} = \tilde{\mathbf{s}} - \mathbf{s}_t \frac{\tilde{\mathbf{s}}^T \mathbf{s}_t}{\mathbf{s}_t^T \mathbf{s}_t} \quad \text{with} \quad \tilde{\mathbf{s}} \sim \mathcal{N}(\boldsymbol{s}_t, \sigma^2), \tag{5.6}$$

whereby $\mathbf{s}_t$ is the current state of the chain and $\mathbf{s}_{t+1}$ is the proposed state that will

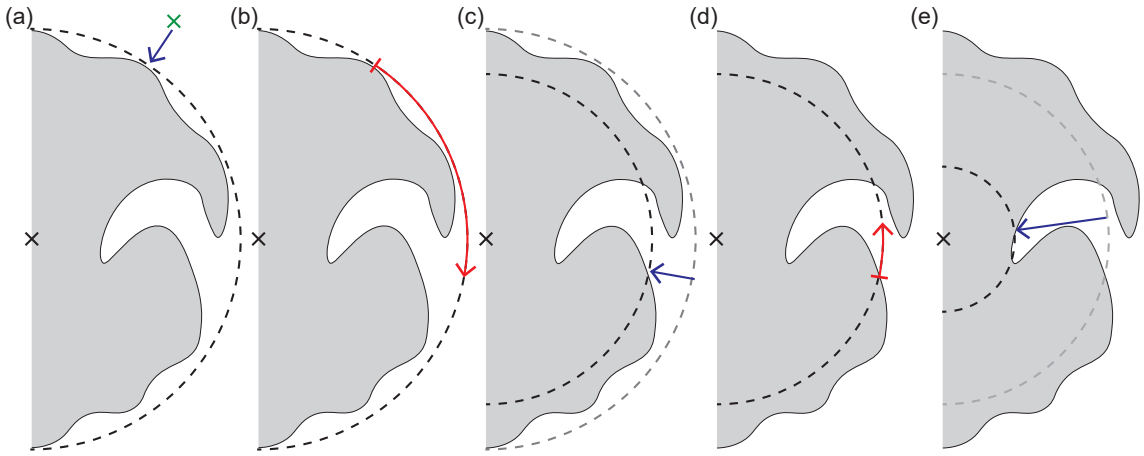


Figure 5.1.: Illustration of the proposed (PC-)CMP attack described in Algorithm 5.1. The black cross represents the unperturbed image, the green one an unperturbed image of another class which is used as the initial search position. The dotted circles correspond to the isolines of f; the purple lines to shrinkage operations in which the direction is kept fixed and only the magnitude is varied (outer loop); and the red arcs correspond to the MCMC search (inner loop).

be the next value sampled from the proposal distribution.

In principle, to generate the stimuli in the context of adversarial examples, one could utilize a broad family of (invertible) dimensionality reduction techniques. Examples of these are bilinear upsampling [1] or a principal component analysis [7, 77]. In this work, only the application of the principal component analysis (PCA) is tested and analyzed. In PCA, an orthogonal transformation is searched which transforms correlated variables into a set of uncorrelated variables. These new variables are called the principal components (PCs) of the original data. Usually, they are ordered in a descending way according to how much variance in the data each PC explains. In other words, the first PC explains most of the variance of the data, whereas the last PC explains very little. By restricting the transformation to use only a small number of PCs (only the first n), one can create a transformation that maps a low-dimensional vector to the high-dimensional space of the original data. To apply this approach in CMP, a PCA was applied to the training subset of the MNIST data. This allows to create a linear transformation matrix $\mathbf{T} \in \mathbb{R}^n \times \mathbb{S}$ that maps a low-dimensional vector to the high-dimensional space of MNIST images while explaining most of the variance of the original MNIST images. A useful property of this matrix is that it is semi-orthogonal and therefore distance-preserving, i.e.

$||\mathbf{Tx}||_2 = ||\mathbf{x}||_2$. By using this matrix as the stimulus generating function $h(\mathbf{s}) = \mathbf{Ts}$, one makes sure that adversarial examples are only searched for in the subspace of data that is close to the original image manifold. The number n of PCs used is a hyperparameter of the attack. It has to be adjusted according to the dataset such that n is minimal with the constraint that the space spanned by the PCs has a sufficient size; otherwise, it would not be possible to construct adversarial examples with a minimal perturbation size in this space. The combination of PCA and the CMP algorithm with the aim of generating adversarial images will be called PC-CMP in the remainder of this work. To improve the efficiency of the PC-CMP attack, the initial state is chosen such that the fist adversarial example proposed $\mathbf{x}'_0$ is a linear interpolation in the PC space between the original image $\mathbf{x}$ and an image of the training dataset that belongs to another class. Unless stated otherwise, the attack will use 50 PCs in the remainder of this chapter; this choice is later also motivated by experimental results.

5.2. Experiments & Results

In this section, the experiments performed with the proposed PC-CMP attack and their results are presented. For the evaluation of PC-CMP, experiments with humans are again not used. This is due to the fact, that designing and executing psychophysical experiments is complex, time-intensive and overall challenging and hence was beyond the scope of this thesis. Instead, the PC-CMP attack is used to attack different CNNs. Thereby, the influence of the hyperparameters of the attack as well as the overall convergence behavior of the attack is analyzed. This can be seen as the first step towards later psychophysical experiments. All experiments are performed on the MNIST dataset, as this is an often-used testbed for adversarial attacks [65, 87].

Because CNN-based image-classifiers are used in this chapter, the decision function $d(\mathbf{x}_1, \mathbf{x}_2)$ introduced above can be made more precise: Given a trained image classification network N, the binary decision function $d : \mathbb{S} \times \mathbb{S} \mapsto \{1, 2\}$ can be expressed as

$$d(\mathbf{x}_1, \mathbf{x}_2) = \begin{cases} 1, & \text{if } N_p(\mathbf{x}_1)_y > N_p(\mathbf{x}_2)_y \\ 2, & \text{otherwise} \end{cases}, \tag{5.7}$$

where N_p corresponds to the class probabilities predicted by the network N and y is the ground-truth label of the original image. This decision function is always used in the remainder of this chapter.

5.2.1. Convergence of the Attack

First, the sensitivity of the proposed PC-CMP attack and its general behavior are examined. For this, a grid search over its three hyperparameters - the step size λ, the number of MCMC steps N and the number of outer loop iterations s - is performed. During this grid search the median ℓ_2 distance ϵ^* is measured for a subset of 500 randomly chosen images. This is repeated 15 times using the same images and the same initial search location of the PC-CMP attack to also test the consistency/reproducibility of the attack's adversarial examples. The results can be found in Figure 5.2; the top image shows the mean over the 15 repetitions, while the bottom image shows the standard deviation. One can observe three trends in the collected data. First, the more outer loop iterations s are performed, the lower the mean and standard deviation get. While the mean does not change strongly anymore after 3 iterations, the standard deviation steadily decreases with an increasing number of iterations. As we aim at constructing adversarial examples with as few queries as possible, s should be chosen as low as possible without worsening the results. As the improvement between $s = 5$ and $s = 6$ is only minute, $s = 6$ outer loop iterations seem to be sufficient to reach stable adversarial examples. Second, there is a sweet spot for the learning rate at $\lambda = 0.2$ where the lowest ℓ_2 distance can be reached; increasing or decreasing λ increases the distance again. Third, the more MCMC steps N are performed, the lower the mean and the standard deviation become. Here one has to deal again with the trade-off between performance gains and additionally required queries. If the learning rate λ is chosen accordingly, increasing N beyond 350 only yields small performance improvements. As a results of the presented explorations, a step size $\lambda = 0.2$ and $N = 350$ MCMC steps are used in the following experiments, as this choice results in both a low mean and low standard deviation.

Next, the convergence behavior of the attack is further analyzed. To this end, the optimal values found in the grid search are used for the step size λ and MCMC steps N. To let the attack run for a longer time, the number of iterations s is modified and set to 20. Next, the ℓ_2 distance between adversarially perturbed and clean

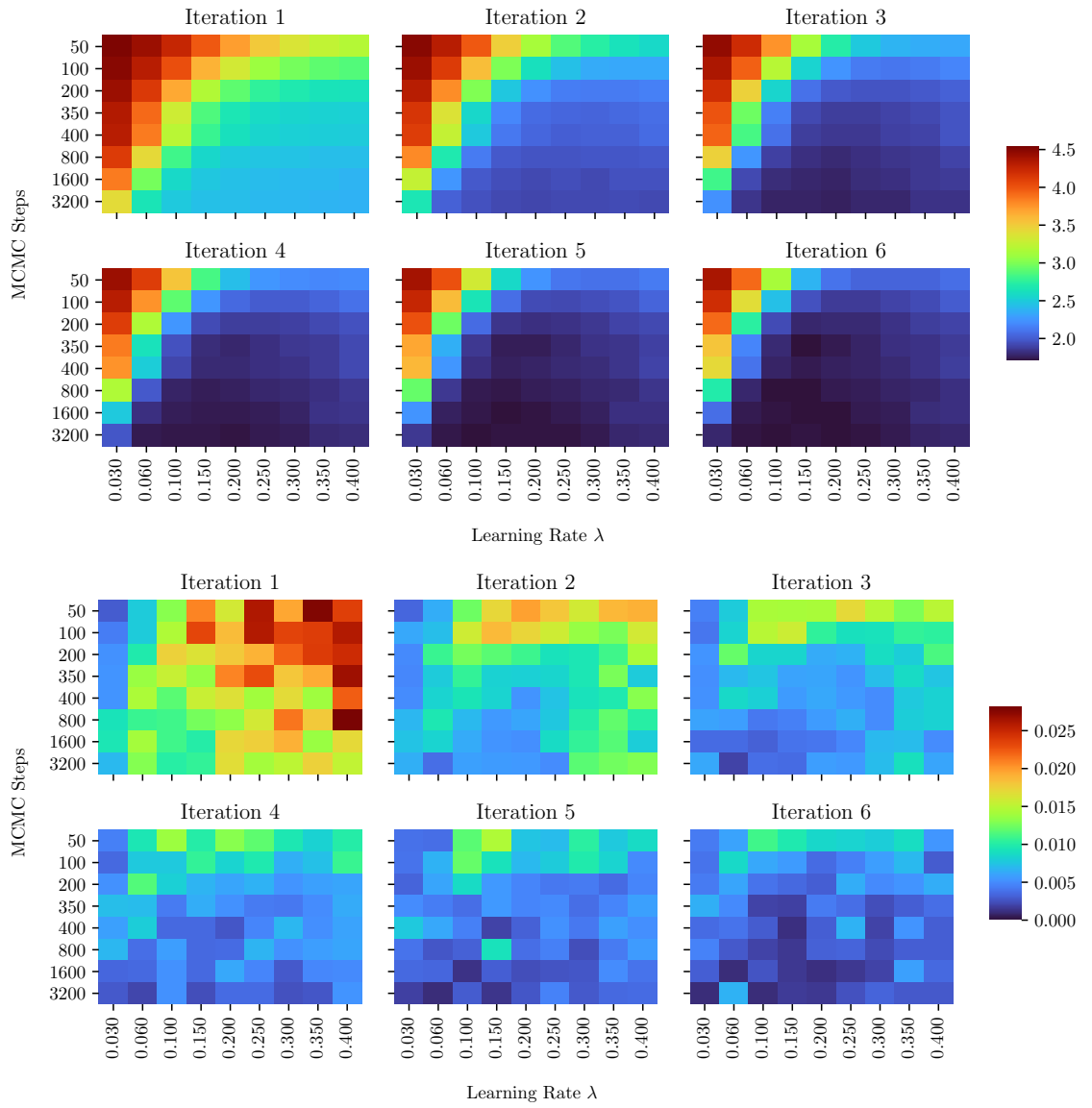


Figure 5.2.: Results of a grid search over the number of MCMC steps N, the learning rate λ and the search steps s for the median ℓ_2 distance of the adversarial images found for 500 images. The grid search was repeated 15 times, with identical initial perturbations for each repetition. While the upper six heat maps show the mean of the resulting median ℓ_2 distances, the ones on the bottom show the corresponding standard deviations.

images is measured as a function of required queries to the network N. Additionally, the probability margin $N_p(\mathbf{x}')_y - \max_{k \neq y} N_p(\mathbf{x}')_k$ between the ground-truth class probability and largest other probability is measured as a function of the queries. This margin is bound between -1 and +1. If the image $\mathbf{x}'$ is adversarial, it is even more tightly bound between -1 and 0. The value is equal to zero if the image $\mathbf{x}'$ is on the decision boundary (i.e. the network infers the same probability for two different classes) and goes to -1 for adversarial examples with maximal class probability for an adversarial class $\neq y$. These two functions are displayed in Figure 5.3 for the C & W model on MNIST. Both curves jump every $N = 350$ queries. These jumps

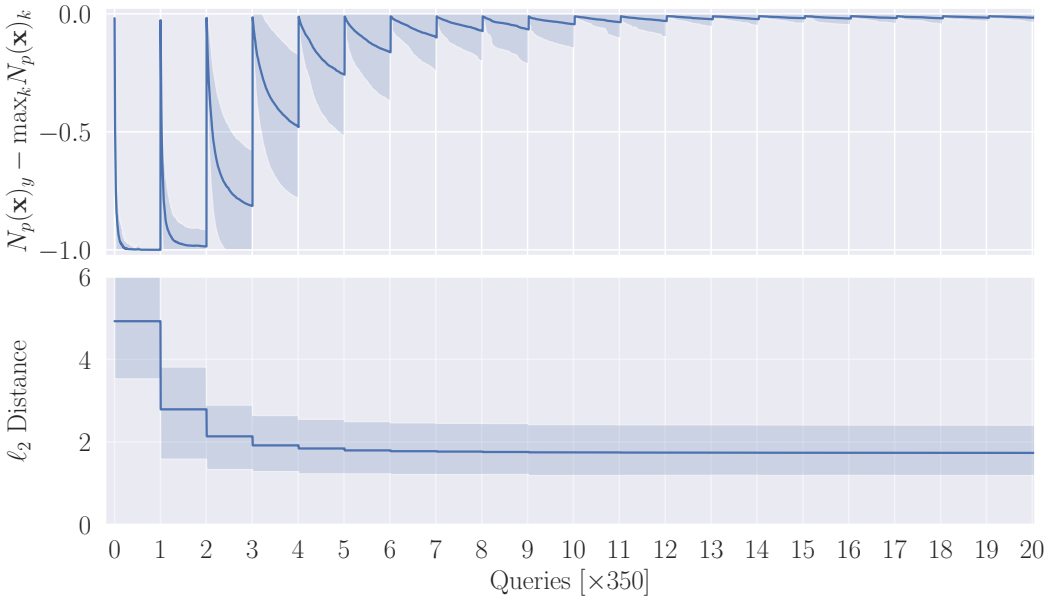


Figure 5.3.: Progress of the optimization performed by the PC-CMP attack with 50 PCs for $N = 350$ MCMC steps, $\lambda = 0.2$ and $s = 20$ outer loop iterations. The outer loop iterations correspond to the jumps in the graphs. The top row shows the difference between the ground-truth label's probability $N_p(\mathbf{x})_y$ and the highest predicted probability $\max_{k \neq y} N_p(\mathbf{x})_k$ which differs from the ground-truth label. The bottom row shows the ℓ_2 distance between original and adversarial images. Solid curves show the median value, shaded areas the 1σ-interval.

arise from the outer loop of the PC-CMP attack, in which the search radius gets reduced. As the radius is kept fixed in the inner loop, the ℓ_2 distance shown in the bottom row is a piecewise constant function that is reduced every N steps. The jumps become smaller and smaller indicating that the optimization is converging. After ≈ 6 iterations the distance does not change significantly anymore but stays

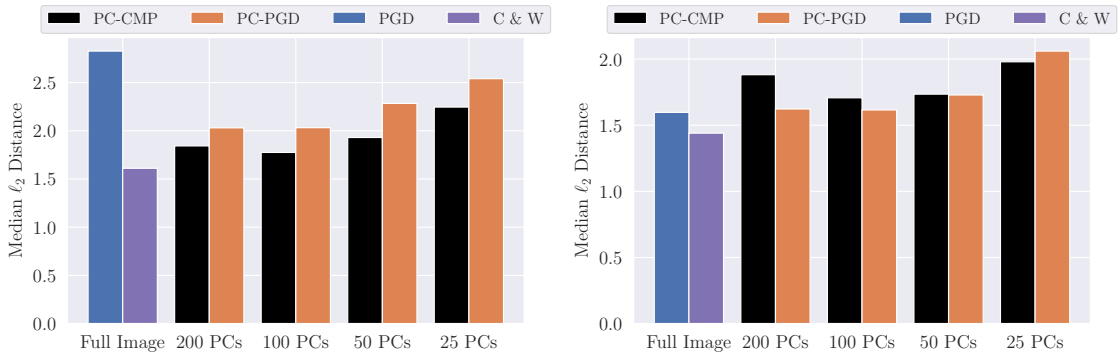
constant. This strengthens the findings of the previously conducted grid search (cf. Figure 5.2). The curve in the top row is bound between 0 and -1. That shows that all images generated by the PC-CMP attack are indeed adversarial by design (cf. Figure 5.1), which can be seen as a sanity check demonstrating that the attack is correctly implemented. Furthermore, in the process of the attack, the shape of the curve changes such that it slowly converges towards 0. This value represents the decision boundary, as the predicted class probability for the ground-truth class is just as high as the largest non-ground-truth class probability at this location. Furthermore, this observation also confirms that the attack is converging properly, as the final results of the attack lie close to the decision boundary. In comparison, the upper probability margin curve converges more slowly than the ℓ_2 distance at the bottom. This shows that near the decision boundary even minute changes of the ℓ_2 radius can drastically influence which class is predicted most strongly.

5.2.2. Comparison with other Attacks

After a well-working set of hyperparameters for the attack has been determined with the previous two experiments, now the performance of the PC-CMP attack shall be compared to other adversarial attacks. For this, the Madry model and the C&W model for MNIST are attacked. The PC-CMP attack with different numbers of Principal Components (PCs) is compared to a gradient-based PGD attack constrained to the space spanned by PCs (PC-PGD), the normal ℓ_2-PGD attack (Section 2.3.1) and the C&W attack (Section 2.3.2). For all attacks, a search over the hyperparameters is performed. The results of this comparison, in form of the median ℓ_2 distance ϵ^* between the original and adversarial image, are shown in Figure 5.4. A visualization of the adversarial examples found by the PC-CMP and the different attacks can be found in Figure B.11 for the C&W model and in Figure B.12 for the Madry model in Appendix B.

In the bar plot, one sees that there exists a sweet spot of PCs used in the PC-CMP attack for both models. If there are too few components the expressive power of the subspace spanned by the PCs is too low. However, the number of queries required to find an optimum becomes larger with an increased number of PCs, as the size of the space one has to search grows exponentially. The 50 PCs seem to be (close to) the sweet spot for both of the attacked models and will, therefore, be used in all experiments unless stated otherwise.

Adversarially trained models, such as the Madry model, are known to be prone



(a) Madry model on MNIST.

(b) C & W model on MNIST.

Figure 5.4.: Comparison of the median ℓ_2 distance between the adversarial examples found by different adversarial attacks and the original images for the MNIST dataset. The proposed algorithm PC-CMP is compared to a PGD variant operating on the subspace spanned by the first k principal components (PCs) for $k \in \{25, 50, 100, 200\}$; these results are also compared to two attacks operating on the full image, namely PGD and C & W. On the left-hand-side results for the Madry model are shown, while on the the right-hand-side results for the C & W model are displayed.

to gradient masking. This means, that the gradient, which is for example used in the PGD attack, almost vanishes and contains only little information about the direction of adversarial perturbations for an image. The data shows that the Madry model performs gradient masking, as the normal PGD attack is less efficient than the PC-CMP attack. This indicates that PC-CMP is indeed robust against gradient masking as it also finds adversarial examples with a smaller distance to the original images than the PGD-based attack using not the full image but only a PC-subspace.

In the previous experiments the PC-CMP attack has always been initialized such that the initial proposal is a linear interpolation between the unperturbed image and an image that belongs to another class. This initialization has turned out to be important to reduce the number of MCMC steps required to reach convergence. In the next experiment it shall be assessed, how much of the success of the attack comes from the actual CMP procedure and how much from the initialization. This will be tested in the following experiment. Here, the efficiency of the initial perturbations is tested. For this, a new attack, that looks for adversarial images using the previously

described linear interpolation, is used:

$$\mathbf{x}' = \mathbf{T}\mathbf{T}^T(\alpha\mathbf{x} + (1-\alpha)\mathbf{x}_o). \tag{5.8}$$

Here $\mathbf{T}$ is the transformation matrix to the PC space which was introduced above and $\mathbf{x}_o$ is an image that is randomly chose from the dataset and belongs to any but the ground-truth class of the original image. The attack then looks for the maximal value of α such that $\mathbf{x}'$ is adversarial. The resulting median ℓ_2 distance ϵ^* of this attack for the two MNIST models used before are shown in Table 5.1. For both

	Number of PCs n			
Model	200	100	50	25
C & W-MNIST	3l.63	3.60	3.58	3.73
Madry-MNIST	4.17	4.19	4.17	4.12

Table 5.1.: Median ℓ_2 distance ϵ^* of the adversarial perturbations found by the linear interpolation attack in the PC space.

models and all numbers of PCs n used, the measured distances are much larger than the values obtained using the PC-CMP attack; for the Madry model they are even twice as big. Therefore, one can conclude that even though the initialization is important for the PC-CMP attack, the success of it depends on the CMP procedure.

5.2.3. Robustness of the Attack

Since the purpose of the derived attack is to attack human decisions, one has to think about the nature of human decisions, too. Compared to the decisions of machines, humans have strong discriminative power, but often fail to notice minute differences in the input. This means, that two close data samples might be indistinguishable for a human, even though they are distinguishable for a CNN. In such an indistinguishable regime, the decisions of a human in a 2-AFC task become random, as the decision boundary becomes very fuzzy. As the proposed PC-CMP attack relies heavily on such decisions, it makes sense to test the robustness of the attack in such situations.

To analyze the robustness of PC-CMP against such stochastic effects, a modified version of it is introduced now. In this version, the stochasticity of the classifier is simulated by modifying the MCMC sampling step in the attack. Here, normally the following task has to be solved by the image classifier N: given two images, $\mathbf{x_1}, \mathbf{x_2}$

and a class label y, chose the image which resembles the class y more strongly. The deterministic choice d of this task is given by calculating and comparing the predicted probabilities $N_p(\mathbf{x})_y$ for the given class for both images; the proposed image with the higher probability is then chosen as the answer. This corresponds to the previously introduced decision rule in Equation (5.7). To allow stochastic mistakes, i.e. make the decisions near the decision boundary $N_p(\mathbf{x}_1)_y \approx N_p(\mathbf{x}_2)_y$ fuzzier, a probabilistic version of this task can be used. In this, the decision is not deterministic but sampled from a probability distribution

$$d \sim \frac{e^{N_p(\mathbf{x}_1)_y/T}}{e^{N_p(\mathbf{x}_1)_y/T} + e^{N_p(\mathbf{x}_2)_y/T}}. \tag{5.9}$$

Here, T is a temperature factor controlling the level of stochasticity of the decision. The higher T is, the more stochastic the decision d becomes; in the limit of vanishing $T \to 0$ the decision d becomes the deterministic choice again. To get a better understanding of how fuzzy the decision boundary becomes for certain levels of T a new toy-task will be used. This task is again a 2-AFC task similar to the one above. But instead of comparing two similar images as in CMP, two images from random classes are compared: given two images, $\mathbf{x_1}$ of class y and $\mathbf{x_2}$ of class $y' \neq y$, chose the image which resembles the class y more strongly. This task is then solved for every combination of images in the dataset. It measures how certain the classifier is that a sample belongs to the ground-truth class compared to another sample. Like in the MCMC process, to solve this task a 10-class image classifier N, trained on the MNIST dataset, has been used. To obtain the decision d, the same sampling method as described above is used. The resulting performance for this gauge-task is shown in Table 5.2.

	T							
Model	0	0.05	0.075	0.100	0.125	0.150	0.175	0.200
C & W-MNIST	100.0%	99.8%	99.8%	99.7%	99.7%	99.6%	99.4%	99.0%
Madry-MNIST	100.0%	99.8%	99.8%	99.7%	99.6%	99.5%	99.2%	98.8%

Table 5.2.: Clean test performance of a classifier in the proposed 2-AFC task with the smoothed decision distribution (Equation (5.2.3)) for different temperature values T.

To increase the robustness of the PC-CMP attack against such a fuzzy decision boundary and random mistakes of the attacked classifier, one can use an ensemble

average over the decision d by sampling it repeatedly m times,

$$\langle d \rangle = \frac{1}{m} \sum^{m} d, \tag{5.10}$$

whereby d is the given by Equation (5.2.3).

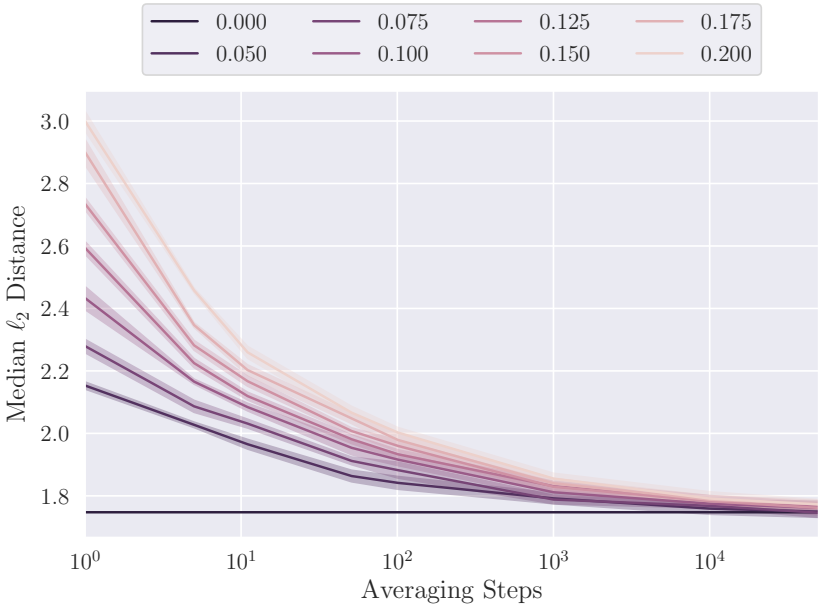


Figure 5.5.: Performance of the PC-CMP attack with decision averaging under a stochastic underlying decision process. The median ℓ_2 distance of the adversarial examples found is shown as a function of averaging steps; the different curves correspond to different strengths T of the simulated stochasticity (the lighter, the higher T and the more stochastic). The deterministic baseline ($T = 0$) is shown in black.

The influence of this averaging process is shown in Figure 5.5. Two things are of interest here: first, the normal PC-CMP attack is not robust against stochastic mistakes, as the median ℓ_2 distance grows strongly for $m = 1$ with increased stochasticity, i.e. increased value of T. The effect is already quite strong even for relatively small levels of stochasticity, which reduces the performance of the classifier in the gauge-task only little (cf. Table 5.2). Second, the performance of the attack can be restored by using the averaged decision $\langle d \rangle$. As expected, the higher the stochasticity is, the more averaging steps are required to reach the deterministic baseline again. To fully restore the performance of the attack against a deterministic model - which does not make mistakes near the decision boundary - more than $m = 1000$ averaging steps are required. This number is too high to realize the averaging process in a

human experiment, as for every averaging step a new human subject is required. Therefore, a smaller number of averaging steps (e.g. 10-100) must be used. Even though this does not lead to optimal performance, the attack's robustness can still be increased significantly. It remains open to see how strong the stochastic effects of a human classifier near the decision boundary are and whether the performed simulation of these in this chapter are accurate.

5.3. Conclusion

The focus of the present chapter was to derive a decision-based adversarial attack that gets by with a small number of queries to the attacked classifier. For this, the mathematical framework of MCMC sampling was utilized. In contrast to other decision-based attacks, the developed method can be easily implemented in a 2-AFC task, which makes it a perfect candidate to be applied in a psychophysical experiment.

The results demonstrate that the proposed method is an efficient tool to find adversarial examples; the ℓ_2 distance of the adversarial examples found is similar to the ones of other attacks. Additionally, the success of the adversarial attack is not very sensitive to the choice of hyperparameters: the attack's performance is almost constant for a broad range of parameters. As the attack is decision-based and does not depend on gradients, it is immune against effects like gradient masking. This means the attack can be applied to a diverse set of classifiers. Experiments show that the model is sensitive to stochastic mistakes of the attacked classifier near the decision boundary. By increasing the number of allowed queries, the robustness of the attack can again partially be recovered. However, for larger levels of stochasticity, this results in an increase in the number of queries by multiple orders of magnitudes, which might make the approach unfeasible for psychophysical experiments. It remains open for future research to validate how realistic the examined level of stochasticity is for experiments with human subjects.

The next future step, as suggested in Chapter 4, is to create a psychophysical experiment to apply the proposed adversarial attack on a human classifier. This will show how severe the stochastic effects on the decision near the decision boundary are for a human and how much it impacts the attack. Furthermore, it allows recording a more precise human baseline on the robustness in an image classification task compared to the experiment proposed in Chapter 4.

6. Improving Robustness Against Natural Corruptions

The last chapter of this thesis is based on a collaborative work conducted with Evgenia Rusak and Lukas Schott [83]. The results of this work are currently under review to be published as a conference paper at the *Conference of Computer Vision and Pattern Recognition* (CVPR). As this is a collaborative effort, only the results and findings that I have actively worked on will be presented here, leaving out parts of the paper that I have not worked on myself.

A different perspective on the robustness of neural networks will be presented now. While the two previous chapters focused on developing methods to asses the robustness of image classifiers against image perturbations, this chapter investigates how networks can be made more robust against distortions. Instead of looking at the robustness against highly artificial adversarial perturbations, in the following the robustness against more realistic and common image corruptions will be examined.

The minimal adversarial perturbations, which were a topic of interest in Chapters 3 and 5, can cause severe mistakes in the decision process of machine learning algorithms, as shown before. But despite their strong effects, the probability of facing them in real-world scenarios is low, which results in low overall risk for most practical applications. Considerably riskier for such applications are the so-called common corruptions. Common corruptions either occur in nature or represent often occurring malfunctions of (image) sensors. They include simple Gaussian or Salt and Pepper noise; natural variations like rain, snow or fog; and compression artifacts such as those caused by JPEG encoding [45]. All of these corruptions do not change the semantic content of the input, and thus, machine learning models should not change their decision-making behavior in their presence.

In the following, two methods to improve the robustness of an image classifier against common corruptions will be analyzed. Their robustness is finally measured

and compared to the current state of the art approaches on the ImageNet-C benchmark dataset (Section 2.1.5).

6.1. Improving Robustness Through Augmented Training

In principle, there are two approaches to improve the robustness of an image classification network: first, one can modify the architecture of the network and second, one can change how the network is trained. In the remainder of this chapter, the focus will solely be on the latter: a fixed network architecture is used and only the way how its weights are trained is changed.

Two methods to increase the robustness will be introduced formally. The first is based on simple Gaussian data augmentation, and can be seen as a baseline. The second combines an adversarial training scheme [33] with the previously introduced noise generators (Chapter 4).

A popular approach to decrease overfitting and to help the network generalize better to previously unseen data is to augment the training dataset by applying randomized manipulations, which do not change the semantics of the image, to all images [71]. The task of increasing the robustness against common corruptions can also be seen as a generalization task to previously unseen samples. Thus, it seems natural to use data augmentation to improve the robustness [19, 22, 28, 29, 31].

In a recent study, Geirhos et al. [28] have examined the robustness that arises from such data augmentation. For this, they trained ImageNet classifiers on a set of image corruptions and tested how robust these classifiers are against unseen corruption types. The results of the study suggest that there is no transferability across different corruptions. However, in their experiments, they used corruptions which are vastly more severe than the ones in the ImageNet-C benchmark (even though those are already quite severe). This observation is backed up by the results of Dodge and Karam [22].

For minimal image distortions, e.g. adversarial examples, it was shown that data augmentation with Gaussian noise improves the (provable) robustness of a recognition model [19]. Gaussian data augmentation has also been used to improve the robustness against the common corruptions in the ImageNet-C benchmark [31, 63]. Conceptually, the study conducted by Gilmer et al. [31] is the closest related to

the work of this chapter. Here, image classifiers were trained solely on images augmented with Gaussian noise - this resulted in a small improvement of the robustness. Additionally, Lopes et al. [63] restricted the Gaussian data augmentation to small patches of the images and observed improved robustness.

6.1.1. Data Augmentation with Gaussian Noise

As stated above, different studies applied data augmentation with Gaussian noise to increase the robustness against common corruption with mixed results. Here, the approach of Gaussian data augmentation is revisited and its efficacy is increased. Concretely, the standard deviation σ of the distribution is treated as a hyper-parameter of the training and its influence on the robustness is measured.

To formally motivate the objective of the training, let $\mathcal{D}$ be the distribution over input pairs $(\mathbf{x}, y)$ on which the image classifier shall be trained. In practice, this distribution is defined by the dataset used. Now, the objective is to find the optimal parameters $\hat{\boldsymbol{\Theta}}$ of an image classifier $N_{\boldsymbol{\Theta}}$, which minimizes the misclassification rate on a dataset with additive Gaussian noise

$$\hat{\boldsymbol{\Theta}} = \arg\min_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\mathbf{x},y\sim\mathcal{D}} \mathop{\mathbb{E}}_{\delta\sim\mathcal{N}(\mathbf{0},\sigma^2)} \left[\mathcal{L}_{\mathrm{CE}}(N_{\boldsymbol{\theta}}(\mathbf{x}+\boldsymbol{\delta}), y)\right]. \tag{6.1}$$

Here, σ is the standard deviation of the Gaussian distribution and the perturbed image $\mathbf{x}+\boldsymbol{\delta}$ is clipped to the valid input range (Section 4.1). To maintain high accuracy on clean data, only 50% of the training samples are augmented with Gaussian noise. In the following, this training method will be called Gaussian Noise Training (GNT).

6.1.2. Data Augmentation with Adversarial Noise

The objective defined above can also be extended to not use Gaussian noise but instead noise from any distribution. Now, instead of using a fixed and predefined distribution, one can further improve the performance by also treating the distribution as a parameterized function one wants to learn. For this, the noise generators derived in Section 4.1 can be used. To increase the robustness, the classifier $N_{\boldsymbol{\Theta}}$ is trained on samples perturbed by the noise generator. These two networks - noise

generator and classifier - are now optimized jointly:

$$\hat{\boldsymbol{\Theta}} = \arg\min_{\boldsymbol{\theta}} \max_{\boldsymbol{\phi}} \mathbb{E}_{\mathbf{x},y\sim\mathcal{D}} \mathbb{E}_{\delta\sim G_{\boldsymbol{\phi}}(\boldsymbol{\delta})} \left[\mathcal{L}_{\mathrm{CE}}(N_{\boldsymbol{\theta}}(\mathbf{x}+\boldsymbol{\delta}), y)\right]. \tag{6.2}$$

For a joint adversarial training, the outer updates of the classifier and inner ones of the generator are applied alternately.

To maintain high classification accuracy on clean samples, every mini-batch is sampled so that it contains 50 % clean data and 50 % perturbed data. The current state of the noise generator is used to distort 30 % of this perturbed data. The remaining 20 % are augmented with randomly chosen noise distributions based on previous states of the generator G, to preserve the already obtained robustness against previous types of noise.

To prevent the noise generator from being stuck in a local minimum, the Adversarial Noise Training (ANT) is halted at regular intervals and a new noise generator is trained from scratch. This noise generator is trained against the current state of the classifier to find a current optimum. The new noise generator replaces the former noise generator in the ANT. Thus, the classifier is trained against a diverse set of noise distributions. This technique has proven crucial to train a robust classifier.

6.2. Experiments & Results

After the two training schemes to improve the robustness against common corruptions have been introduced and motivated, they will both be applied and evaluated in the following experiments.

6.2.1. Training with Gaussian Noise

In the first experiment the Gaussian Noise Training (GNT) will be examined. For this, a pretrained ResNet-50 ImageNet classifier is fine-tuned with Gaussian data augmentation that uses noise from the distribution $\mathcal{N}(\mathbf{0}, \sigma^2)$. The standard deviation σ is then treated as a hyperparameter and varied. Using a single σ value and sampling σ from a set of multiple possible values is compared. In all experiments, the networks are trained until they converge. The resulting accuracies in the ImageNet-C benchmark are shown in Figure 6.1. Here, every black point represents the performance of one model which was fine-tuned with a specific σ. The performance of the baseline without any augmentation is marked by the point at $\sigma = 0$.

Horizontal lines indicate that the model was fine-tuned with Gaussian noise whose standard deviation σ was uniformly sampled from a fixed set for each image. Both the results on the full ImageNet-C benchmark and the results on the ImageNet-C benchmark without the Noise corruptions are shown in the figure, as Gaussian noise itself is part of the latter set. The results in Figure 6.1 show three interesting

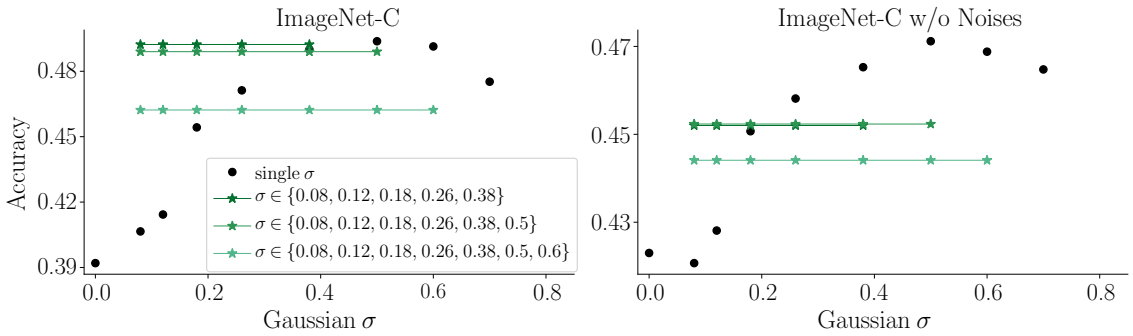


Figure 6.1.: Accuracy on ImageNet-C (left) and ImageNet-C without the Noise corruptions (right) of a ResNet-50 fine-tuned with Gaussian data augmentation of varying standard deviations σ. The networks are trained on Gaussian noise sampled from a distribution with a single σ (black dots) and on distributions where σ is uniformly sampled from different sets (green lines with stars). The baseline performance without any Gaussian noise data augmentation is shown in the bottom left ($\sigma = 0$).

aspects. First, training against Gaussian noise generalizes well against non-noise corruption types of the ImageNet-C benchmark. Therefore, GNT can be seen as a powerful baseline. Compared to the related work of Geirhos et al. [28], Lopes et al. [63] and Gilmer et al. [31] this is surprising as previous studies suggested that training against Gaussian noise either does not yield any [28, 63] or only little [31] improvement. Second, the standard deviation σ is a crucial hyperparameter which needs to be tuned carefully. For the ImageNet-C benchmark the optimal value is about $\sigma = 0.5$ for the ResNet-50 architecture. Third, a single and carefully chosen standard deviation σ is enough and performs better than sampling σ uniformly from a fixed set. Using multiple values for σ even had the opposite effect and reduced the robustness against non-noise corruptions drastically. The results show that the shape of the distribution, from which the noise for the data augmentation is sampled, is important. A more detailed overview of the robustness of the models trained using GNT can be found below in Section 6.2.3.

6.2.2. Training with Learned Noise

In this section, the Adversarial Noise Training (ANT) will be evaluated. For this, the previously used Gaussian distribution is replaced with a distribution approximated by a noise generator network. Since Section 4.1 has already shown that the simple architectures, such as the Spatial-1 generator, can produce more severe noise than a Gaussian distribution, this type of network is used now. In contrast to the chapter above, there is now no requirement to train the network using ES and gradient optimizers like SGD can be used here; therefore, the network size is not that constrained. This allows one to modify the network architecture by adding a convolutional layer - this turned out to improve the performance of the noise generator further. The noise generator is used to generate the most severe noise for a classifier; this type of noise will be called adversarial noise (AN) in the remainder of this chapter.

At first, the performance of this modified noise generator is tested by calculating the required median ℓ_2 distance for the examined ResNet-50 network on ImageNet (ϵ^*_{AN}). This value is then compared to the values for Gaussian (ϵ^*_{GN}) and for uniform noise (ϵ^*_{UN}). These values are displayed in Table 6.1. One can see, that AN is much more effective at fooling the classifier compared to Gaussian and uniform noise as it requires on average a lower ℓ_2 distance.

model	ϵ^*_{GN}	ϵ^*_{UN}	ϵ^*_{AN}
Vanilla RN50	39.0	39.1	16.2

Table 6.1.: Median ℓ_2 perturbation size ϵ^* that is required to misclassify an image for Gaussian (GN), uniform (UN) and adversarial noise (AN). A lower ϵ^* indicates a more severe noise, since on average, a smaller perturbation size is sufficient to fool a classifier.

Next, after the efficiency of AN has been shown, it can be utilized in the ANT scheme, where both noise generator and image classifier are trained simultaneously. In the experiments, the weights of the two networks are updated alternately as described in Section 6.1.2. The networks are again trained until convergence. During the training process, the noise generator was repeatedly reset and trained from scratch to make sure the joint training does not converge towards a sub-optimal local minimum. To better understand how these minima differ, the temporal evolution of

the probability density function $p(\delta)$ of the noise added during the ANT is visualized in Figure 6.2. This shows that the generator converges to different distributions

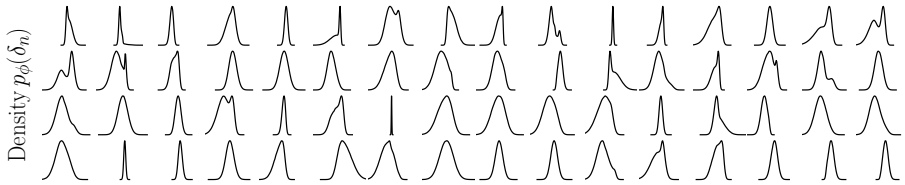


Figure 6.2.: Examples of learned probability densities over the grayscale version of the noise $\boldsymbol{\delta}$ for ImageNet-C during the ANT procedure. Each density corresponds to one local minimum found. Time advances from top left to bottom right in reading direction.

during the ANT procedure. Therefore, the classifier has been trained against a rich variety of distributions. Furthermore, as both the shape and the mean of the distributions change over time, it motivates the increased robustness to different corruptions compared to the networks trained with a fixed noise distribution.

A detailed overview of the robustness of the models trained using ANT can be found in the next section.

6.2.3. Comparison of the Proposed Methods

In the final stage of the experiments, the two proposed methods, GNT and ANT, will be compared. For this, the two best performing configurations of GNT are used: using a single σ value of 0.5 (GNT$\sigma_{0.5}$) and using multiple values (GNT$_{\text{mult}}$). At first, the robustness against Gaussian, uniform and adversarial noise is analyzed in Table 6.2. These results show that both GNT and ANT do not significantly reduce the accuracy on clean images but can strongly improve the robustness against the noises. All in all, the model trained with ANT achieves the strongest robustness level against these noises.

Next, the performance of the three models on the actual ImageNet-C benchmark is tested. They are also compared against several baseline models from the literature. For this, all published results, that are based on a ResNet-50 architecture, are used:

- Shift Inv: The model is modified to enhance the shift-equivariance with respect to the input using anti-aliasing techniques from the domain of signal processing [100].

Training Method	Clean Acc.	ϵ^*_{GN}	ϵ^*_{UN}	ϵ^*_{AN}
Vanilla	**76.1%**	39.0	39.1	16.2
GNT$\sigma_{0.5}$	75.9%	74.8	74.9	30.8
GNT$_{\text{mult}}$	**76.1%**	130.1	130.7	23.2
ANT	76.0%	**136.7**	**137.0**	**127.9**

Table 6.2.: Accuracy on clean data and robustness against different noise distributions measured by the median ℓ_2 perturbation size ϵ. Normal training on clean data (Vanilla) is compared to the proposed Adversarial Noise Training (ANT), to Gaussian Noise Training with a fixed standard deviation (GNT$\sigma_{0.5}$) and to Gaussian Noise Training with multiple values (GNT$_{\text{mult}}$). A higher ϵ indicates a more robust model.

- Patch GN: The model was trained on locally disturbed images: Gaussian noise was applied only to small patches and not the entire image [63].
- SIN+IN: The mode was trained on a combination of stylized images and clean images [28]. The stylized images are obtained by performing a style transfer on clean ImageNet samples with styles from randomly chosen paintings [26].

The resulting average accuracies on ImageNet-C are displayed in Table 6.3. In this table again two values per model are reported: the accuracy on the full ImageNet-C set with all distortions and the one on all distortions except noise category. A detailed overview which shows the accuracy per distortion type and not only the average can be found in Table C.4 in Appendix C. The results on ImageNet-C are striking: A very simple baseline, namely a model trained with Gaussian noise data augmentation beats all previous baselines. The GN$\sigma_{0.5}$ surpasses the current state of the art SIN+IN not only on the Noise categories but also on almost all other corruptions. The ANT model produces the best results on full ImageNet-C with and without noises. Thus, it is slightly superior to Gaussian data augmentation.

6.3. Conclusion

So far, multiple studies tried to apply simple data augmentation to improve the robustness of an image classifier. The results of these studies range from a failure to generalize from one type of distortion to another distortion type [29] to marginal increases of the robustness [31, 63]. In this chapter, it was demonstrated that by training on both clean samples as well as perturbations from carefully tuned

Training	IN Clean Acc. [%]	IN-C Acc. [%]	IN-C w/o Noises Acc. [%]
Vanilla	76.1	39.2	42.3
Shift Inv [100]	**77.0**	41.4	44.2
Patch GN [63]	76.0	(43.6)	43.7
SIN+IN [28]	74.6	45.2	46.6
GNT_{mult}	76.1	(49.2)	45.2
$\text{GNT}\sigma_{0.5}$	75.9	(49.4)	47.1
ANT	76.0	(**51.1**)	**47.7**

Table 6.3.: Average accuracy on clean data and accuracies on ImageNet-C and ImageNet-C without the noise categories in percent (higher is better). The results obtained by the means of Gaussian noise data augmentation (GNT) and with Adversarial Noise Training (ANT) are compared to several baselines. Numbers in brackets indicate scenarios where a corruption from the test set was used during training. A detailed overview of the accuracies per corruption can be found in Table C.4 in Appendix C.

Gaussian noise distributions, one can already reach or even surpass the current state-of-the-art defense method against common corruptions. These results can be improved further by designing an adversarial training scheme in which the distribution of worst-case uncorrelated noise is learned. This training scheme outperforms the already strong baseline defense. Furthermore, this method can be combined with different defense methods, such as the stylization training [28], which has been the previous state-of-the-art results in the ImageNet-C benchmark.

Even though the proposed Adversarial Noise Training can reach a new overall state-of-the-art result, it still underperforms on a few corruption types such as Elastic or Fog. This shows that there is still room for future improvements for the ImageNet-C benchmark. Possible extensions of the approach include the exploration of noise generators that can produce perturbations with varying correlation lengths or to not restrict the perturbations to be image-independent but allow perturbations to depend on the images.

7. Conclusion

In this thesis, the robustness of artificial neural networks was examined. To do so, on the one hand, tools for highlighting the missing robustness of image classification networks were designed. These tools were created such that they might be used in future studies not only to assess the robustness of a computer vision algorithm but possibly also to measure the gold standard of robustness, which is achieved by humans. On the other hand, ways to increase the robustness of neural networks were explored and evaluated.

The first method to assess the robustness of an image classifier uses image independent noise. Different ways to generate this noise were proposed and evaluated. It was demonstrated that CNNs are by far more susceptible to this generated adversarial noise than to baselines, e.g. Gaussian noise.

The second method presented is able to create image dependent adversarial perturbations. The attack was created such that it poses as few requirements as possible to the classifier, such that it can be used to evaluate a broad range of these. It was demonstrated that for CNNs the attack requires only a few hundred queries to find imperceptible adversarial examples which are comparable to the results of gradient-based attacks.

To obtain a better understanding of defense strategies against perturbations and corruptions, a defense strategy proposed by Liu et al. [62], which utilizes a Bayesian formalism of neural networks, was carefully evaluated. It was demonstrated by experiments that the improvements of robustness claimed by the original study are not due to a powerful defense but rather too weak attack methods used in the study's evaluation. To solve this problem an adversarial attack customized for the proposed defense was created in this thesis. The experiments showed that the defense proposed by Liu et al. [62] does not work against the new attack. The defense shows no improvement compared to a baseline without any defense.

Finally, a new strategy to increase the robustness of an image classification network against common image corruptions was proposed. It was demonstrated that already simple data augmentation can reach state-of-the-art results if applied properly. It was also shown that an adversarial training scheme can improve the data augmentation and, thus, increase the robustness even further.

Future work to built upon the efficient and powerful tools to measure the robustness of arbitrary image classifiers against perturbations, which were developed in this thesis, could assess the true robustness of humans in image classification tasks. This can be done by applying the proposed adversarial attacks in a psychophysical experiment. The results of such a study would provide a clear gold standard in terms of the robustness for image classification. The robustness of any computer vision algorithm can then be compared against this standard to see more clearly how large the gap between the robustness of the current state-of-the-art in computer vision and biological visual systems is. After the gap is known, research can be directed more carefully to decrease it. The disappearance of it could allow for a more confident application of computer vision algorithms also in security relevant tasks. Since even though they would still be susceptible to perturbations, they would not be more susceptible than humans - however, they would possibly be susceptible to a different type of distortion. Reaching this level of confidence about both the accuracy as well as the robustness of computer vision algorithms will finally allow the large-scale adaptation in industry.

Appendices

A. Algorithms

Algorithm A.1: General Proposal Distribution for CMP

Input: Current state of MCMC-P $\mathbf{s}_t \in \mathbb{R}^n$, Number of projection steps K, Stimulus generating function $h : \mathbb{R}^n \mapsto \mathbb{S}$, Objective function f

Result: Sample $\mathbf{s}_{t+1}$ from proposal distribution

```
// Get objective value for current state
ϵ ← f(h(s_t));
for t ← 1 to K do
    // Generate state perturbation w/ same objective value as current state
    p_0 ~ N(0, σ^2)
    for i ← 1 to K do
        if f(h(s_{t-1} + p_i)) < ϵ then
            // Increase state (c_+ > 1)
            p_i ← c_+ · p_{i-1};
        else
            // Decrease state (c_- < 1)
            p_i ← c_- · p_{i-1};
        end
    end
end
return s_{t+1}
```

B. Figures

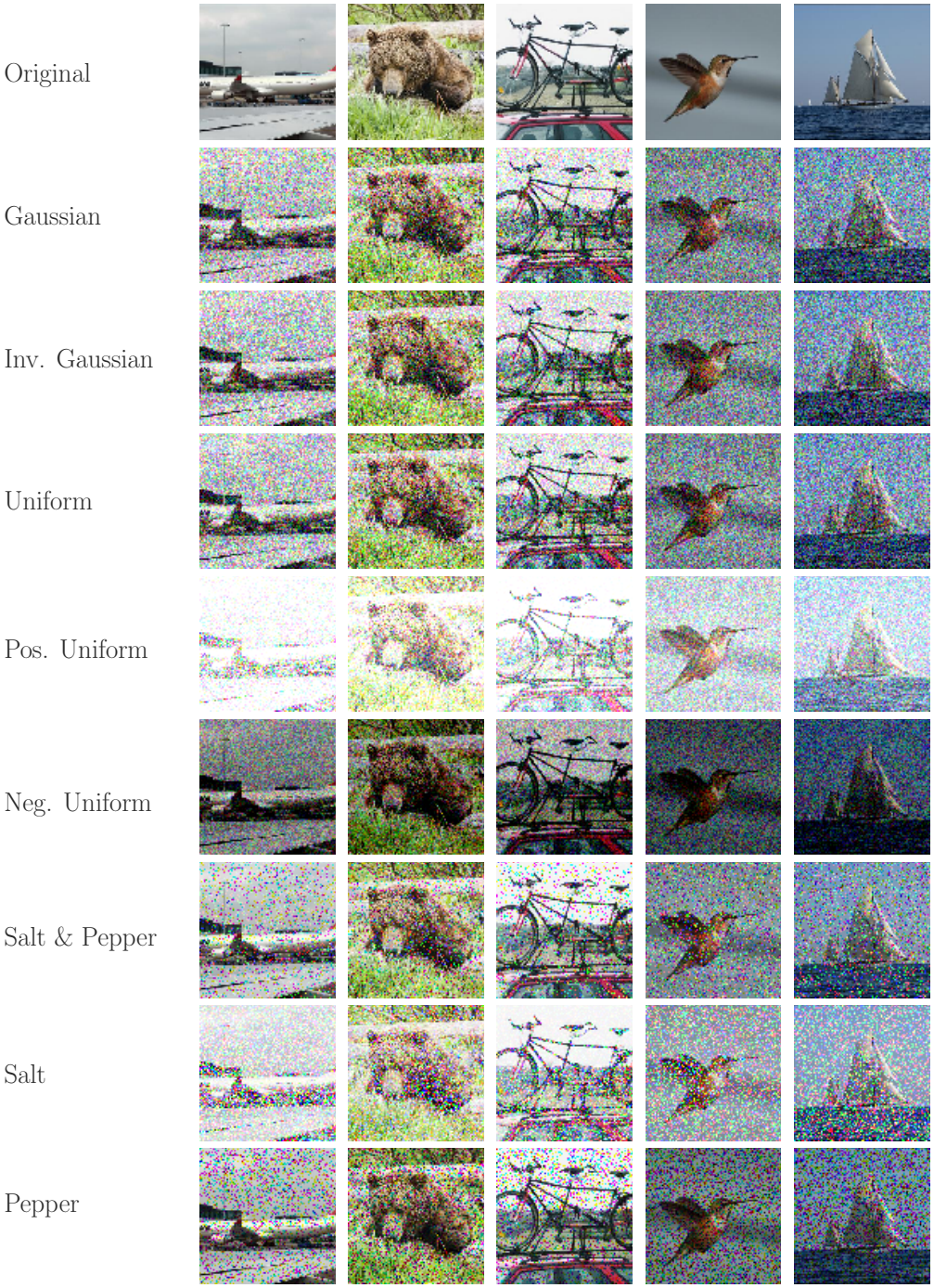


Figure B.1.: Images of the ImageNet-16 dataset perturbed with noise sampled from different baseline distributions. The strength of the displayed perturbations matches the measured ϵ^* distance against a ResNet-50 classifier. This distance corresponds to the distance at which the classifier's accuracy is reduced to 50 %.

Figure B.2.: Images of the CIFAR-10 dataset perturbed with with noise sampled from different baseline distributions. The strength of the displayed perturbations matches the measured ϵ^* distance against a ResNset-18 classifier.

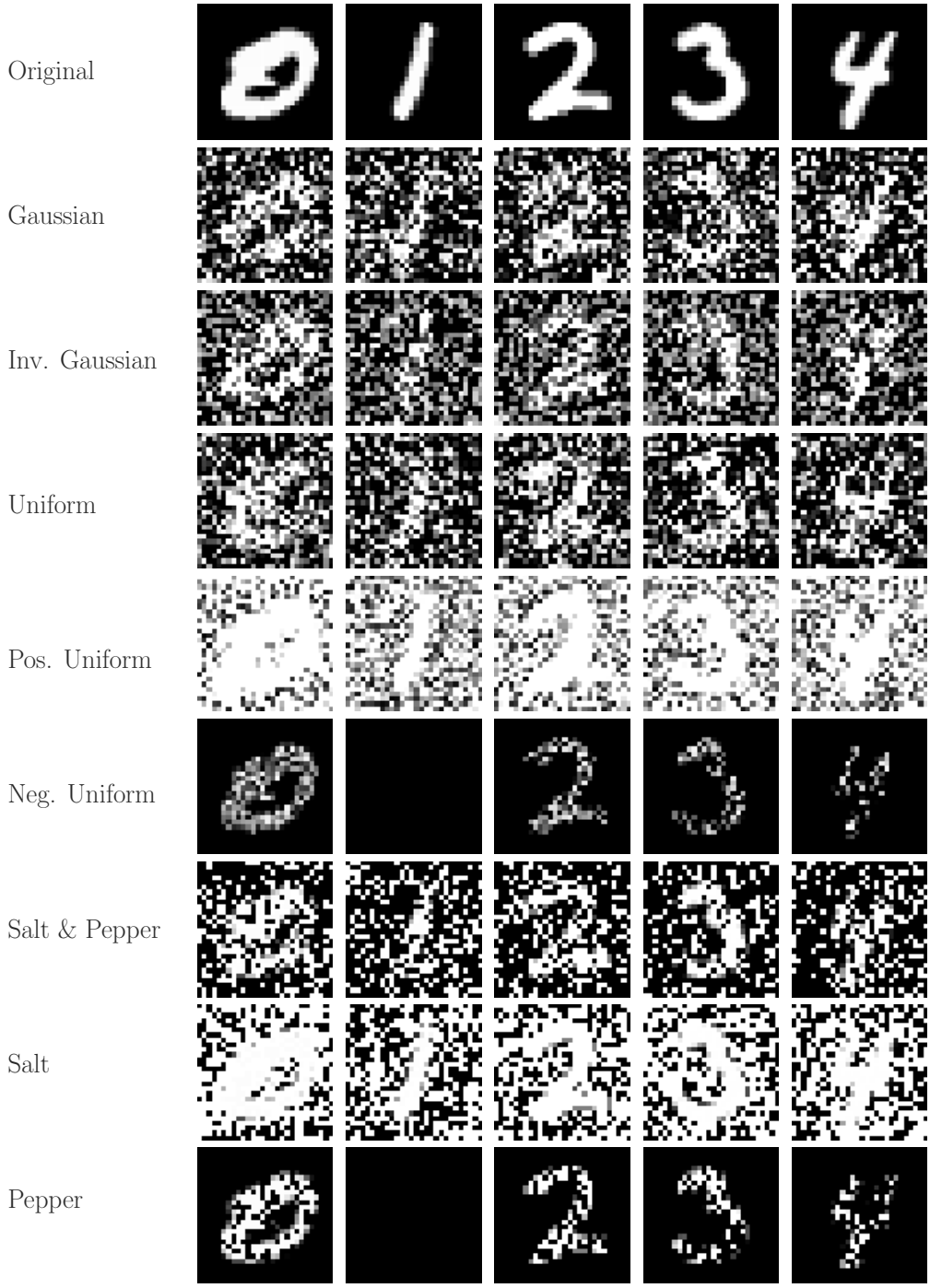
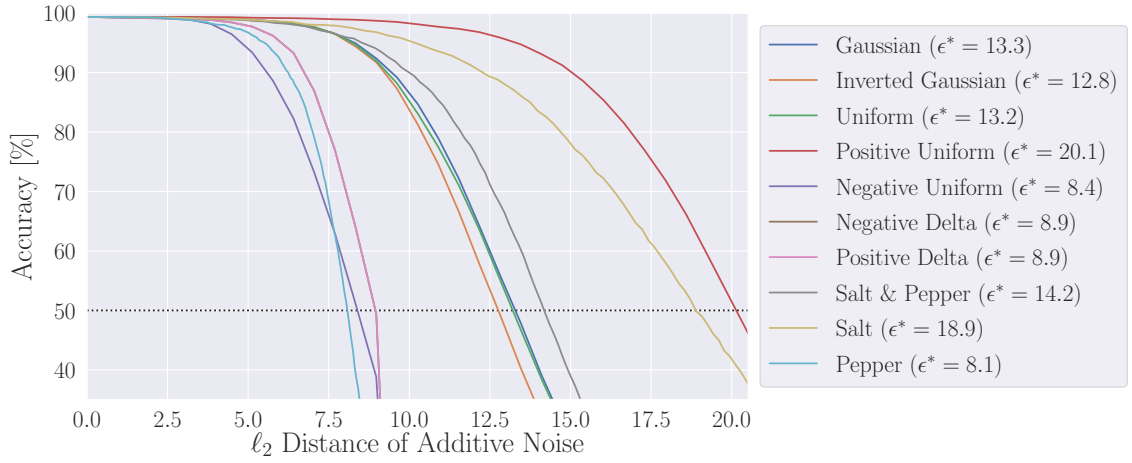
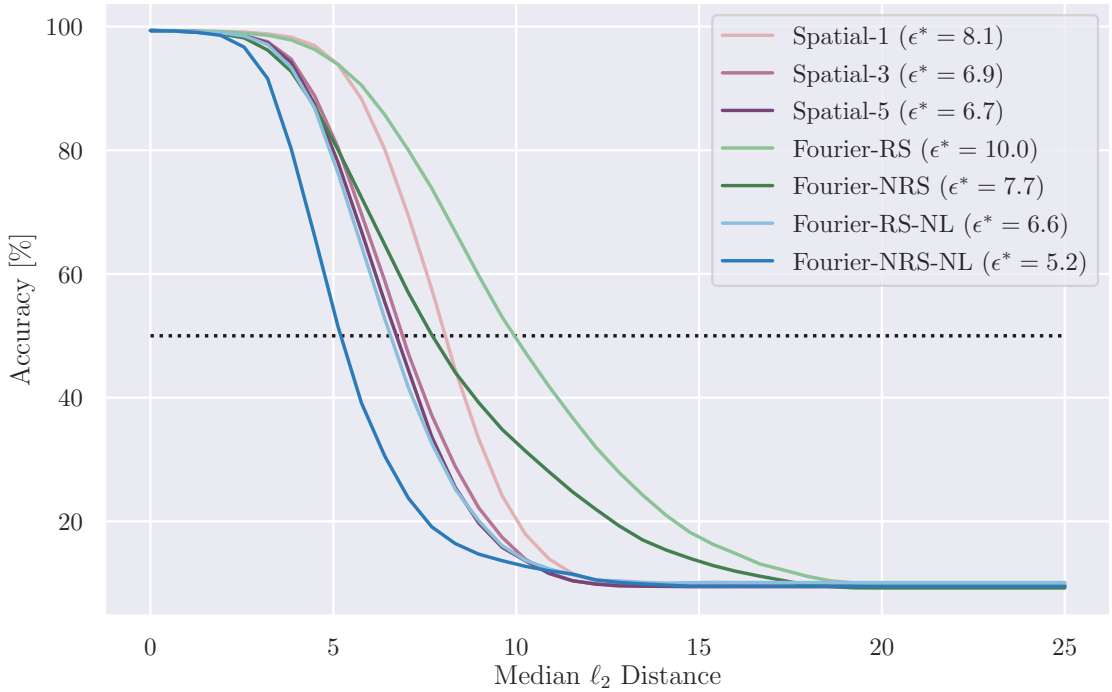


Figure B.3.: Images of the MNIST dataset perturbed with noise sampled from different baseline distributions. The strength of the displayed perturbations matches the measured ϵ^* distance against a C & W classifier.

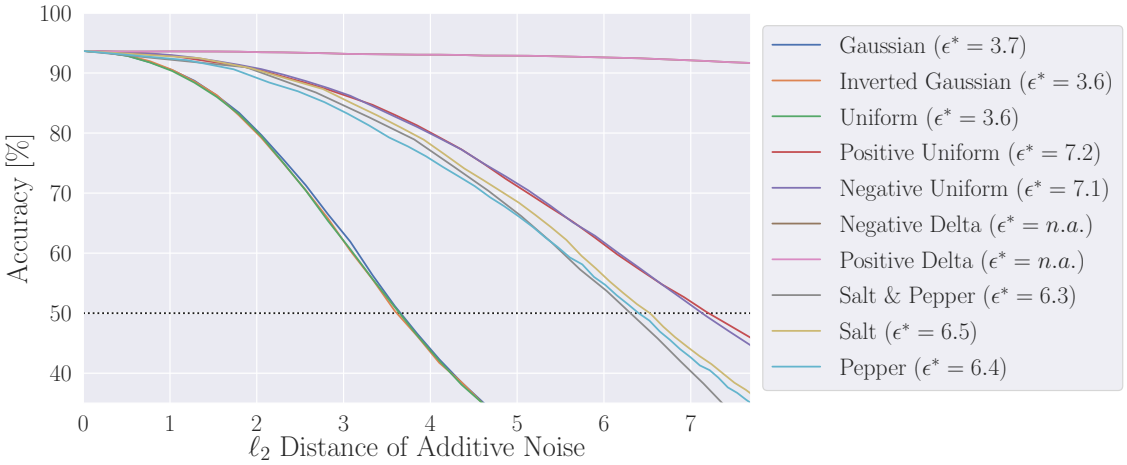


(a) Accuracy as a function of the ℓ_2 distance of the additive noise for different noise distributions.

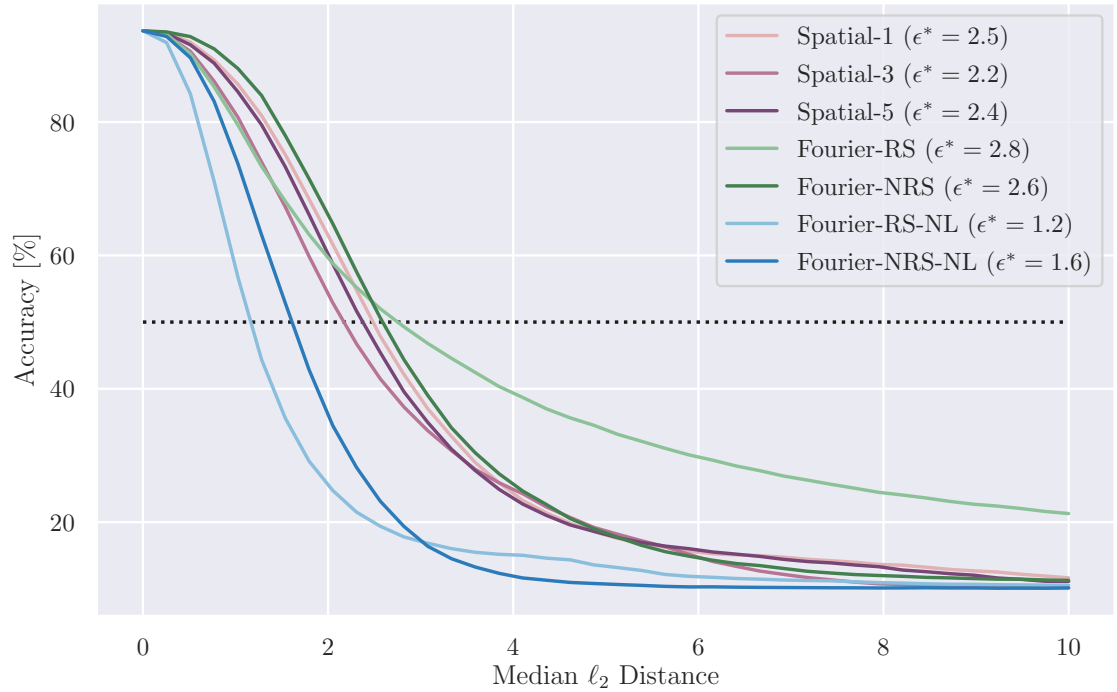


(b) Accuracy as a function of the ℓ_2 distance of the additive noise for different noise generators trained with CMA-ES.

Figure B.4.: Accuracy of the C&W model on noisy images from the MNIST dataset as a function of the ℓ_2 distance of the additive noise for fixed noise distributions (a) and learned noise generators (b).



(a) Accuracy as a function of the ℓ_2 distance of the additive noise for different noise distributions.



(b) Accuracy as a function of the ℓ_2 distance of the additive noise for different noise generators trained with CMA-ES.

Figure B.5.: Accuracy of the ResNet18 model on noisy images from the CIFAR-10 dataset as a function of the ℓ_2 distance of the additive noise for fixed noise distributions (a) and learned noise generators (b).

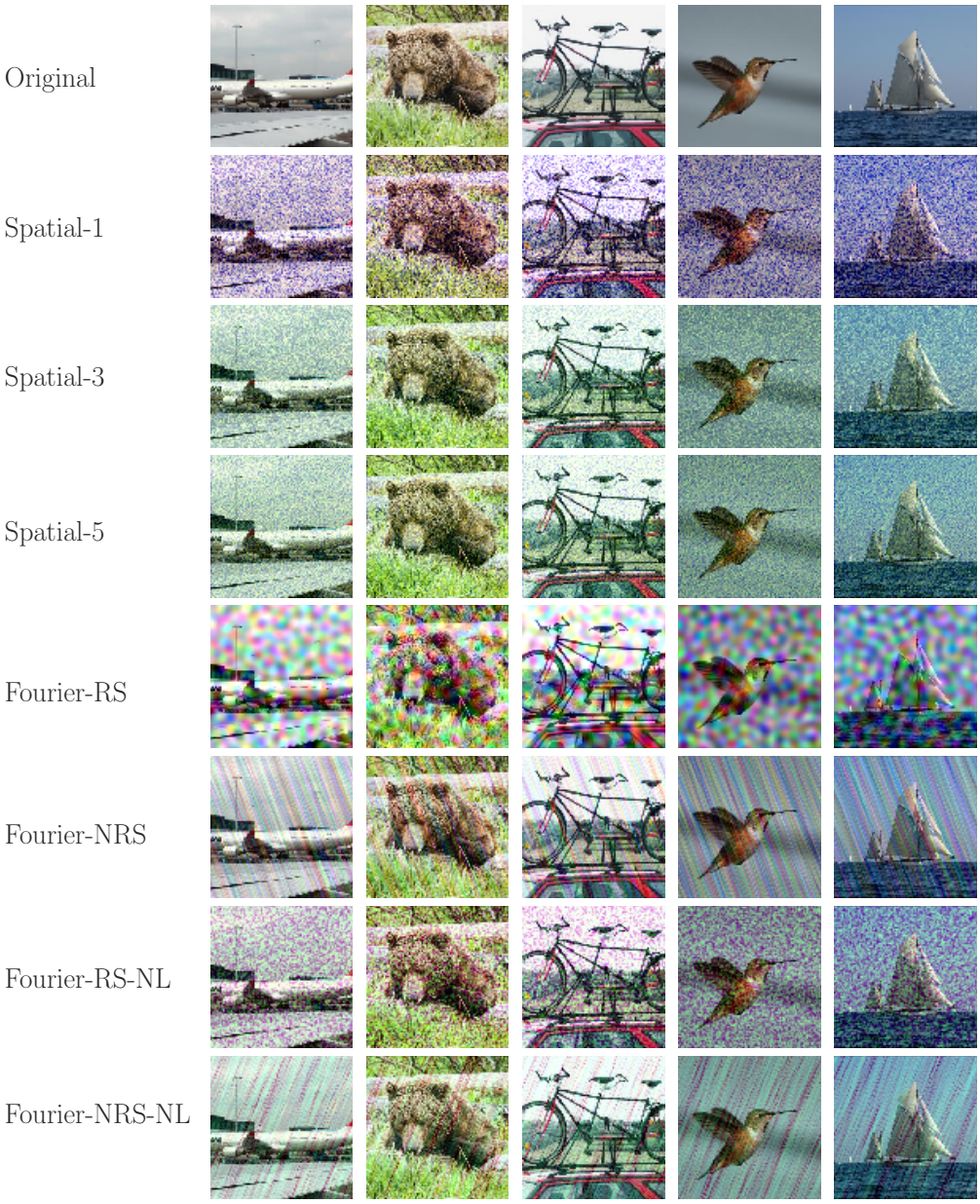


Figure B.6.: Images of the ImageNet-16 dataset perturbed with adversarial noise trained against a ResNet-50 classifier. The strength of the perturbation matches the measured ϵ^* value for this classifier.

Figure B.7.: Images of the CIFAR-10 dataset perturbed with adversarial noise trained against a ResNset-18 classifier. The strength of the perturbation matches the measured ϵ^* value for this classifier.

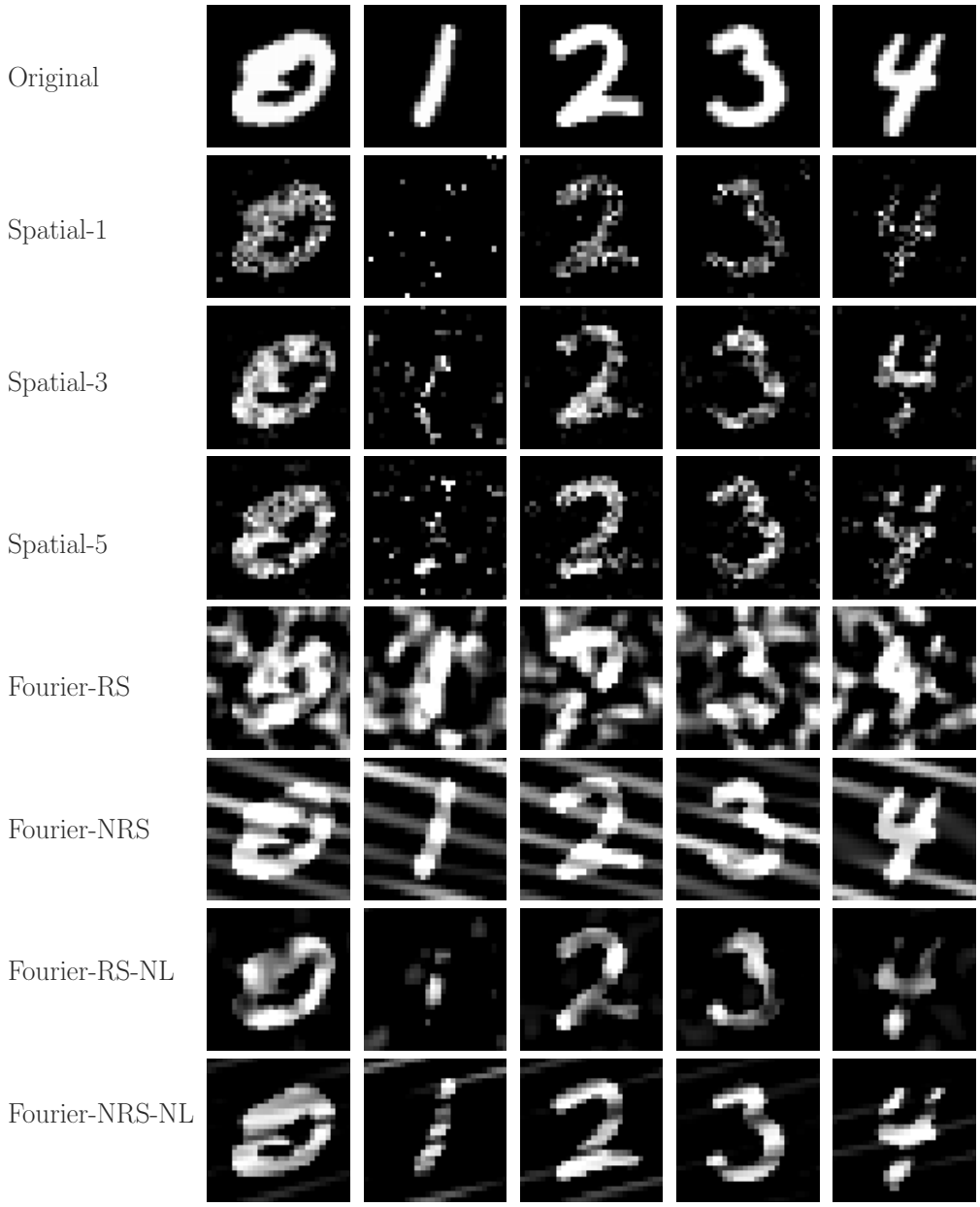


Figure B.8.: Images of the MNIST dataset perturbed with adversarial noise trained against a C & W classifier. The strength of the perturbation matches the measured ϵ^* value for this classifier.

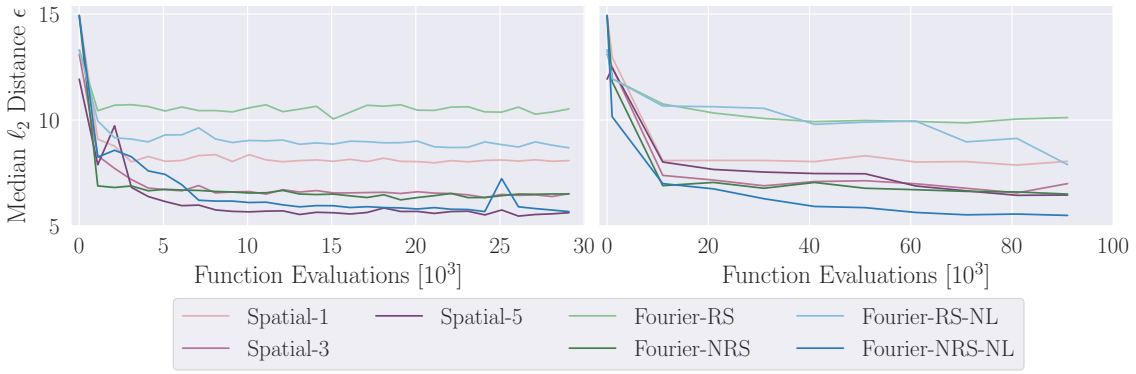


Figure B.9.: Comparison between the training progress of noise generators trained with SGD (left) and CMA-ES (right) against the C&W model on MNIST. The training progress is measured as the required median ℓ_2 distance to reduce the classifier's accuracy to 50% as a function of the function evaluations.

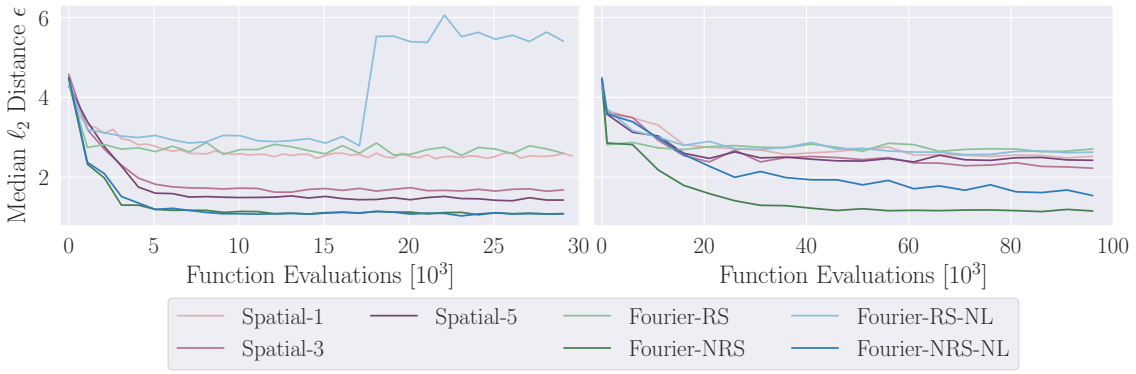


Figure B.10.: Comparison between the training progress of noise generators trained with SGD (left) and CMA-ES (right) against the ResNet-18 model on CIFAR-10. The training progress is measured as the required median ℓ_2 distance to reduce the classifier's accuracy to 50% as a function of the function evaluations.

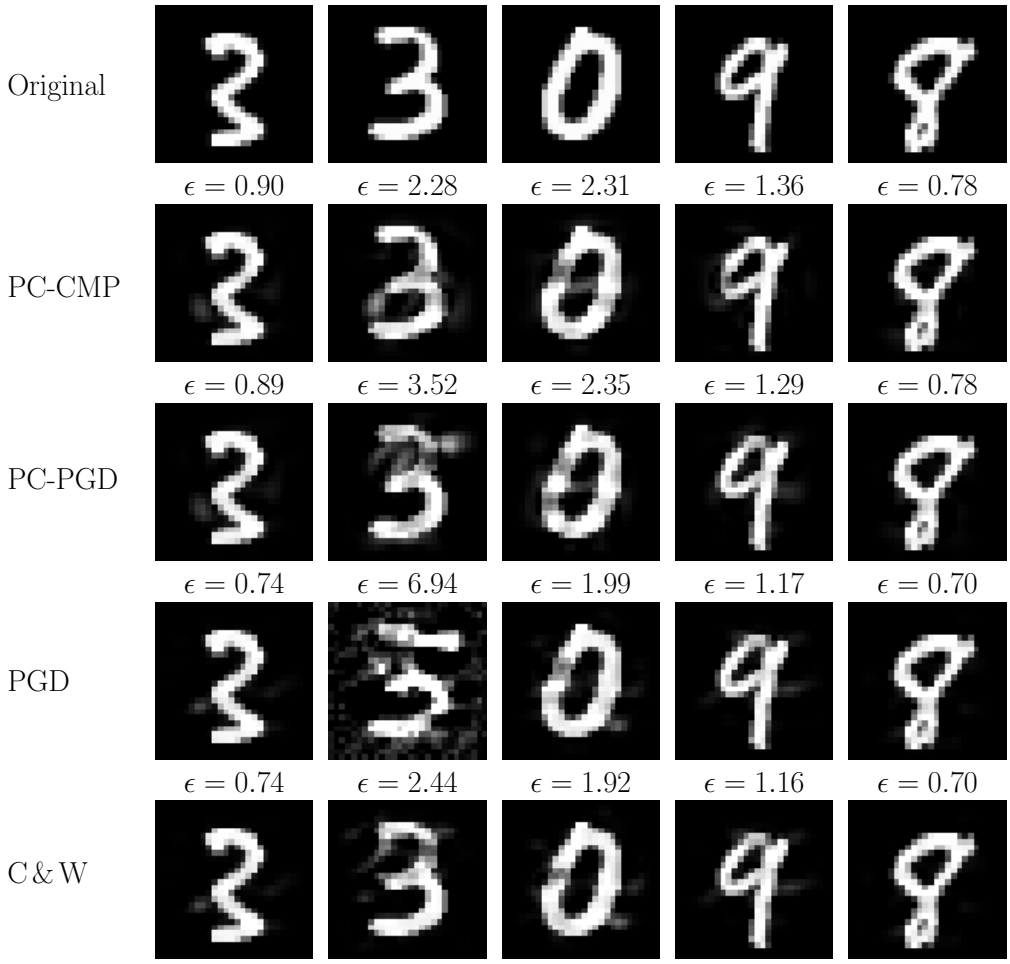


Figure B.11.: Adversarial examples found for the C & W model on MNIST by the PC-CMP, (PC-)PGD and C & W attack.

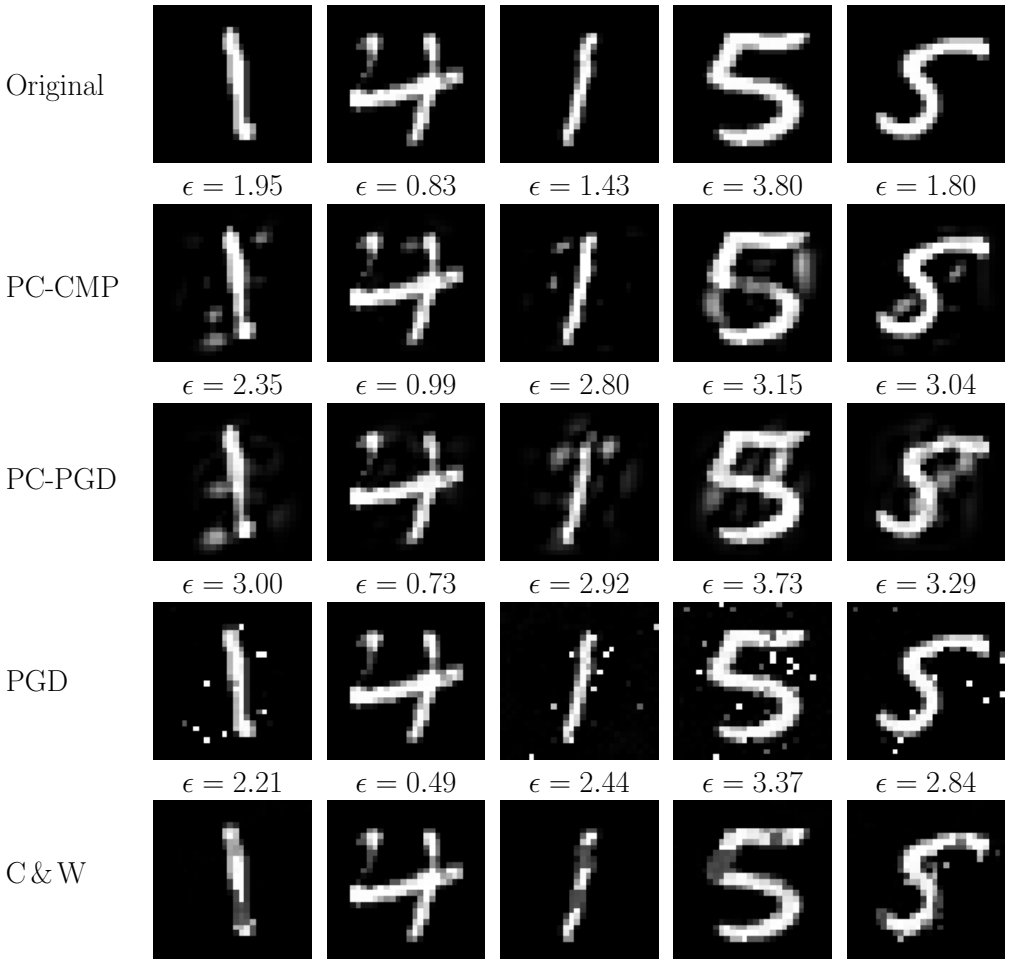


Figure B.12.: Adversarial examples found for the adversarially trained Madry model on MNIST by the PC-CMP, (PC-)PGD and C & W attack.

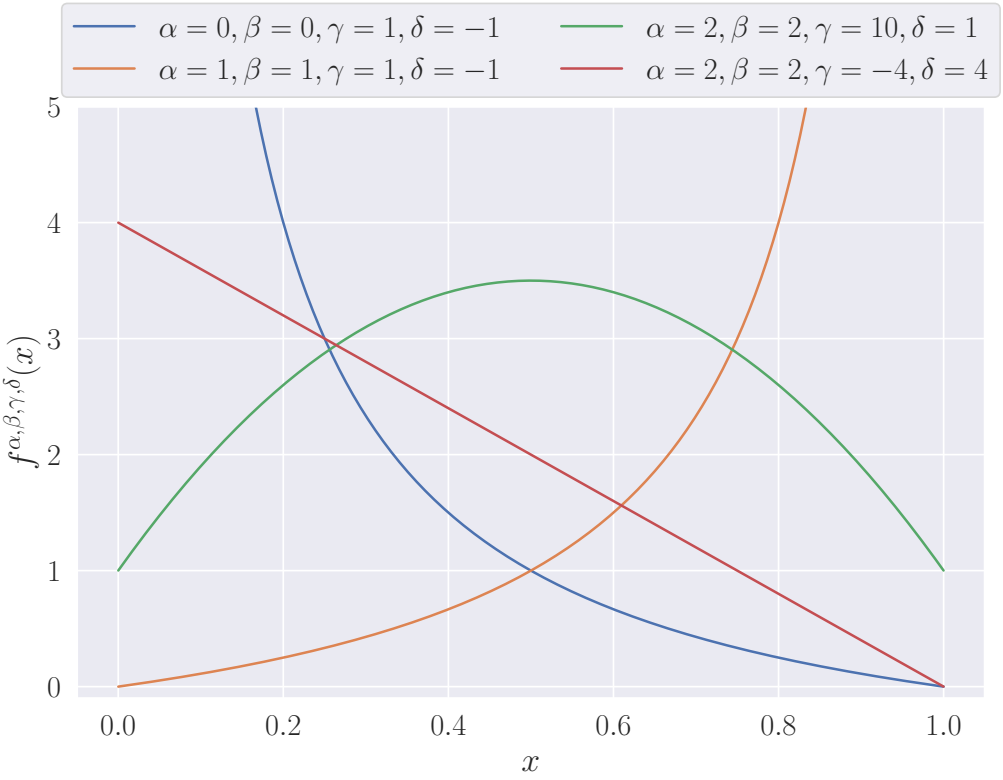


Figure B.13.: Illustration of the diversity of the power function used in the Fourier-based noise generators described in Equation (4.8). The function is shown for four different value-tuples of the parameters α, β, γ and δ.

C. Tables

Parameter	Value
Batch Size	100 (60 for ResNet-50)
Learning Rate	10^{-2}
Momentum	0.5

Table C.1.: Parameters of the SGD (Stochastic Gradient Descent) optimizer used to train the Spatial-K and Fourier-based generators against the ResNet-50 model on ImageNet-16, the Madry and C&W model on MNIST, and the ResNet-18 on CIFAR-10.

Parameter	Value
Batch Size	100 (60 for ResNet-50)
Search Variance σ^0	0.2
Population Size λ	10
Number of Considered Members in Update μ	5

Table C.2.: Parameters of the CMA-ES (Covariance Matrix Adaptation Evolution Strategy) optimizer used to train the Spatial-K and Fourier-based generators in the gradient-free scenario against the ResNet-50 model on ImageNet-16, the Madry and C&W model on MNIST, and the ResNet-18 on CIFAR-10.

Parameter	Value
Principal Components (PCs) n	50
Outer Loop Iterations s	6
MCMC Steps N	350
MCMC Step Size λ	0.20

Table C.3.: Parameters of the PC-CMP (Constrained MCMC with People using Principal Components) attack used to attack the Madry and C&W model for the MNIST dataset.

		Noise			Blur			
model	mean	Gauss	Shot	Impulse	Defocus	Glass	Motion	Zoom
Vanilla RN50	39	29	27	24	39	27	39	36
Shift Inv	42	36	34	30	40	29	38	39
Patch GN	44	45	43	42	38	26	39	38
SIN+IN	45	41	40	37	43	32	45	36
Speckle	46	55	58	49	43	32	40	36
GNT_{mult}	49	**67**	65	**64**	43	33	41	37
$\text{GNT}\sigma_{0.5}$	49	58	59	57	**47**	**38**	43	**42**
ANT	51	65	**66**	**64**	**47**	37	43	40
ANT+SIN	**52**	64	65	63	46	37	**46**	39

	Weather				Digital			
model	Snow	Frost	Fog	Bright	Contrast	Elastic	Pixel	JPEG
Vanilla RN50	29	38	46	68	39	45	45	53
Shift Inv	36	40	48	68	42	45	49	57
Patch GN	45	39	**54**	67	39	**52**	47	56
SIN+IN	41	42	47	67	43	50	56	58
Speckle	55	41	46	68	41	47	49	58
GNT_{mult}	**67**	42	45	68	41	48	50	60
$\text{GNT}\sigma_{0.5}$	58	44	44	68	39	50	55	**62**
ANT	65	46	44	**70**	43	49	55	**62**
ANT+SIN	64	**46**	50	69	**48**	50	**57**	57

Table C.4.: Average accuracy over 5 severities of common corruptions on the ImageNet-C benchmark dataset in percent obtained by different models; higher is better. Best values are highlighted in bold font.

D. Derivation of C-MCMC-P

In the following, a detailed derivation of the objective given in Equation 5.5 will be presented. The initial objective used in CMP (Constrained MCMC with People) is given by

$$\min_{\mathbf{x}} f(\mathbf{x}) \quad \wedge \quad \log p_C(\mathbf{x}|c) <= \xi, \tag{D.1}$$

where f is a scalar function which shall be minimized and $p_c(\mathbf{x}|c)$ is the class-dependent probability of a sample $\mathbf{x}$. Using the Lagrangian formalism this can be rewritten as

$$\min_{\mathbf{x}} f(\mathbf{x}) \quad \wedge \quad \log p_C(\mathbf{x}|c) <= \xi \tag{D.2}$$

$$= \min_{t,\mathbf{x}} t \quad \text{s.t.} \quad f(\mathbf{x}) = t \quad \wedge \quad \log p_C(\mathbf{x}|c) <= \xi \tag{D.3}$$

$$= \min_{t,\mathbf{x}} \max_{\mu \geq 0} t + \mu \left(\log p_C(\mathbf{x}|c) - \xi\right) \quad \text{s.t.} \quad f(\mathbf{x}) = t \tag{D.4}$$

$$= \underbrace{\min_{t} \max_{\mu \geq 0} t - \mu\xi + \underbrace{\max_{\substack{\mathbf{x} \\ \text{s.t. } f(\mathbf{x})=t}} \mu \left[\log p_C(\mathbf{x}|c)\right]}_{\text{Inner Loop (MCMC-P)}}}_{\text{Outer Loop}}, \tag{D.5}$$

whereby μ is the newly introduced Lagrangian multiplier which ensures that the constraint is fulfilled.

Bibliography

[1] Moustafa Alzantot, Yash Sharma, Supriyo Chakraborty, Huan Zhang, Cho-Jui Hsieh, and Mani B Srivastava. Genattack: Practical black-box attacks with gradient-free optimization. In *Genetic and Evolutionary Computation Conference*, pages 1111–1119. ACM, 2019.

[2] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing Robust Adversarial Examples. In *International Conference on Machine Learning*, pages 284–293, 2018.

[3] Anne Auger, Nikolaus Hansen, JM Perez Zerpa, Raymond Ros, and Marc Schoenauer. Experimental Comparisons of Derivative Free Optimization Algorithms. In *International Symposium on Experimental Algorithms*, pages 3–15. Springer, 2009.

[4] David Barber and Christopher M Bishop. Ensemble learning in Bayesian neural networks. *Nato ASI Series F Computer and Systems Sciences*, 168: 215–238, 1998.

[5] Av A Barker. Monte carlo calculations of the radial distribution functions for a proton-electron plasma. *Australian Journal of Physics*, 18(2):119–134, 1965.

[6] Battista Biggio, Igino Corona, Davide Maiorca, and Blaine Nelson. Evasion Attacks against Machine Learning at Test Time. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 387–402, 2013.

[7] Charles Blundell, Adam Sanborn, and Thomas L. Griffiths. Look-Ahead Monte Carlo with People. In *Annual Meeting of the Cognitive Science*, 2012.

[8] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight Uncertainty in Neural Network. In *International Conference on Machine Learning*, pages 1613–1622, 2015.

[9] Adith Boloor, Xin He, Christopher Gill, Yevgeniy Vorobeychik, and Xuan Zhang. Simple physical adversarial examples against end-to-end autonomous driving models. *arXiv preprint arXiv:1903.05157*, 2019.

[10] Avishek Joey Bose and Parham Aarabi. Adversarial attacks on face detectors using neural net based constrained optimization. In *2018 IEEE 20th International Workshop on Multimedia Signal Processing (MMSP)*, pages 1–6. IEEE, 2018.

[11] Y Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in vision algorithms. In *Proc. International Conference on Machine learning (ICML'10)*, volume 28, 2010.

[12] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambride University Press, 2009. ISBN 9780521833783.

[13] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.

[14] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. *AISec 2017 - Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, co-located with CCS 2017*, pages 3–14, 2017.

[15] Nicholas Carlini and David Wagner. Towards Evaluating the Robustness of Neural Networks. *arXiv preprint arXiv:1608.04644v2*, 2017.

[16] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, and Aleksander Madry. On evaluating adversarial robustness. *arXiv preprint arXiv:1902.06705*, 2019.

[17] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223, 2011.

[18] Andrew L. Cohen and Michael G. Ross. Exploring Mass Perception With Markov Chain Monte Carlo. *Journal of Experimental Psychology: Human Perception and Performance*, 35(6):1833–1844, 2009.

[19] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified Adversarial Robustness via Randomized Smoothing. In *International Conference on Machine Learning*, pages 1310–1320, 2019.

[20] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[21] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[22] Samuel Dodge and Lina Karam. A study and comparison of human and deep learning recognition performance under visual distortions. In *2017 26th international conference on computer communication and networks (ICCCN)*, pages 1–7. IEEE, 2017.

[23] Marin Dujmović, Gaurav Malhotra, and Jeffrey S Bowers. Humans cannot decipher adversarial images: Revisiting Zhou and Firestone (2019). In *Conference on Cognitive Computational Neuroscience*, 2019.

[24] Gamaleldin Elsayed, Shreya Shankar, Brian Cheung, Nicolas Papernot, Alexey Kurakin, Ian Goodfellow, and Jascha Sohl-Dickstein. Adversarial examples that fool both computer vision and time-limited humans. In *Advances in Neural Information Processing Systems*, pages 3910–3920, 2018.

[25] Ingo Fruend and Elee Stalker. Human sensitivity to perturbations constrained by a model of the natural image manifold. *Journal of vision*, 18(11):20, 2018.

[26] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016.

[27] Robert Geirhos, David H J Janssen, Heiko H Schütt, Jonas Rauber, Matthias Bethge, and Felix A Wichmann. Comparing deep neural networks against humans: object recognition when the signal gets weaker. *arXiv preprint arXiv:1706.06969*, 2017.

[28] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A Wichmann, and Wieland Brendel. ImageNet-trained CNNs are biased to-

wards texture; increasing shape bias improves accuracy and robustness. *arXiv preprint arXiv:1811.12231*, 2018.

[29] Robert Geirhos, Carlos R M Temme, Jonas Rauber, Heiko H Schütt, Matthias Bethge, and Felix A Wichmann. Generalisation in humans and deep neural networks. In S Bengio, H Wallach, H Larochelle, K Grauman, N Cesa-Bianchi, and R Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 7538–7550. Curran Associates, Inc., 2018.

[30] Justin Gilmer and Dan Hendrycks. A Discussion of 'Adversarial Examples Are Not Bugs, They Are Features': Adversarial Example Researchers Need to Expand What is Meant by 'Robustness'. *Distill*, 2019.

[31] Justin Gilmer, Nicolas Ford, Nicholas Carlini, and Ekin Cubuk. Adversarial Examples Are a Natural Consequence of Test Error in Noise. In *International Conference on Machine Learning*, pages 2280–2289, 2019.

[32] Gabriel Goh. A Discussion of 'Adversarial Examples Are Not Bugs, They Are Features': Two Examples of Useful, Non-Robust Features. *Distill*, 2019.

[33] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[34] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. ISBN 9784048930628.

[35] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. *arXiv preprint arxiv:1412.6572v3*, 2014.

[36] David Ha. Evolving Stable Strategies. *blog.otoro.net*, 2017.

[37] Bruce C. Hansen and Robert F. Hess. On the effectiveness of noise masks: Naturalistic vs. un-naturalistic image statistics. *Vision Research*, 2012.

[38] Nikolaus Hansen. The CMA Evolution Strategy : A Tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[39] Nikolaus Hansen and Andreas Ostermeier. Adapting Arbitrary Normal Mutation Distributions in Evolution Strategies : The Covariance Matrix Adaptation. In *IEEE iInternational Conference on Evolutionary Computation*, 1996.

[40] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[41] Jamie Hayes and George Danezis. Learning Universal Adversarial Perturbations with Generative Models. *arXiv preprint arXiv:1708:05207*, pages 1–13, 2017.

[42] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[43] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016.

[45] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, 2019.

[46] John R Hershey and Peder A Olsen. Approximating the Kullback Leibler divergence between Gaussian mixture models. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 4, pages IV–317. IEEE, 2007.

[47] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[48] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. *arXiv preprint arXiv:1905.02175*, 2019.

[49] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[50] Aleksei Grigorevich Ivakhnenko and Valentin Grigorevich Lapa. Cybernetic predicting devices. Technical report, School of Electrical and Computer Engineering at Purdue University, 1966.

[51] Robert A Jacobs and Christopher J Bates. Comparing the Visual Representations and Performance of Humans and Deep Neural Networks. *Current Directions in Psychological Science*, 28(1):34–39, 2019.

[52] Harini Kannan, Alexey Kurakin, and Ian Goodfellow. Adversarial logit pairing. *arXiv preprint arXiv:1803.06373*, 2018.

[53] Robert E. Kass, W. R. Gilks, S. Richardson, and D. J. Spiegelhalter. Markov Chain Monte Carlo in Practice. *Journal of the American Statistical Association*, page 1645, 1997.

[54] Alexander Khintchine. Korrelationstheorie der stationären stochastischen Prozesse. *Mathematische Annalen*, 109(1):604–615, 1934.

[55] Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pages 2575–2583, 2015.

[56] Alex Krizhevsky, Geoffrey Hinton, and others. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[57] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[58] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[59] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, and others. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[60] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 740–755. Springer, 2014.

[61] Xuanqing Liu, Minhao Cheng, Huan Zhang, and Cho-Jui Hsieh. Towards robust neural networks via random self-ensemble. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 369–385, 2018.

[62] Xuanqing Liu, Yao Li, Chongruo Wu, and Cho-Jui Hsieh. Adv-BNN: Improved Adversarial Defense through Robust Bayesian Neural Network. In *ICLR 2019*, 2019.

[63] Raphael Gontijo Lopes, Dong Yin, Ben Poole, Justin Gilmer, and Ekin D Cubuk. Improving Robustness Without Sacrificing Accuracy with Patch Gaussian Augmentation. *arXiv preprint arXiv:1906.02611*, 2019.

[64] Robert Duncan Luce. Detection and recognition. In R. D. Luce, R. R. Bush, and E. Galanter, editors, *Handbook of Mathematical Psychology*, chapter Detection. Wiley, 1963.

[65] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[66] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, pages 1–28, 2018.

[67] Jay B. Martin, Thomas L. Griffiths, and Adam N. Sanborn. Testing the efficiency of Markov chain Monte Carlo with people using facial affect categories. *Cognitive Science*, 36(1):150–162, 2012.

[68] Pamela McCorduck and Cli Cfe. *Machines who think: A personal inquiry into the history and prospects of artificial intelligence*. CRC Press, 2004.

[69] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4): 115–133, 1943.

[70] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. In *The journal of chemical physics*, pages 1087–1092, 1953.

[71] Agnieszka Mikołajczyk and Michał Grochowski. Data augmentation for improving deep learning in image classification problem. In *2018 international interdisciplinary PhD workshop (IIPhDW)*, pages 117–122. IEEE, 2018.

[72] George A Miller. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[73] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1765–1773, 2017.

[74] Preetum Nakkiran. A Discussion of 'Adversarial Examples Are Not Bugs, They Are Features': Adversarial Examples are Just Bugs, Too. *Distill*, 2019.

[75] Radford M. Neal. Bayesian Training of Backpropagation Networks by the Hybrid Monte Carlo Method. Technical report, University of Toronto, 1992.

[76] A Emin Orhan and Brenden M Lake. Improving the robustness of ImageNet classifiers using elements of human visual cognition. *arXiv preprint arXiv:1906.08416*, 2019.

[77] Karl Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[78] Rayomond Perrault, Yoav Shoham, Erik Brynjolfsson, Jack Clark, John Etchemendy, Barbara Grosz, Terah Lyons, James Manyika, Suarabh Mishra, and Juan Carlos Niebles. Artificial Intelligence Index Report 2019. Technical report, AI Index Steering Committee, Human-Centered AI Institute, Stanford University, 2019.

[79] Omid Poursaeed, Isay Katsman, Bicheng Gao, and Serge Belongie. Generative Adversarial Perturbations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4422–4431, 2018.

[80] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[81] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[82] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[83] Evgenia Rusak, Lukas Schott, Roland Zimmermann, Julian Bitterwolf, Oliver Bringmann, Matthias Bethge, and Wieland Brendel. Increasing the robustness of DNNs against image corruptions by playing the Game of Noise. *arXiv preprint arXiv:2001.06057*, 2020.

[84] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3): 211–252, 2015.

[85] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *arXiv preprint arXiv:1703.03864*, 2017.

[86] Adam N Sanborn and Thomas L Griffiths. Markov Chain Monte Carlo with People. In *Advances in neural information processing systems*, pages 1265–1272, 2008.

[87] L Schott, J Rauber, M Bethge, and W Brendel. Towards the First Adversarially Robust Neural Network Model on MNIST. In *International Conference on Learning Representations*, 2019.

[88] Roger N Shepard. Stimulus and response generalization: A stochastic model relating generalization to distance in psychological space. *Psychometrika*, 22 (4):325–345, 1957.

[89] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian

Bolton, and others. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[90] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations*, 2015.

[91] Fabian H Sinz, Xaq Pitkow, Jacob Reimer, Matthias Bethge, and Andreas S Tolias. Engineering a less artificial intelligence. *Neuron*, 103(6):967–979, 2019.

[92] Chawin Sitawarin and David Wagner. Defending Against Adversarial Examples with K-Nearest Neighbor. *arXiv preprint arXiv:1906.09525*, 2019.

[93] Garrett Swan, Brad Wyble, and Hui Chen. Working memory representations persist in the face of unexpected task alterations. *Attention, Perception, & Psychophysics*, 79(5):1408–1414, 2017.

[94] Christian Szegedy, Joan Bruna, Dumitru Erhan, and Ian Goodfellow. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

[95] Eric Wallace. A Discussion of 'Adversarial Examples Are Not Bugs, They Are Features': Learning from Incorrectly Labeled Data. *Distill*, 2019.

[96] Norbert Wiener. *Time series*. MIT press, 1949.

[97] Geoffrey F Woodman and Edward K Vogel. Selective storage and maintenance of an object's features in visual working memory. *Psychonomic bulletin & review*, 15(1):223–229, 2008.

[98] Wayne Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Mike Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. Achieving human parity in conversational speech recognition. *arXiv preprint arXiv:1610.05256*, 2016.

[99] Dong Yin, Raphael Gontijo Lopes, Jonathon Shlens, Ekin D Cubuk, and Justin Gilmer. A fourier perspective on model robustness in computer vision. *arXiv preprint arXiv:1906.08988*, 2019.

[100] Richard Zhang. Making Convolutional Networks Shift-Invariant Again. In *International Conference on Machine Learning*, pages 7324–7334, 2019.

[101] Yi-Tong Zhou and Rama Chellappa. Computation of optical flow using a neural network. In *IEEE International Conference on Neural Networks*, volume 1998, pages 71–78, 1988.

[102] Zhenglong Zhou and Chaz Firestone. Humans can decipher adversarial images. *Nature communications*, 10(1):1334, 2019.

[103] Roland S. Zimmermann. Comment on "ADV-BNN: Improved adversarial defense through robust Bayesian neural network". *arXiv preprint arXiv:1907.00895*, 2019.

Acknowledgements

First of all, I want to thank Dr. Wieland Brendel for initiating the ideas the thesis is build upon and his support and guidance. Furthermore, I would like to thank Prof. Matthias Bethge for offering me the possibility to join his research group for writing my Master's thesis. In this context, I would like to thank all members of the research group for their support, their feedback and the fruitful discussions we had. Especially I would like to thank Judy Borowski and Robert Geirhos for their patience and guidance. Furthermore, I would like to thank Prof. Florentin Wörgötter for his role as the second referee for this thesis. Finally, I would like to thank Felix Zimmermann, Judith Schepers, Judy Borowski, Laurenz Hemmen and Max Burg for proofreading this thesis.

Erklärung nach §18(8) der Prüfungsordnung für den Bachelor-Studiengang Physik und den Master-Studiengang Physik an der Universität Göttingen:

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe.

Darüberhinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, im Rahmen einer nichtbestandenen Prüfung an dieser oder einer anderen Hochschule eingereicht wurde.

Göttingen, den 5. Februar 2020

(Roland Simon Zimmermann)