A Robust Optimization Approach against Adversarial Attacks on Medical Images

Oliver Lundqvist

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 09.01.2023

Supervisor

Prof. Fabricio Oliveira

Advisor

Prof. Fabricio Oliveira

Aalto University
School of Science

Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Oliver Lundqvist

Title A Robust Optimization Approach against Adversarial Attacks on Medical Images

Degree programme Mathematics and Operations Research

Major Systems and Operations Research **Code of major** SCI3055

Supervisor Prof. Fabricio Oliveira

Advisor Prof. Fabricio Oliveira

Date 09.01.2023 **Number of pages** 59+4 **Language** English

Abstract

Adversarial attacks have been identified as a vulnerability in the medical artificial intelligence field. According to studies, adversarial attacks may be used to conduct fraud within the medical field for obtaining financial benefit. In order to provide neural networks robustness against adversarial attacks one needs to solve a min-max or saddle point problem, where one tries to minimize the loss with respect to the weights, while simultaneously maximizing the loss with respect to a perturbation on each input sample. This problem is in general unsolvable in closed form, and even numerically.

Current practical methods for providing robustness to neural networks against adversarial attacks have been mostly based on computationally expensive adversarial training, where the idea is to train a neural network on a dataset of adversarial examples. Methods for specifically tailored architectures, which guarantee certain level of robustness, have been proposed, but these methods are not practical and generalizable for all architectures. A recently developed method called the Approximate Robust Upper Bound (aRUB) utilizes methods from robust optimization in an effort to transfer the saddle point problem via its robust counterpart. The aRUB provides an approximate upper bound to the original saddle point problem resulting in a tractable minimization problem.

In this work, robustness of aRUB is tested in practice against two different adversarial attacks, the Fast Gradient Sign Method (FGSM) and the Projected Gradient Descent (PGD), on three different medical image datasets. A common pre-trained ResNet-18 architecture is used as the underlying model. Also, the adversarial training method is tested for benchmarking the performance of the aRUB. Both aRUB and adversarial training is viewed through the lens of robust optimization. The aRUB shows competitive results compared to adversarial training.

Keywords robust optimization, adversarial training, adversarial attacks, robustness, medical artificial intelligence, neural networks

Aalto University
School of Science

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi
Diplomityön tiivistelmä

Tekijä Oliver Lundqvist	
Työn nimi Robustin optimointimenetelmän hyödyntäminen vihamielisten esimerkkihyökkäysten puolustamisessa lääketieteellisessä kuvissa	
Koulutusohjelma Matematiikka ja operaatiotutkimus	
Pääaine Operaatiotutkimus	**Pääaineen koodi** SCI3055
Työn valvoja Prof. Fabricio Oliveira	
Työn ohjaaja Prof. Fabricio Oliveira	
Päivämäärä 09.01.2023	**Sivumäärä** 59+4 — **Kieli** Englanti

Tiivistelmä

Lääketieteessä tekoälyä hyödyntävien menetelmien on havaittu olevan haavoittuvaisia vihamielisille hyökkäyksille (eng. *adversarial attacks*) ja tutkimusten mukaan vihamielisiä hyökkäyksiä voidaan käyttää petokseen. Robustoidakseen ja puolustaakseen neuroverkkoja vihamielisiltä esimerkkihyökkäyksiltä on ratkottava min-max-tehtävä tai satulapistetehtävä, jossa minimoidaan tappiofunktiota mallin parametrien suhteen, samalla maksimoiden tappiofunktiota häiriön suhteen, joka on lisätty opetusesimerkkeihin. Tehtävällä ei ole yleisesti ratkaisua suljetussa muodossa ja tehtävä ei yleisesti ratkea numeerisesti.

Tämänhetkiset käytännölliset menetelmät neuroverkon robustisoimiseen ovat pääsääntöisesti olleet neuroverkon opettaminen datalla, jotka ovat vihamielisistä hyökkäyksistä tehtyjä esimerkkejä (eng. *adversarial training*). Menetelmä on laskennallisesti raskas. Lisäksi on menetelmiä, jotka takaavat robustisuuden, mutta ne ovat neuroverkkoarkkitehtuurikohtaisia. Uusi kehitetty menetelmä approksimoitu suurin yläarvo (eng. *Approximate Robust Upper Bound*, lyhennettynä aRUB) hyödyntää robustin optimoinnin periaatteita muuntamalla alkuperäisen satulapistetehtävän robustiksi vastineeksi (eng. *robust counterpart*). Uusi aRUB menetelmä approksimoi alkuperäistä tehtävää ylärajalla, jolloin alkuperäinen tehtävä yksinkertaistuu minimointitehtäväksi, joka on ratkottavissa.

Tässä työssä arvioidaan aRUB menetelmän käytännöllistä toimivuutta kahdelle eri hyökkäykselle, nopealle gradienttimenetelmälle (eng. *Fast Gradient Sign Method*, lyhennettynä FGSM) ja projisoidulle gradienttimentelmälle (eng. *Projected Gradient Method*, lyhennettynä PGD), kolmella eri lääketieteellisellä kuva-aineistolla. Neuroverkkomallina käytetään yleistä ResNet-18 neuroverkkoarkkitehtuuria. Mallia opetetaan myös vihamielisillä esimerkeillä aRUB menetelmän vertailun vuoksi. Molempia menetelmiä tarkastellaan robustin optimoinnin näkökulmasta. Uusi aRUB menetelmä osoittautuu kilpailukykyiseksi verrattaessa opettamista vihamielisillä esimerkeillä.

Avainsanat Robusti optimointi, esimerkkihyökkäys, esimerkkiopetus, robusti, lääketieteellinen tekoäly, neuroverkko

Aalto University
School of Science

Aalto-universitetet, PB 11000, 00076 AALTO
www.aalto.fi
Sammandrag av diplomarbetet

Författare Oliver Lundqvist

Titel Användning av robust optimering i försvar mot fientliga attacker för medicinska bilder

Utbildningsprogram Matematik och operationsforskning

Huvudämne Operationsforskning **Huvudämnets kod** SCI3055

Övervakare Prof. Fabricio Oliveira

Handledare Prof. Fabricio Oliveira

Datum 09.01.2023 **Sidantal** 59+4 **Språk** Engelska

Sammandrag

Fientliga attacker (eng. *adversarial attacks*) har identifierats som ett potentiellt hot mot neuronnät i medicinska applikationer och enligt forskning kan fientliga attacker användas för att ingå bedrägeri. För att skydda neuronnät mot fientliga attacker måste man lösa ett min-max-problem eller ett sadelpunktsproblem genom att minimiera förlustfunktionen med avseende på modellparametrarna och samtidigt maximera förlustfunktionen med avseende på störningen i dataexemplen. Detta problem kan inte lösas i sluten form eller numeriskt.

De nuvarande praktiska metoderna för att robustifiera neuronnät har varit för det mesta fientlig exempelträning (eng. *adversarial training*) där idén är att träna neuronnätet med fientliga exempel. Fientlig exempelträning är beräkningsmässigt kostsamt. Andra metoder har utvecklats som garanterar en viss robusthet för specifika neuronnäts arkitekturer men metoderna är inte praktiska eller generaliserar inte för alla arkitekturer. En nyligen utvecklad metod som heter approximativ robust övre gräns (eng. *Approximate Robust Upper Bound*, förkortas aRUB), utnyttjar metoder från robust optimering för att omvandla det originala sadelpunktsproblemet till en robust motpart (eng. *robust counterpart*). Den approximativa robusta övre gränsen omvandlar det originala sadelpunktsproblemet till ett lösbart minimeringsproblem.

I detta forskningsarbete undersöks om aRUB fungerar i praktiken mot två olika attacker, nämligen snabba gradientmetoden (eng. *Fast Gradient Sign Method*, förkortas FGSM) och projicerade gradientmetoden (eng. *Projected Gradient Descent*, förkortas PGD), på tre olika medicinska dataset. Som neuronnätsarkitektur används den vanliga ResNet-18 arkitekturen. Fientlig exempelträning testas också mot fientliga attacker för att jämföra med aRUBs prestanda. Båda metoderna presenteras och undersöks från en robust optimeringssynvinkel. Den nya aRUB metoden visar lovande konkurrenskraftiga resultat jämfört med fientlig exempelträning.

Nyckelord Robust optimering, fientlig träning, fientliga attacker, robustisität, robusthet, medicinsk AI, neuronnät

Preface

First, I would like to thank my professor Fabricio Oliveira for providing the idea for the thesis as well as for the guidance and support throughout the process. It has been a hectic year for me with full-time work, writing the thesis and most importantly, becoming a father to my first son Benjamin. I highly appreciate your understanding and flexibility towards me and my current life situation. Also, I'm grateful towards my employer Sweco for providing the flexibility and possibility to integrate full-time work with my studies.

Second, I would like to thank my friends and my family, especially my parents as well as my beloved Saara, for support and encouragement throughout my studies. Without your support and commitment, this would not have been possible. I'm also grateful to my deceased grandparents who always supported and encouraged me, especially in my educational goals.

Last, but not least, I would like to thank my little baby boy Benjamin just for his presence. I'm so grateful.

Espoo, 9.1.2023 Oliver G. S. Lundqvist

Contents

Symbols and abbreviations

Symbols

b	bias term
B	a batch of input samples
C	classifier
d	a direction vector
f	arbitrary function
h	neural network model
K	number of classes
l	current layer
L	number of layers
m	input dimension
N	number of data points
p	number for norm space
q	number for norm space
r	dimension parameter
x	input vector of neural network
$\tilde{x}$	adversarial example
y	true class label of input x
z	output of a neural network layer
W	weight matrix of a neural network
β	batch normalization parameter
γ	batch normalization parameter
δ	perturbation of input
ϵ	batch normalization hyperparameter
ε	perturbation size and the constraining limit for the uncertainty set $\mathcal{U}_p$
η	learning rate
θ	model parameters
λ	small real value
σ	activation function or softmax function
ϕ	arbitrary function
$\mathcal{B}$	block inside ResNet
$\mathcal{K}$	kernel of a convolutional layer
$\mathcal{L}$	loss function
$\mathbb{R}$	the set of all real numbers
$\mathcal{S}$	nonempty compact topological space
$\mathcal{U}_p$	uncertainty set bounded by the p-norm
$\mathcal{X}$	input space
$\mathcal{Y}$	output space

Operators

∇_x	gradient with respect to x
∇_θ	gradient with respect to θ
$\sum_i$	sum over index i
$\mathbb{E}[x]$	expected value of x
$\mathrm{Var}[x]$	variance of x
$\|\|x\|\|_p$	p-norm of x
$\lim_{x\to 0} f(x)$	limit of $f(x)$ as x approaches 0
$\to$	mapping of one space to another
max	maximum of set or function
min	minimum of set or function
sup	supremum of set or function
inf	infimum of set or function
$\mathcal{P}(x)$	projection operator of x to domain boundary if outside domain
arg max	argument that maximizes the function or set
arg min	argument that minimizes the function or set
$\mathrm{sign}(x)$	sign of x
$f'(x; d)$	directional derivative of f with respect to x in direction d

Abbreviations

AI	artificial intelligence
BIM	Basic Iterative Method
CW	Carlini and Wagner attack
FDA	Food and Drug Administration
FGSM	Fast Gradient Sign Method
FN	false negatives
FP	false positives
JSMA	Jacobian-based Saliency Map Attack
ML	machine learning
LP	linear program
L-BFGS	Limited Memory Broyden–Fletcher–Goldfarb–Shanno
PGD	Projected Gradient Descent
ReLU	Rectified Linear Unit
TN	true negatives
TP	true positives
U.S.	United States
UAP	universal adversarial perturbation
WHO	World Health Organization

1 Introduction

During recent years, there has been a keen interest in implementing artificial intelligence (AI) and machine learning (ML) algorithms and systems into the medical field to perform different tasks, such as work automatization, medical image segmentation, decision support and for diagnosing illnesses (Davenport and Kalakota, 2019). A milestone of AI development in the medical field was passed in 2018 when the first computer vision algorithm for diagnosis for clinical use without a clinician's input was approved by the U.S. Food and Drug Administration (FDA) (U.S. Food and Drug Administration, 2018a). Also, the use of AI-based medical devices has increased significantly in the past years and the trend is expected only to be increasing (Muehlematter et al., 2021).

Even though AI provides help and assistance in medical treatment within healthcare, AI, especially deep neural networks, is prone to security threats. One of these threats is called adversarial attacks, where the input of an AI algorithm is altered with small changes in order to fool the AI algorithm or the deep neural network. These changes could, for example, be slightly modifying pixel values of medical image data in order to engineer a desired outcome or mislead the AI algorithm to give false predictions. A key characteristic in adversarial attacks is that the difference between the original image and the attack, i.e., contaminated image with pixel changes, is not typically visible and distinguishable to humans. This kind of vulnerability allows the possibility to conduct fraud in systems where AI is a key component, such as AI-powered medical devices. This is especially problematic in the healthcare sector of the U.S. where health care is not generally public and is built on private healthcare and private health insurance policies, which are for-profit businesses and organizations. Thus, organizations, businesses and even private persons are susceptible to fraud in form of financial benefit. That is why health insurance companies have invested millions of dollars in personnel and IT to detect fraud (Wynia et al., 2000; Kesselheim and Brennan, 2005). For example, it is estimated that in 2011 the amount of money insurance companies used in detecting fraudulent cases within the healthcare is around \$82 to \$282 billion in the U.S. alone (Jain et al., 2014). Similarly, the World Health Organization (WHO) estimated that the fraudulent monetary gain for the year 2013 reached up to \$455 billion of the total \$7.35 trillion healthcare market (Gee and Button, 2015). As AI-powered medical devices are more widespread it can only be assumed that malicious attacks, such as adversarial attacks, with fraudulent intentions will also increase in the future for the AI-powered healthcare industry. Preparing and making AI and ML algorithms more robust and invulnerable to adversarial attacks is still an unsolved problem of today.

Since the discovery of adversarial attacks by Szegedy et al. (2013), adversarial attacks have gained a lot of attention within the AI research field and community. Adversarial attacks have been studied to understand why they exist but also how to prevent and defend against adversarial attacks. Some methods for defending against adversarial attacks have been proposed, for example, adversarial training (Goodfellow et al., 2015), adversarial example detection (Gong et al., 2017) and model distillation (Papernot et al., 2016a). Also, different types of attacks have been studied by Madry et al. (2018), Carlini and Wagner (2017) and Kurakin et al. (2017) in order to detect vulnerabilities in deep neural networks. Different types of attacks have been used to construct more resilient AI systems by using these attacks to train AI systems and deep neural networks.

Adversarial attacks on medical image data have been studied by Finlayson et al. (2019), Ma et al. (2021), Bortsova et al. (2021) and Hirano et al. (2021) to identify the vulnerability in the medical field. The general conclusion from the research is that deep neural networks trained on medical data are highly vulnerable against adversarial attacks and will give false predictions even with very small manipulation of the input data. More worryingly, it seems that a minimal amount of research has been conducted in implementing methods for counteracting or defending against adversarial attacks for AI within the medical field. This lack of research and testing of defense methods against adversarial attacks for deep neural networks within the medical field serves as the practical motivation for this thesis.

The main goal of this research is to test modern defense methods against adversarial attacks for the practical case of medical data, i.e., robustifying the neural networks for adversarial attacks. The research tries to answer the question if modern methods from robust optimization can provide robustness against adversarial attacks within the medical field of AI and whether it works in practice. Adversarial training and a robust upper bound are compared as strategies for defending against adversarial attacks on three different medical image datasets: the diabetic retinopathy dataset (Kaggle, 2015), the nevus image dataset (ISIC, 2019), and the chest X-ray dataset (Irvin et al., 2019). The robust upper bound and adversarial training are viewed as an optimization problem through a robust optimization lens, i.e., the thesis focuses on strategies for solving a saddle point problem (min-max problem) that arises from the robust learning problem. First, the common adversarial training problem is viewed as a lower bound solution to the saddle point problem and clear theoretical justification for the case is derived. Second, the robust upper bound is viewed as an approximate upper bound to the original saddle point problem. The second method is a recently developed Approximate Robust Upper Bound by Bertsimas et al. (2021a). The derivation and use of the approximate robust upper bound is presented and discussed.

In total five models are tested, namely one nominal model, two approximate robust upper bound models with different norms and two models trained with adversarial samples. The models are tested for robustness against adversarial attacks. A common convolutional ResNet-18 (He et al., 2016) neural network architecture with a fully connected four-layer neural network for the classification task is used for all models. The two attacks are the Fast Gradient Sign Method and the Projected Gradient Descent bounded in the L_∞-norm. The neural network is also limited to a binary classification task.

The structure of the thesis is divided into six main chapters. Chapter 2 presents the background of adversarial attacks, training and the robustification of neural networks. In chapter 3, the used and implemented methods are presented, derived and discussed through a robust optimization lens. After this, in chapter 4, the experimental setup is presented, whereafter in chapter 5, the results are presented and discussed. Finally, in chapter 6, the research is summarized and concluded.

2 Background

This chapter provides background to adversarial attacks and training robust neural networks as well as discusses related work within the field. First, in section 2.1, the neural networks are mathematically defined in order to thereafter properly present previous research of adversarial attacks and adversarial training in section 2.2. Thereafter, in section 2.3, background to robustness of neural networks is discussed. Also, the research and related work on adversarial attacks on medical image data is discussed in section 2.4.

2.1 Neural Networks

A neural network classifier C is a mapping from an input space $\mathcal{X}$ to an output space $\mathcal{Y}$, i.e., $C : \mathcal{X} \rightarrow \mathcal{Y}$. Typically for a classification task, the input space $\mathcal{X}$ is the m-dimensional Euclidian space $\mathbb{R}^m$ and the output space is the classification set $\mathcal{Y} = \{1, ..., K\}$ where K is the number of classes and each number in set $\mathcal{Y}$ corresponds to a specific numbered class label. The numbering of class labels can also start from zero as zero is typically the first index for arrays in different programming languages making it more practical. For each input vector x there is a corresponding true class label $y \in \mathcal{Y}$. The target of a neural network classifier is to learn the classifier $C(x)$ of a scoring function $h(x, \theta)$ such that

$$y = C(x) = \arg\max_{i \in \mathcal{Y}} \{h_i(x, \theta)\}, \tag{1}$$

where θ refers to the parameters of the scoring function and $\{h_i(x, \theta)\}$ to the set of vector components for the output vector of the scoring function. The actual neural network model can be considered to be the scoring function $h(x, \theta)$ with parameters θ. The neural network is then a mapping of $h : \mathcal{X} \rightarrow \mathbb{R}^K$. The neural network is constructed of layers l of the form

$$z_l = \sigma(W_l z_{l-1} + b_l), \tag{2}$$

where $W_l \in \mathbb{R}^{r_l \times r_{l-1}}$ is the weight matrix, b_l is the bias vector and σ is a nonlinear activation function. The dimension parameter r_l refers to the size of corresponding layers l output dimension size. A variety of activation functions exists and they are chosen depending on the problem and model architecture. A common activation function is the *Rectified Linear Unit* (ReLU) defined as

$$\mathrm{ReLU}(x) = \max(0, x). \tag{3}$$

Another common activation function is the *softmax* function, which gives a probabilistic output within the range $[0, 1]$ for each vector component and where the vector components sum up to one. Softmax is typically used in the last layer for providing the output as a probabilistic interpretation. The softmax is defined as

$$\sigma_i(x) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}. \tag{4}$$

A fully connected neural network model with L number of layers is thus simply described as

$$
\begin{aligned}
z_1 &= \sigma(W_1 x + b_1), \\
z_l &= \sigma(W_l z_{l-1} + b_l) \quad 2 \leq l \leq L,
\end{aligned}
\tag{5}
$$

where, sometimes, the activation of the last layer is omitted. The parameters of the model is the set $\theta = \{W_l,\ b_l\ :\ l = 1, ..., L\}$.

A *convolutional neural network* is a special case of a fully connected neural network, where a convolution operation is applied. In neural networks, the convolution operation is typically replaced with the cross-correlation and thus a two-dimensional convolutional layer is defined as

$$
z_l(i,j) = \sum_h^H \sum_w^W z_{l-1}(i+h, j+w)\mathcal{K}_l(h,w), \tag{6}
$$

where i and j are the input locations or indexes of the two-dimensional input tensor and $\mathcal{K}$ is the kernel with width W and height H corresponding to the weight matrix for the convolutional layer. A convolutional layer is typically followed by pooling operations. Pooling operations are functions that downsample the input vector. For example, the *max-pooling* with height H and width W, is defined as

$$
z_l(i,j) = \max_{h=0..H, w=0..W} z_{l-1}(i+h, j+w). \tag{7}
$$

Different normalization operations are typically used in-between convolutional layers. One common normalization function is the *batch normalization* operation and it is defined as

$$
z = \frac{x - \mathbb{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}}\gamma + \beta, \tag{8}
$$

where γ and β are trainable parameters, ϵ is a hyperparameter (not trainable), $\mathbb{E}[x]$ is the expected value of x and $\mathrm{Var}[x]$ is the variance of x.

Formulation (5) of a neural network is general and works both for fully connected networks as well as convolutional neural networks, as convolutional neural networks are just special cases of matrix-vector multiplications and pooling, as well as batch normalization, a special kind of activation functions (Bertsimas et al., 2021a). Without loss of generality, the aforementioned general formulation of (5) can be used to describe a neural network.

Given a dataset $X_N = \{x_n, y_n\}_{n=1}^N$ with N input vectors, training of the neural network breaks down to solving the optimization problem with respect to the parameters θ and can be stated as

$$
\min_\theta \ \frac{1}{N} \sum_{n=1}^N \mathcal{L}(h(x_n, \theta), y_n), \tag{9}
$$

where $\mathcal{L}$ is a loss function. Various loss functions exist, but in this work, the *cross entropy* loss is used. The equation for cross entropy is given as

$$\mathcal{L}(h(x,\theta),y) = \log(\sum_{j=1}^{K} e^{(e_k-e_y)^T h(x,\theta)}), \tag{10}$$

where e_k and e_y are vectors of K zeros, except for the location indicated with the subindex where the vector takes a value of one. The minimization problem (9) can be solved by gradient descent or other gradient-based optimization methods. Applying gradient descent to problem (9) yields

$$\theta_{i+1} = \theta_i - \eta \frac{1}{N} \sum_{n=1}^{N} \nabla_\theta \mathcal{L}(h(x_n,\theta),y_n), \tag{11}$$

where η is a step size parameter also known as the learning rate. One variant of the gradient descent is the *stochastic gradient descent* where the gradient is evaluated for a minibatch B consisting of $|B|$ number of input samples, i.e.,

$$\theta_{i+1} = \theta_i - \eta \frac{1}{|B|} \sum_{n=1}^{|B|} \nabla_\theta \mathcal{L}(h(x_n,\theta),y_n). \tag{12}$$

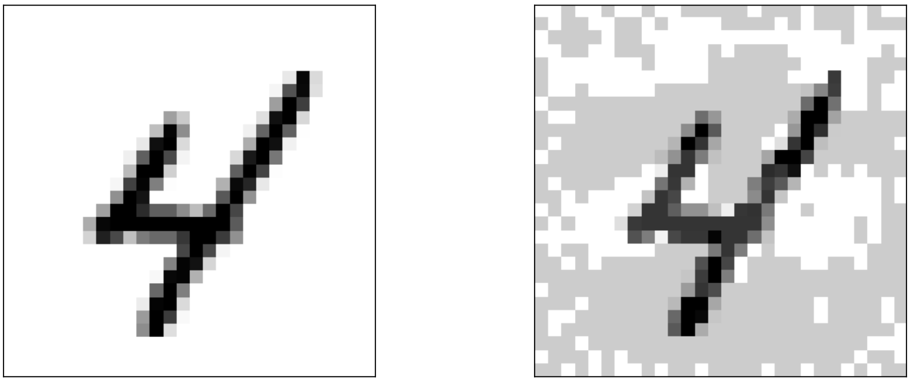
The number of samples in a minibatch is less than the number of samples in the dataset, i.e., $|B| < N$. The method (12) is then looped over the mini-batches so that the full dataset is used in the training. One iteration round over the whole dataset is called an *epoch* and typically multiple epochs are used when training neural networks. Other versions of the stochastic gradient descent also exist, but the method (12) is typically used to train neural networks.

2.2 Adversarial Examples, Attacks and Training

Szegedy et al. (2013) stated that neural networks could be fooled to misclassify and produce false predictions by just slightly manipulating the input. This manipulation could be, for example, changing pixel values of image data. Typically, these manipulations are not perceivable to humans. Data that has been manipulated or altered with a perturbation in this way is called an *adversarial example* or an *adversarial sample* and a systematic approach to fool a neural network or AI system is called an *adversarial attack*. The perturbation is here noted as δ and thus the adversarial example $\tilde{x}$ is constructed by simply summing the perturbation to the original vector x, hence

$$\tilde{x} = x + \delta. \tag{13}$$

An example of this kind of adversarial sample is given in Figure 1, where a clean image x and an adversarial example made from the same image by summing the perturbation on the clean image, i.e., $x + \delta$, is shown. The image represents a handwritten digit of the number four. The perturbation of the original image is shown as grey areas. Typically the perturbation is not visible to the eye, but in



(a) Clean original image (b) Adversarial example

Figure 1: Example of adversarial attacks from example images from the MNIST dataset (LeCun, 1998) of handwritten images. (a) Original clean image and (b) the same image manipulated with an FGSM attack with a large perturbation.

Figure 1 the perturbation has been scaled for illustrative purposes.

Various researchers have studied adversarial examples and attacks in order to understand the cause and effects of the vulnerability. Despite extensive studies, a consensus on the root cause has not been determined. Goodfellow et al. (2015) propose that the vulnerability is the cause of the neural networks being too linear as linear behavior in high dimensions is already sufficient to produce vulnerability against adversarial examples, while Ilyas et al. (2019) state that the root cause of the vulnerability is that neural networks are sensitive to data features that are not robust. In other words, neural networks tend to learn features and structures within the data that are not perceivable to humans. This is probably also one of the reasons why neural networks seem to outperform humans in some tasks.

Adversarial attacks are generally categorized into different kinds of attacks as to how they are constructed. The attacks are usually classified into the following categories:

- **Black-Box Attack**. A black-box attack is an attack where the attacker has incomplete information about the model. The attacker cannot access the model weights but may have access to the output of the model as well as the general architecture layout of the model.

- **White-Box Attack**. A white-box attack is an attack that is performed with full or complete information about the model being attacked, i.e., the attacker has full access to the model, its weights and architecture layout. A white-box attack is thus considered the strongest form of attack.

- **Targeted Attack**. A targeted attack is an attack with a pre-specified desired target, i.e., the attacker aims to obtain a specific output of the model.

- **Untargeted Attack**. An untargeted attack is an attack where the output of the model is not of importance, except that the output of the model must be incorrect.

The discovery of adversarial attacks has started a search for different methods to provide robustness to neural networks and how to train a robust classifier, i.e., a classifier that performs well on input data consisting of adversarial samples. One of these methods is the *adversarial training*, where the neural network is trained on adversarial examples together with regular samples or solely on adversarial examples. This corresponds to modifying equation (9) to the form

$$\min_{\theta} \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(h(x_n + \delta_n, \theta), y_n). \tag{14}$$

Usually the perturbation δ is given an uncertainty domain or set

$$\mathcal{U}_p = \{ \, \delta : ||\delta||_p \leq \varepsilon \, \}, \tag{15}$$

where p defines any norm of choice and ε is the constraining limit for the uncertainty domain. Broadly speaking, ε is considered as the size of the perturbation and adversarial attack as it determines the maximum value of the norm for the perturbation. In order to find good candidates of δ, one usually maximizes the loss $\mathcal{L}$ with respect to the perturbation, which leads to the robust training problem

$$\min_{\theta} \frac{1}{N} \sum_{n=1}^{N} \max_{\delta_n \in \mathcal{U}_p} \mathcal{L}(h(x_n + \delta_n, \theta), y_n). \tag{16}$$

Adversarial training was first proposed by Goodfellow et al. (2015), where they used the *Fast Gradient Sign Method* (FGSM) to train neural networks on the MNIST dataset (LeCun, 1998). For their tests Goodfellow et al. (2015) obtained an error rate of 17.9% for the FGSM-trained model, while for the nominally trained (without adversarial examples) they obtained an error rate of 89.4%. FGSM is argued to be a fast one-step method for constructing adversarial examples when the perturbation is bounded by the infinity norm, i.e., $||\delta||_\infty \leq \varepsilon$. The adversarial example with FGSM is calculated as

$$\widetilde{x} = x + \varepsilon \, \mathrm{sign}(\nabla_x \mathcal{L}(h(x, \theta), y)). \tag{17}$$

Similarly, Kurakin et al. (2017) presented an improved version of the FGSM which they called *Basic Iterative Method* (BIM). The idea of BIM is to perform FGSM iteratively with a smaller step size $\alpha < \varepsilon$ and for a fixed number of steps I, i.e.,

$$x_{i+1} = x_i + \alpha \nabla_x \mathcal{L}(h(x_i, \theta), y) \quad \text{for} \quad i = 1, ..., I. \tag{18}$$

Kurakin et al. (2017) showed that adversarial attacks made by BIM showed larger drops of accuracy than attacks made by FGSM on the ImageNet dataset (Deng et al., 2009). The error rate for their test model with a BIM attack was about 70%, while for an attack with FGSM the error rate was 60%, indicating that BIM is a stronger attack than FGSM.

The BIM attack was further improved by Madry et al. (2018) by introducing the *Projected Gradient Descent* (PGD), which they argue to be the strongest first-order attack. First order in this context means attacks based on only first-order derivatives, i.e., the gradients of the loss function. The PGD is of the form

$$x_{i+1} = \mathcal{P}(x_i + \alpha \nabla_x \mathcal{L}(h(x_i, \theta), y)) \quad \text{for} \quad i = 1, ..., I, \tag{19}$$

where $\mathcal{P}$ is the projection operator that maps the point back to the feasible domain's boundary $\partial \mathcal{U}_p$, if it is outside $\mathcal{U}_p$, i.e, $\mathcal{P} : \mathcal{X} \setminus \mathcal{U}_p \to \partial \mathcal{U}_p$. Madry et al. (2018) showed that a model trained only on FGSM adversarial samples obtained an error rate of 100% on the CIFAR10 dataset (Krizhevsky, 2009) with a PGD attack, while a PGD-trained model had only an error rate of 54.2%. While Madry et al. (2018) did not provide any theoretical proof of their claim of the strongest first-order attack, they provided empirical justifications by evaluating a PGD-trained model against other types of attacks.

Other adversarial attacks and adversarial example-generating methods have also been developed and studied. For example, Papernot et al. (2016b) developed an attack based on minimizing the difference between the original image and the adversarial example by some distance metric, for example, the L_2-norm, and by providing the misclassification requirement as a constraint. This kind of formulation can be viewed as an optimization problem of the form

$$\begin{aligned} &\min ||\delta||_p \\ &\text{s.t} \quad C(x) \neq y. \end{aligned} \tag{20}$$

To solve this kind of minimization problem, they provide an iterative algorithm that computes the Jacobian of the neural network with respect to the input and then utilize this Jacobian to create a saliency map to indicate which features in the input to perturb, in order to affect the output most efficiently. The input is then perturbed with the given saliency mapping and repeated until desired convergence. This *Jacobian-based Saliency Map Attack* (JSMA) proved to only need a 4.02% change of the original input to fool a neural network with a 97% error rate on the MNIST dataset and their test neural network. Similarly, Carlini and Wagner (2017) based also their attack on minimizing the needed perturbation of the original input with respect to a norm. They proposed a box-domain constraint for image data and use the Limited Memory Broyden–Fletcher–Goldfarb–Shanno algorithm (L-BFGS) to solve their optimization problem. This allows the changes to be minimal and thus hard to detect and identify by different defenses. This form of an attack is called a *Carlini and Wagner*(CW) attack.

To defend against adversarial attacks the most common and widely researched method is adversarial training. However, other methods and strategies exist. For example, Papernot et al. (2016a) presented a method called *model distillation* for increasing the robustness of neural networks against adversarial attacks. The motivation behind model distillation is to transfer knowledge from larger neural networks to smaller neural networks, thereby reducing the complexity of the network and its architecture by utilizing the information obtained during training in the output probability vector. The idea is to have a larger neural network with a modified softmax layer as the last layer to produce a probability output vector that is then fed as input to a smaller neural network. The last layer's softmax has a regularizing parameter called temperature, in order to produce soft labels opposed to the normal hard labels, which would then classify the input. Papernot et al. (2016a) obtained an error rate of 0.45% on the MNIST dataset and a 5.11% error rate on the CIFAR10 dataset against JSMA attacks. However, this technique was quickly proven to be vulnerable to the CW attack by Carlini and Wagner (2017), where they obtained a 100% error rate against a distilled neural network for the MNIST and CIFAR10 datasets.

Another researched method to defend against adversarial attacks is adversarial sample detection. Typically, these detection methods are used as preprocessing algorithms before the actual neural network to protect the neural network against adversarial attacks by filtering out adversarial samples from the input data, when the neural network is used in production. For example, Gong et al. (2017) developed a simple binary classifier to detect adversarial samples from normal input data. Metzen et al. (2017) used a similar approach by augmenting the neural network with a subnetwork that branches off the main network at some layers and produces a probability output that describes the probability of the input being adversarial. This subnetwork showed good results for weaker attacks such as FGSM but failed for stronger attacks such as BIM. A logistic regression detector based on kernel density and bayesian uncertainty features was proposed by Feinman et al. (2017) and showed promising results in detecting adversarial samples.

2.3 Robustness Certificates and Verification

During the years of studying and researching adversarial attacks and defenses, every time a new heuristic defense is proposed, a new attack is developed to break that defense, resulting in an arms race between defenses and attacks. This never-ending arms race has led to studies of certain guarantees of robustness, called *robustness certificates* or *robustness verification*. The standard nominal verification of neural networks consists of splitting the datasets into train and test sets where the model is trained on the train set and after that, the neural network is evaluated on the test set, which is data that the neural network has not seen before. The evaluation of the neural network's success and acceptability can be on standard metrics like accuracy or even loss value. Robustness certificates and verification require the neural network to be robust against all or certain adversarial attacks within a certain closed domain. In other words, the neural network shall not change its prediction given a small perturbation of the input in a given (small) domain (Dvijotham et al., 2018; Lecuyer et al., 2019).

In order to achieve guaranteed robustness certificates and verification of neural networks, a theoretical framework needs to be established. This usually corresponds to transforming the training problem into another problem with the desired properties. For example, Wong and Kolter (2018) used a convex optimization approach with duality and constructs an optimization formulation for the verification. They construct a convex outer bound for the adversarial polytope, which is the set of all final layer activations (outputs) that are achieved by perturbing the input within the given domain. The result is a *Linear Program* (LP) that is an optimization problem consisting only of linear equalities, inequalities and objective function. Also, Katz et al. (2017) provided a method for neural networks with ReLU activation functions. The verification problem results in an optimization problem with a linear objective function and nonlinear constraints, based on the ReLU activation functions. Katz et al. (2017) extended the standard Simplex method and utilize the piece-wise linear definition of the ReLU. This method they called Reluplex.

A similar approach was also proposed by Raghunathan et al. (2018) by computing a robust upper bound of the worst case loss for neural networks with one hidden layer, based on a semidefinete relaxation. This robustness certificate results in a semidefinite program. The method achieved a test error rate of 35% on the MNIST dataset with a perturbation size of $\varepsilon = 0.1$. Dvijotham et al. (2018) formulate a verification as an optimization problem with the objective to find the maximum violation of the verification and solve the problem by Lagrangian relaxation that results in an upper bound on the worst case violation. The method by Dvijotham et al. (2018) was proven to be efficient and provided in the best case an error rate of approximately 20% with perturbations of $= 0.1$ on the MNIST dataset.

The aforementioned methods rely either on transforming the training of a neural network with robustness verification to another optimization problem that can either be solved exactly or be approximated with an upper bound. Nonetheless, these methods are either designed for specific neural network architectures or do not work well with larger models that have millions of parameters. Training and guaranteeing robustness for general models seems to have been computationally intractable until now. Bertsimas et al. (2021a) proposed two new computationally tractable methods to train robust classifiers by utilizing tools from robust optimization. Robust optimization, as developed by Ben-Tal et al. (2009), is used to solve optimization problems under uncertainty. Typically, in robust optimization, the original problem with uncertainty is transformed into a *robust counterpart* that is computationally tractable. In their classical book, Ben-Tal et al. (2009) present methods for LPs under different uncertainty sets and also for some special cases of nonlinear optimization problems under uncertainty. These classical methods from robust optimization rely on the convexity of the problem or the uncertainty sets. Since neural networks are neither convex or concave and the dimensions which they work in may be large, the classical robust optimization methods do not work theoretically and might not scale well for problems in AI and ML (Bertsimas et al., 2021a).

Bertsimas et al. (2021a) claimed that their methods can be used in a normal neural network training framework and the methods are theoretically solid as well as computationally tractable. Both methods provide an upper bound to the problem (16) such that

$$\min_{\theta} \frac{1}{N} \sum_{n=1}^{N} \max_{\delta_n \in \mathcal{U}} \mathcal{L}(h(x_n + \delta_n, \theta), y_n) \leq \min_{\theta} \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}_{UB}(h(x_n, \theta), y_n, \varepsilon), \tag{21}$$

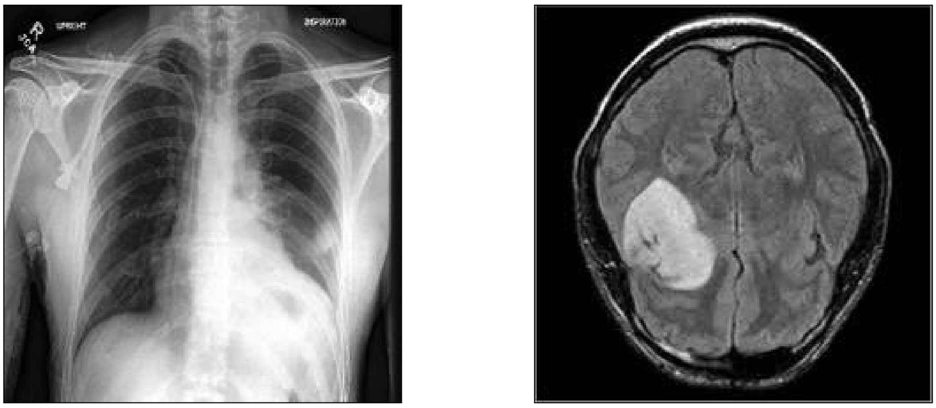
where $\mathcal{L}_{UB}$ is a modified loss function and is also a function of the perturbation size ε. Upper bounding the solution can be seen as finding a robust counterpart as it still provides a robust solution to the problem. The first method, the *Approximate Robust Upper Bound* (aRUB), utilizes a first-order approximation of the neural network h. This results in a regularizing term for each individual class in the loss function and requires an additional computation of the gradient of the neural network with respect to the input. Luckily, this can quite easily be implemented in modern deep learning frameworks. Bertsimas et al. (2021b) tested this method together with other robustification methods on a variety of datasets. The results seemed promising but are tested on a small three-layer feed-forward (fully connected) network with ReLU activation functions, which is an architecture that probably would not be used as such in a real-world application.

The second method by Bertsimas et al. (2021a), the *Robust Upper Bound* (RUB), splits each layer of the neural network as a sum of convex and concave functions and thereby obtains a robust upper bound by further exploiting dual norms. This, however only works when the defined uncertainty set $\mathcal{U}$ is the L_1 sphere, i.e, $\mathcal{U}_1 = \{\ \delta : ||\delta||_1 \leq \varepsilon\ \}$. The key idea here is to utilize the fact that the dual norm of the L_1 norm is the L_∞ norm, which corresponds computationally to finding the maximum value of each vector component, making it computationally feasible and tractable. Unfortunately, RUB is also dependent on the number of dimensions of the input and thus makes it slow to compute for practical systems, even though the authors argue that it is highly parallelizable and can be computed effectively. This might be the case if one has access to sufficient computational resources.

2.4 Adversarial Attacks on Medical Images

Medical images are images obtained in the medical treatment of patients in order to perform diagnosis or support the diagnosis decision as well as treatment decision. The images can, for example, be X-ray images from internal organs such as lungs, images from eyes as well as images from nevi. The image data is usually simple RGB (red, green and blue) or gray-scale images of different pixel sizes. From a ML and AI perspective, a RGB image is a three-dimensional tensor and a gray-scale image is a two-dimensional tensor. Figure 2 shows an example of some typical gray-scale medical image data.

There has been an increase in using ML and AI in the medical field and this trend is expected to continue (Muehlematter et al., 2021). To support the development of this technology and to provide safety criteria, standards and demands, FDA has



(a) Chest X-ray image.

(b) Brain MRI image.

Figure 2: Examples of medical data. (a) A chest X-ray image (Irvin et al., 2019) and (b) a brain MRI image (Kaggle, 2020).

published AI and ML guideline documents and regulations that apply to medical devices utilizing ML and AI (U.S. Food and Drug Administration, 2012, 2018b). Currently, there are 343 AI and ML utilizing medical devices approved and registered by the FDA in the U.S. (U.S. Food and Drug Administration, 2018b).

Adversarial attacks and their effect on artificial intelligence trained on medical data have gained some interest during the past years. Several studies have been conducted, where vulnerabilities of these deep neural networks have been identified. Finlayson et al. (2019) demonstrated that practical medical deep neural network classifiers are vulnerable against adversarial attacks. Even though robustness and safety would be preferable for medical AI systems, it seems that the driving motivation is financial from a cybersecurity perspective rather than a patient safety perspective. Finlayson et al. (2019) provided practical examples in dermatology, radiology and ophthalmology of how a fraudulent actor could use adversarial attacks to conduct insurance fraud. The same arguments and motivation to prevent financial fraud in the medical field are argued by Ma et al. (2021), Xu et al. (2020) and Bortsova et al. (2021). Nonetheless, the FDA has not recognized this threat of adversarial attacks against medical devices in their currently binding cybersecurity guideline and regulations (U.S. Food and Drug Administration, 2014) nor their newly released updated draft of the cybersecurity guideline (U.S. Food and Drug Administration, 2022), which indicates that the problem is new and unsolved in practice.

Different types of adversarial attacks on medical image data were studied by Ma et al. (2021). They used FGSM, BIM, PGD and CW attacks on fundoscopy, dermatology and chest X-ray datasets with a pre-trained ResNet-50 neural network model, where the last fully connected layers were customized to the specific classification task and the model was fine-tuned for the medical data. Ma et al. (2021) showed that even a perturbation size of $\varepsilon \approx 0.001$ in the L_∞-norm (which corresponds to a $0.3/255$ pixel value change) resulted in a complete drop of accuracy and performance. Similar results were already shown by Finlayson et al. (2019) with white-box and black-box attacks where the performance was tested against a maximal pixel perturbation of $\varepsilon = 0.02$ which corresponds to approximately $5/255$ pixels.

A more "realistic" adversarial attack on medical data was studied by Bortsova et al. (2021), where they used a black-box approach with a surrogate model to construct an attack, which was then utilized on the target model. The surrogate model's task is to replicate the target model as well as possible. In reality, the attacker might not know all the details of the architecture but may know its basic underlying architecture. Bortsova et al. (2021) showed that this kind of attacking strategy is successful for medical neural network models, especially if they are pre-trained with other publicly accessible data. Pre-training is commonly used as a training strategy for neural networks as using pre-trained and initialized model weights saves time and effort. In addition, Bortsova et al. (2021) showed that using a surrogate model creates adversarial attacks, which are similar to attacks produced by the target model and thereby further confirm the vulnerability of current medical AI systems, which are trained on public data.

Even worse, Hirano et al. (2021) and Minagi et al. (2022) showed that medical deep neural networks are vulnerable to *universal adversarial perturbations* (UAP). A UAP is a general targeted attack constructed from general and openly available data. Given a dataset, the goal of a UAP is to construct a universal perturbation that fools a neural network on as many images as possible while minimizing the perturbation according to some norm (Moosavi-Dezfooli et al., 2017). The jointly obtained results from Hirano et al. (2021) and Minagi et al. (2022) showed that an UAP-attack is capable of causing a 80% error rate on common architectures and even higher when transfer learning has been applied. Transfer learning, briefly, is transforming knowledge obtained from a problem and transferring it to another problem that has to be solved (Pan and Yang, 2009). The earlier mentioned pre-training can be considered a form of transfer learning.

Defenses against adversarial attacks on medical deep neural networks have gained considerably less attention than identifying the actual vulnerability. Xu et al. (2020) tried to address this issue by proposing two different methods used for defensive purposes. The first method, *Multi-Perturbations Adversarial Training* (MPAdvT), is a method where normal adversarial training is performed but with different perturbation sizes at different iteration stages. The second method, *Misclassification-Aware Adversarial Training* (MAAdvT), is a penalty method where misclassified adversarial samples are penalized in the loss function. Both these methods showed increased accuracy for adversarial attacks when compared to standard adversarial training but still resulted in poor absolute accuracy of approximately 30% for their test attack.

Even though multiple defense strategies against adversarial attacks have been developed, the use of these defenses in medical deep neural networks has been adversarial training, adversarial detection or other manipulations of the architecture (Apostolidis and Papakostas, 2021). Also adding to the problem with defenses is that training medical deep neural networks are difficult enough with the limited amount of data and the demand for high accuracy. No simple and efficient method for providing robustness for medical deep neural networks seems to have been implemented and tested yet. Hence, the problem of robustifying neural networks in the medical field is still an unsolved problem.

3 The Robust Optimization Approach

This chapter focus on solving the robust loss problem (16) by viewing the problem through a lens of robust optimization as a common saddle point problem. First, the robust loss problem is presented and discussed in section 3.1. Next, an approximate method of solving the inner maximization is presented practically as well as investigated theoretically in section 3.2. This approximate method results in adversarial training and is considered a lower bound solution to the saddle point problem. Thereafter, in section 3.3, a modern method proposed by Bertsimas et al. (2021a) is derived and discussed, which provides an approximate upper bound for the saddle point problem. This approximation can be considered as a robust counterpart for the original problem.

3.1 The Saddle Point Problem

Given a classification problem of K different classes with a given dataset $X_N = \{x_n, y_n\}_{n=1}^N$ of N datapoints and where $x \in \mathbb{R}^m$, m is the number of dimensions and $y \in \{1, ..., K\}$, the task of training a robust neural network h is to solve the robust loss problem or min-max problem with loss function $\mathcal{L}$ and a perturbation δ within $\mathcal{U}_p = \{\ \delta : ||\delta||_p \leq \varepsilon\ \}$, thus

$$\min_{\theta} \frac{1}{N} \sum_{n=1}^{N} \max_{\delta_n \in \mathcal{U}_p} \mathcal{L}(h(x_n + \delta_n, \theta), y_n). \tag{22}$$

Here, the parameters of interest are the neural network weights and biases summarized in the set θ. Equation (22) is known as a saddle point problem or min-max problem (hereinafter referred to only as the saddle point problem). This kind of a problem can be interpreted as minimizing the possible worst-case scenario. In general, problem (22) does not have a general solution method nor is it computationally tractable, but for some special cases, it can be solved. The tractability depends on the problem characteristics, i.e., the convexity of the objective function as well as the uncertainty domain.

For artificial intelligence, solving the saddle point problem (22) coincides with robustifying the neural networks for perturbations in the input by minimizing the loss over the worst case. As discussed in chapter 2, perturbations of the input can be utilized in adversarial training. Thus, a good feasible solution should provide desired robustness for the neural network.

Neural networks pose additional problems in solving (22), as neural networks are neither convex nor concave functions, thus, making it very difficult to solve generally or to find robust counterparts as done in robust optimization (Bertsimas et al., 2021a). In general, three solution and approximate solution strategies can be implemented for the given problem, namely:

- **Lower bound solution**. Any feasible solution to the problem (22) can be considered a lower bound solution, as there are no guarantees that the solution is the optimal or a true saddle point. Adversarial training falls into this category of lower bound solutions.

- **Exact solution**. An exact or guaranteed optimal solution for general neural networks has not been discovered.
- **Upper bound solution**. An upper bound solution is where the solution (objective function value) of the problem (22) is guaranteed to be less or equal to an upper bound. This upper bound might be a simplification of the original problem and hopefully, the upper bound is computationally tractable.

The focus is on the lower bound solution as well as the upper bound solution strategy in the next sections.

3.2 A Lower Bound

A lower bound solution to (22) is any solution that is feasible. This means that all solutions where $\delta \in \mathcal{U}_p$ are feasible since there is no constraint on θ. However, the solution might not be optimal or of value and, thus, the task is to find good candidate solutions, which, at least practically, would perform better than a nominally trained neural network for a perturbed input. One way of finding good candidate perturbations δ is to try and solve the inner maximization at each iteration step when training the neural network. Training neural networks are usually performed with the gradient descent, as shown in equation (11), or its variants such as, for example, stochastic gradient descent.

Before trying to solve the inner maximization, gradient descent needs to be applied to the problem. Without loss of generality, the standard gradient descent is used. Applying gradient descent on equation (22) and calculating the next iteration step $i+1$ with learning rate η and optimizing over model parameters θ yields

$$\begin{aligned}\theta_{i+1} &= \theta_i - \eta\nabla_\theta\Big(\frac{1}{N}\sum_{n=1}^{N}\max_{\delta_n\in\mathcal{U}_p}\mathcal{L}(h(x_n+\delta_n,\theta),y_n)\Big)\\ &= \theta_i - \eta\frac{1}{N}\sum_{n=1}^{N}\nabla_\theta\max_{\delta_n\in\mathcal{U}_p}\mathcal{L}(h(x_n+\delta_n,\theta),y_n).\end{aligned} \tag{23}$$

Now, for obtaining a good approximation, the gradient at the maximum of $\mathcal{L}(h(x_n + \delta_n, \theta), y_n)$ needs to be calculated. Intuitively, since the maximization is over the perturbation δ and the gradient is for the parameters θ, it may seem trivial to do the following for each datapoint n, namely first finding the maximum

$$\delta_n^* = \arg\max_{\delta_n\in\mathcal{U}_p}\mathcal{L}(h(x_n+\delta_n,\theta),y_n), \tag{24}$$

and then evaluating the gradient at δ_n^* and calculating the next step

$$\theta_{i+1} = \theta_i - \eta\frac{1}{N}\sum_{n=1}^{N}\nabla_\theta\mathcal{L}(h(x_n+\delta_n^*,\theta),y_n). \tag{25}$$

However, this calculation is not necessarily true and known a priori (Madry et al., 2018). Hence, the following equality does not hold for all cases

$$\nabla_\theta \max_{\delta \in \mathcal{U}_p} \mathcal{L}(h(x+\delta, \theta), y) = \nabla_\theta \mathcal{L}(h(x+\delta^*, \theta), y). \tag{26}$$

Therefore, certain preconditions on the function $\mathcal{L}$ and the uncertainty space $\mathcal{U}_p$ needs to be set. For example, consider the function $\phi(x, z) = -|x||z|$ which obtains its maximum at $z = 0$. Clearly the function $\phi(x, z)$ is continuous but is not differentiable with respect to x at the maximum $z = 0$. However, it has a *directional derivative* that can be defined generally for vectors, given in Definition 1.

Definition 1 (Directional Derivative). *Let f be a continuous function and $f : \mathbb{R}^m \to \mathbb{R}$. Also, let $x, d \in \mathbb{R}^m$ be nonzero vectors. The directional derivative of f at x along direction d is said to exist if the following limit exists*

$$f'(x; d) = \lim_{\lambda \to 0} \frac{f(x + \lambda d) - f(x)}{\lambda}.$$

The classical Danskin's theorem from optimization has been loosely used to justify the previous computations (24) and (25) of the gradient at the maximum, but Danskin (1967) originally formulated the theorem for a minimum and directional derivative requiring the functions to be continuously differentiable, which is not generally true for neural networks. Similarly, Bertsekas (1999) presents the classical Danskin's theorem for convex functions but assumes only continuity, but as neural networks are not convex functions, this formulation is also not directly applicable. Madry et al. (2018) adapts Danskin's theorem for neural networks but also assumes continuous differentiability everywhere in $\mathcal{U}_p$ and does not provide a proof for their modified theorem. Therefore, Danskin's theorem in the context of robust neural networks needs a proper reformulation. Here, Danskin's theorem is stated for maximization and with the directional derivative on continuous functions as in the following theorem.

Theorem 1 (Modified Danskin's Theorem). *Let $\mathcal{S}$ be a nonempty compact topological space and function $\phi : \mathbb{R}^m \times \mathcal{S} \to \mathbb{R}$ be such that $\phi(x, z)$ is continuous for all $x \in \mathbb{R}^m$ and $z \in \mathcal{S}$. Also, let $\mathcal{Z}(x) = \{z : z \in \arg\max_z \phi(x, z)\}$ and*

$$f(x) = \max_{z \in \mathcal{S}} \phi(x, z).$$

Then the max-function f is directionally differentiable in direction $d \in \mathbb{R}^m$ at $z \in \mathcal{Z}(x)$ and

$$f'(x; d) = \sup_{z \in \mathcal{Z}(x)} \phi'(x, z; d).$$

Proof. The proof here follows generally the steps in Bernhard and Rapaport (1995). Some details are also included from Danskin (1967) and Bertsekas (1999). Let $x, d \in \mathbb{R}^m$ and $z \in \mathcal{Z}(x)$ as well as $\tilde{z} \in \mathcal{Z}(x + \lambda y)$ be the respective maximizers. Also, let $\lambda \in \mathbb{R}$. By the definition of function f, it holds that

$$\begin{aligned}\frac{f(x+\lambda d)-f(x)}{\lambda} &= \frac{\phi(x+\lambda d,\tilde{z})-\phi(x,z)}{\lambda} \\ &= \frac{\phi(x+\lambda d,\tilde{z})-\phi(x+\lambda d,z)}{\lambda}+\frac{\phi(x+\lambda d,z)-\phi(x,z)}{\lambda} \\ &\geq \frac{\phi(x+\lambda d,z)-\phi(x,z)}{\lambda}.\end{aligned} \tag{27}$$

Taking the limit as $\lambda \to 0$, gives

$$f'(x;d) \geq \phi'(x,z;d). \tag{28}$$

Since inequality (28) holds for all $z \in \mathcal{Z}(x)$, it can be concluded that

$$f'(x;d) \geq \sup_{z\in\mathcal{Z}(x)} \phi'(x,z;d). \tag{29}$$

Next, let $\{\lambda_k\}$ be a sequence of positive scalars that converge to zero. Also, let $x_k = x + \lambda_k d$ be such that $x_k \in \mathbb{R}^m$ and z_k be the maximizer for each k, i.e., $z_k \in Z(x_k)$. Statement (28) implies that $\frac{f(x+\lambda_k d)-f(x)}{\lambda_k}$ is bounded below. Thus, there exists a constant C such that for large enough k, it holds that

$$f(x+\lambda_k d) \geq f(x) + C\lambda_k, \tag{30}$$

and therefore,

$$\liminf_{k\to\infty} f(x+\lambda_k d) \geq f(x). \tag{31}$$

Since $\mathcal{S}$ is compact it holds that a subsequence $\{z_k\}$ converges to $\hat{z}$. Without loss of generality, it may be assumed that the whole sequence $\{z_k\}$ converges to $\hat{z}$. Thereby, it holds that

$$\begin{aligned}f(x) &\geq \phi(x,\hat{z}) \\ &\geq \limsup_{k\to\infty} \phi(x+\lambda_k d, z_k) \\ &\geq \liminf_{k\to\infty} \phi(x+\lambda_k d, z_k) \\ &\geq f(x),\end{aligned} \tag{32}$$

Statement (32) proves that $\hat{z} \in \mathcal{Z}(x)$ and also the existence of the limit

$$\lim_{k\to\infty} \phi(x+\lambda_k d, z_k) = f(x). \tag{33}$$

Again, by the definition of $f(x)$ it holds that

$$\begin{aligned}\frac{f(x+\lambda_k d)-f(x)}{\lambda_k} &= \frac{\phi(x+\lambda_k d,z_k)-\phi(x,z)}{\lambda_k} \\ &= \frac{\phi(x+\lambda_k d,z_k)-\phi(x,z_k)}{\lambda_k}+\frac{\phi(x,z_k)-\phi(x,z)}{\lambda_k} \\ &\leq \frac{\phi(x+\lambda_k d,z_k)-\phi(x,z_k)}{\lambda_k}.\end{aligned} \tag{34}$$

As $\{\lambda_k\}$ is assumed to be a converging subsequence to zero and it was proven that $\{z_k\} \to \hat{z} \in \mathcal{Z}(x)$ and the limit (33) exists, it holds that

$$\lim_{k\to\infty} \frac{f(x+\lambda_k d) - f(x)}{\lambda_k} \leq \lim_{k\to\infty} \frac{\phi(x+\lambda_k d, z_k) - \phi(x, z_k)}{\lambda_k}, \tag{35}$$

finally resulting in

$$f'(x; d) \leq \phi'(x, \hat{z}; d). \tag{36}$$

Equation (36) holds for all $\hat{z} \in \mathcal{Z}(x)$ and therefore also for the supremum in $\mathcal{Z}(x)$ and thus

$$f'(x; d) \leq \sup_{\hat{z}\in\mathcal{Z}(x)} \phi'(x, \hat{z}; d). \tag{37}$$

Only way for both the lower bound (29) and upper bound (40) to hold is that

$$f'(x; d) = \sup_{z\in\mathcal{Z}(x)} \phi'(x, z; d), \tag{38}$$

hence the proof is complete. □

The interpretation behind Theorem 1 is that even though a gradient may not exist for a function at a given maximum point, the directional derivative can be determined at the maximum, which determines the rate of change in that direction. Also, the theorem implicitly states the existence of a maximum as the space $\mathcal{S}$ is compact and ϕ is continuous in z, which is a result of the fundamental Weistrass's Theorem in optimization. A more powerful statement can be made about the gradients with some assumptions on the differentiability and the number of maximums, which is summarized in Corollary 1.1.

Corollary 1.1. *If $\mathcal{Z}(x) = \{z^*\}$ is a singleton and ϕ is differentiable for all $x \in \mathbb{R}^m$, then*

$$\nabla_x f(x) = \nabla_x \phi(x, z^*).$$

Proof. If $\mathcal{Z}(x)$ is a singeleton, then z^* is the single maximizer for $\phi(x, z)$ for the corresponding x. Thereby, the supremum is at z^* and it holds that

$$f'(x; d) = \phi'(x, z^*; d). \tag{39}$$

Moreover, if ϕ is differentiable for all $x \in \mathbb{R}^m$, then

$$f'(x; d) = \phi'(x, z^*; d) = d^T \nabla_x \phi(x, z^*), \tag{40}$$

implying that $\nabla_x f(x) = \nabla_x \phi(x, z^*)$. □

For the context of neural networks, Corollary 1.1 is not always true, since ReLU and max-pooling functions are not continuously differentiable everywhere. Thus, Corollary 1.1 does not generally hold throughout the working domain. However, as explained by Madry et al. (2018), gradients are local objects and evaluating the gradient at a point that is not continuously differentiable is unlikely when training the neural network and thus in practice, the gradient can almost always be evaluated. In addition, neural network frameworks, such as PyTorch, are usually capable of evaluating gradients at non-differentiable points with some prescribed rules (Paszke et al., 2019). In the worst case, for a point of non-differentiability, one can always resort back to the directional derivative and Theorem 1. Returning to equation (25) and the training of robust neural networks, the gradient exists for the maximum and can be evaluated if the constraints are relaxed, i.e., it is assumed that the neural network is differentiable everywhere or at least locally.

Lastly, to prove that the gradient descent method is moving towards the minimum of the loss function for the robust case, the stepping direction needs to be descending. A direction d is said to be a *descent direction* if it satisfies Definition 2.

Definition 2 (Descent Direction)**.** *Let f be a differentiable function in $\mathbb{R}^m$ and $f : \mathbb{R}^m \to \mathbb{R}$. Also, let $x, d \in \mathbb{R}^m$. A direction d is said to be a descent direction if*

$$d^T \nabla_x f(x) < 0.$$

Using Definition 2, it can be proven that the robust gradient descent training of equation (23) minimizes the loss, hence the following corollary.

Corollary 1.2. *A direction $d = -\nabla_\theta \mathcal{L}(h(x + \delta^*, \theta), y)$ is a descent direction for the loss function $\mathcal{L}(h(x + \delta^*, \theta), y)$ if d is not a zero vector.*

Proof. Inserting d into Definition 2, gives

$$d^T \nabla_\theta \mathcal{L}(h(x + \delta^*, \theta), y) = -||\nabla_\theta \mathcal{L}(h(x + \delta^*, \theta), y)||_2^2 \leq 0. \tag{41}$$

The last inequality becomes strict when we assume d is not a zero vector by the definition of a norm. □

So far, evaluating the gradient at the perturbation δ^*, which causes the maximum loss within the uncertainty set $\mathcal{U}_p$, produces an iterative method that converges towards a minimum. Hence, the gradient descent method can be used for robust learning. However, it has been assumed that the perturbation δ^* that maximizes the loss is known. This task, again, is not trivial and one has to resort to techniques from constrained optimization. That is, one must solve

$$\delta^* = \arg\max_{\delta \in \mathcal{U}_p} \mathcal{L}(h(x + \delta, \theta), y). \tag{42}$$

for each datapoint (x_n, y_n) during the training.

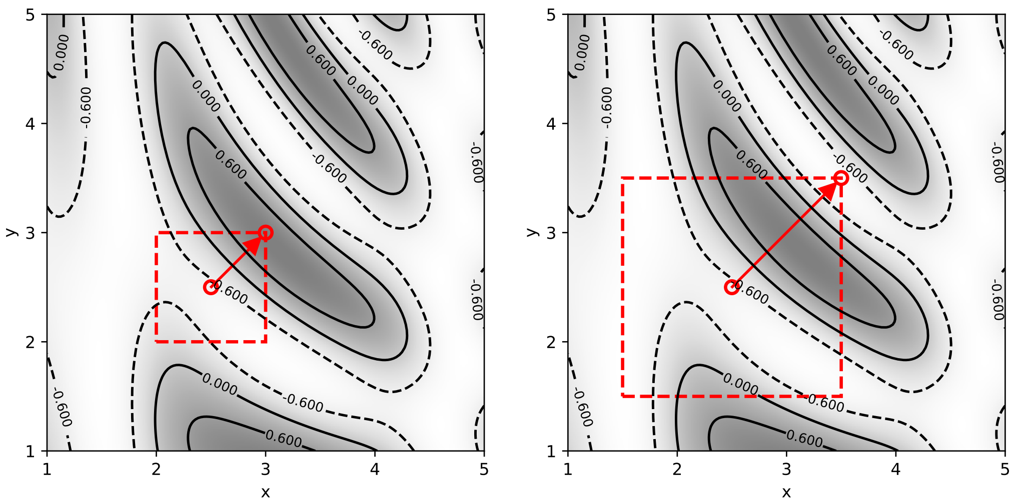
One simple approximate method already introduced is the *Fast Gradient Sign Method* (FGSM) where the idea is to evaluate the gradient at x and then move in its direction as much as possible within the feasible domain $\mathcal{U}_p$. FGSM is an one-step method, as it requires only a single calculation step to produce a candidate perturbation δ^*, as

$$\delta^* = \varepsilon \operatorname{sign}(\nabla_x \mathcal{L}(h(x, \theta), y)). \tag{43}$$

The final adversarial sample is then given as

$$x^* = x + \delta^*. \tag{44}$$

The FGSM is a fast and computationally inexpensive way of creating adversarial examples, but it does not guarantee to find a maximum inside the feasible domain $\mathcal{U}_p$. Basically, the FGSM evaluates the gradient and takes a maximum step until it reaches the border of the domain. If the maximum is not at the border FGSM will always miss the maximum. This is illustrated in Figure 3.



(a) FGSM method with $\varepsilon = 0.5$ (b) FGSM method with $\varepsilon = 1.0$

Figure 3: Example of the FGSM method with two different perturbation sizes ε within the uncertainty set $\mathcal{U}_\infty$. The example function here is defined as $\mathcal{L}(x, y) = -\sin(x)^{10} - \cos(10 + yx)\cos(x)$. The red arrow points to created adversarial example $x + \delta^*$.

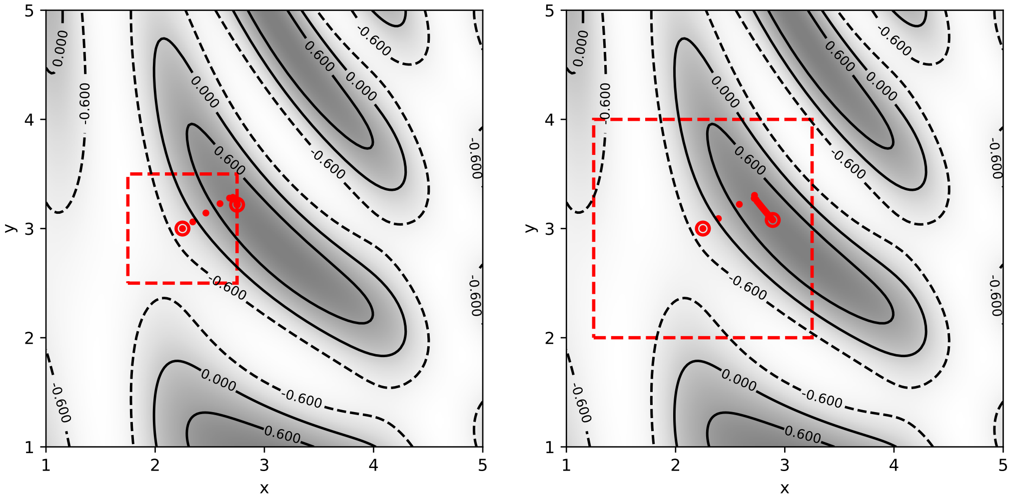
A better method to solve the inner maximization is the *Projected Gradient Descent* (PGD), which was briefly introduced in section 2.2. The PGD solves the inner maximization as

$$\delta^* = \mathcal{P}(x + \alpha \nabla_x \mathcal{L}(h(x, \theta), y)) - x, \tag{45}$$

where α is a step size parameter and the final adversarial sample can be evaluated with (44). In contrast to FGSM, PGD is a multi-step method that is repeated for a fixed number of iterations or, alternatively, stopped after a stopping criterion is met. The fixed iterative method of PGD can be written as

$$x_{i+1} = \mathcal{P}(x_i + \alpha \nabla_x \mathcal{L}(h(x_i, \theta), y)) \quad \text{for} \quad i = 1 \ldots I. \tag{46}$$

PGD is able to find the local maximum inside the feasible domain, given a sufficient number of iterations I and a suitable step size parameter α. An example of this is given in Figure 4, where the PGD converges to the best candidate on the boundary



(a) PGD method with $\varepsilon = 0.5$ (b) PGD method with $\varepsilon = 1.0$

Figure 4: Example of the PGD method with two different perturbation sizes ε within the uncertainty set $\mathcal{U}_\infty$. The example function here is defined as $\mathcal{L}(x, y) = -\sin(x)^{10} - \cos(10 + yx)\cos(x)$. The smaller dots show the steps of the iteration while the circles show the start x and the final $x + \delta$.

of the domain and then to the local maximum inside the domain.

So far, it has been concluded that applying gradient descent to the saddle point problem (22) generally results in descent directions, and thus the method moves towards a minimum. Furthermore, it was shown that adversarial training works theoretically for networks that are differentiable everywhere. Common activation functions such as ReLU and max-pooling introduce some discrepancies in the theoretical framework since they are not differentiable everywhere. However, one rarely ends up in non-differentiable points and modern deep learning frameworks have prescribed rules that handle non-differentiable functions in the backward pass. Thus, in practice, lack of differentiability at some points is not a problem when training neural networks. Another source of discrepancy comes from solving the maximizer for the loss function at the given iteration step. For solving the gradient at the maximum, some method for calculating the maximum needs to be used, for example, FGSM or PGD, which finds feasible and good candidates for the maximizer of the loss function for the given set of weights at that iteration. The calculated maximum is not necessarily guaranteed to be the true maximizer within the feasible domain. Nonetheless, the general procedure for adversarial training can therefore be summarized in Algorithm 1.

As adversarial training does not guarantee to find the local minimum for the problem (22) nor does it guarantee that the adversarial example created during the process is the worst (maximizer of the loss function), it can only be considered as a lower bound solution. Any perturbation created during the iteration step is just a feasible solution to the inner maximizer and, thus, the maximization step is also only approximate.

Algorithm 1: General stochastic gradient algorithm for adversarial training.

for minibatch B in dataset **do**
  Initialize gradient g $= \nabla_\theta \mathcal{L}(h(x,\theta), y) = 0$ for minibatch B
  for each datapoint (x_n, y_n) in minibatch B **do**
    find $\delta_n^* = \arg\max_{\delta_n \in \mathcal{U}} \mathcal{L}(h(x_n + \delta_n, \theta), y_n)$ with PGD, FGSM etc.
    update gradient: $g = g + \nabla_\theta \mathcal{L}(h(x_n + \delta_n^*, \theta), y_n)$
  end
  update parameters: $\theta = \theta - \frac{\eta}{|B|} g$
end

Hence, it can be concluded that adversarial training provides a lower bound solution to the saddle point problem.

3.3 An Approximate Upper Bound

An upper bound solution to the saddle point problem (22) is a solution, which guarantees the solution to be under a certain threshold, i.e., upper bound, and can be stated as in inequality (21). Finding an upper bound to (22) is not a trivial task and no general way of finding an upper bound exists, nor are there guarantees that it exists. Further complicating the problem, the common technique from robust optimization of finding a robust counterpart, highly depends on the neural network architecture and the uncertainty set $\mathcal{U}_p$. A *robust counterpart* is a transformation of the original problem of optimization under uncertainty to a solvable robust counterpart which is a tractable deterministic formulation. The solution can be exact, but for the saddle point problem (22), only an approximate upper bound can be obtained for the general case.

As a reminder, the goal here is to transform the saddle point problem (22) to another single minimization problem which guarantees the solution to be upper bounded, such that

$$\min_\theta \frac{1}{N} \sum_{n=1}^{N} \max_{\delta_n \in \mathcal{U}} \mathcal{L}(h(x_n + \delta_n, \theta), y_n) \leq \min_\theta \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}_{UB}(h(x_n, \theta), y_n, \varepsilon), \tag{47}$$

where $\mathcal{L}_{UB}$ is a new loss function which upper bounds the original problem. This loss function already considers the inner maximization over δ and therefore one does not need to solve a maximization problem.

To find an upper bound for the saddle point problem Bertsimas et al. (2021a) provided a new method for finding an approximate and an exact robust upper bound. Here, the derivation for the *Approximate Robust Upper Bound* (aRUB) is shown and follows the derivation done in Bertsimas et al. (2021a,b) with some more details. The focus here is only on the aRUB and not on the exact *Robust Upper Bound* (RUB). The RUB method depends heavily on the input dimension size and the network architecture and thus does not scale in practice. Also, the RUB only works for the $\mathcal{U}_1 = \{ \delta : ||\delta||_1 \leq \varepsilon \}$ uncertainty set. For this reason, the RUB is omitted in this thesis.

The key idea of aRUB is to linearize the neural network h close to the input data x, hence

$$h(x + \delta, \theta) \approx h(x, \theta) + \delta \nabla_x h(x, \theta). \tag{48}$$

This linearization of the neural network with respect to the perturbation δ is considered to be better than linearizing the loss function $\mathcal{L}$ since the loss function is highly nonlinear and neural networks are partly linear close to the original point x. Further, linearizing the loss function with respect to the perturbation

$$\mathcal{L}(h(x + \delta, \theta), y) \approx \mathcal{L}(h(x, \theta), y) + \delta \nabla_x \mathcal{L}(h(x, \theta), y), \tag{49}$$

and considering the uncertainty set $\mathcal{U}_\infty = \{ \delta : ||\delta||_\infty \leq \varepsilon \}$, it can be concluded that the perturbation that maximizes (49) is $\delta^* = \varepsilon \, \text{sign}(\nabla_x \mathcal{L}(h(x, \theta), y))$, which is the FGSM used in adversarial training (Bertsimas et al., 2021a).

Inserting the linear approximation of the neural networks h into the cross entropy loss of equation (10) with K number of classes , yields

$$\log(\sum_{j=1}^{K} e^{(e_k - e_y)^T h(x+\delta,\theta)}) \approx \log(\sum_{j=1}^{K} e^{(e_k - e_y)^T h(x,\theta) + \delta^T (\nabla_x (e_k - e_y)^T h(x,\theta))}), \tag{50}$$

where e_k and e_y are zero vectors except for the k and y index locations, which have a value of one. Further noting $c_{ky} = e_k - e_y$ a new approximate linearized saddle point problem is obtained as

$$\min_{\theta} \frac{1}{N} \sum_{n=1}^{N} \max_{\delta_n \in \mathcal{U}_p} \log(\sum_{j=1}^{K} e^{c_{ky_n}^T h(x_n,\theta) + \delta_n^T (\nabla_x c_{ky_n}^T h(x_n,\theta))}). \tag{51}$$

Since both exponents and logarithms are both strictly monotonically increasing functions, the inner maximization of (51) is equivalent to maximizing only the exponent for each datapoint (x_n, y_n) and plugging the maximizing arguments back to the original problem. Therefore, the problem can be simplified to

$$\max_{\delta \in \mathcal{U}_p} c_{ky}^T (h(x, \theta) + \delta^T \nabla_x c_{ky}^T h(x, \theta)), \tag{52}$$

without loss of generality. Since the uncertainty set $\mathcal{U}_p$ is defined as a bounded normed space, the solution to the maximization depends on the used norm. For this reason, introducing the concept of the *dual norm* is needed in order to solve the maximization problem (52).

Definition 3 (Dual Norm (Boyd and Vandenberghe, 2004)). *Let $|| \cdot ||_p$ be a norm in $\mathbb{R}^m$. The dual norm $|| \cdot ||_q$ is defined as*

$$||z||_q = \sup\{z^T x : ||x||_p \leq 1\},$$

where $x, z \in \mathbb{R}^m$.

The concept of the dual norm is rather more advanced and general than presented here with Euclidian space $\mathbb{R}^m$, but it suffices for the purpose of the derivation to be presented. The dual norm is formulated as an LP with the constraint being maxed at one if it is assumed that the maximum exists. Since the maximization problem (52) is a LP for δ, the definition of the dual norm can be used in solving (52). Hence, considering the first term of equation (52) to be constant, the maximization problem is transformed to

$$\max_{\delta \in \mathcal{U}_p} c_{ky}^T (h(x,\theta) + \delta^T \nabla_x h(x,\theta)) = c_{ky}^T h(x,\theta) + \varepsilon ||\nabla_x c_{ky}^T h(x,\theta)||_q. \tag{53}$$

So far there has been no relationship established between p and q-norms and thus the problem is still unsolved. Luckily, based on Hölder's inequality, the following theorem can be stated.

Theorem 2 (Dual Norm Theorem)**.** *Let $p, q \in \mathbb{N}$ be Hölder conjugates so that $\frac{1}{p} + \frac{1}{q} = 1$, then the L_p norm is the dual of the L_q norm and vice versa, i.e., the following relations hold*

$$||z||_q = \sup\{z^T x : ||x||_p \leq 1\} \text{ and } \frac{1}{p} + \frac{1}{q} = 1$$

Proof. A proof can be found at Kreyszig (1989) for l-spaces, that is infinite-dimensional normed vector spaces. Since L-spaces are finite-dimensional, the proof also holds for L-spaces.

Theorem 2 gives a clear relation between the dual norm and the original norm. Equation (53) has thereby a solution if we assume a maximum exists and the supremum can be evaluated as a maximum. Here, defining the uncertainty set $\mathcal{U}_p$ already defines the dual norm due to Hölder's conjugate relation. For example, if the uncertainty space is defined with norm $p = 1$ the corresponding dual norm is $q = \infty$ and respectively if $p = 2$ then $q = 2$.

Returning to the original cross entropy loss function the following steps can be made for approximately upper bounding the solution

$$\begin{aligned} \max_{\delta \in \mathcal{U}_p} \log(\sum_{j=1}^{K} e^{c_{ky}^T h(x+\delta,\theta)}) &\approx \max_{\delta \in \mathcal{U}_p} \log(\sum_{j=1}^{K} e^{c_{ky}^T h(x,\theta) + \delta^T \nabla_x (c_{ky}^T h(x,\theta))}) \\ &\leq \log(\sum_{j=1}^{K} e^{\max_{\delta \in \mathcal{U}_p} c_{ky}^T h(x,\theta) + \delta^T \nabla_x (c_{ky}^T h(x,\theta)))}) \\ &= \log(\sum_{j=1}^{K} e^{c_{ky}^T h(x,\theta) + \varepsilon ||\nabla_x c_{ky}^T h(x,\theta)||_q}). \end{aligned} \tag{54}$$

Finally, the saddle point problem has been transformed to an approximate upper bound by using a first-order linearization of the neural network h and solving the inner maximization via the dual norm. Hence, the approximate robust upper bound results in the following optimization problem

$$\min_{\theta} \frac{1}{N} \sum_{n=1}^{N} \log(\sum_{j=1}^{K} e^{c_{kyn}^T h(x_n,\theta)+\varepsilon||\nabla_x c_{kyn}^T h(x_n,\theta)||_q}). \tag{55}$$

For solving problem (55), common optimization methods can be used like gradient descent or its variants.

To train a neural network with the new approximate robust upper bound, it resolves to implement the loss function of

$$\mathcal{L}_{UB}(h(x,\theta), y, \varepsilon) = \log(\sum_{j=1}^{K} e^{c_{ky}^T h(x,\theta)+\varepsilon||\nabla_x c_{ky}^T h(x,\theta)||_q}), \tag{56}$$

into a deep learning framework. The only cumbersome task is to evaluate the gradient of the neural network with respect to the input. Luckily, modern deep learning frameworks like PyTorch have automatic functionalities to evaluate gradients of functions and neural networks automatically by constructing the computational graph and necessary gradients for backpropagation (Paszke et al., 2019). The most important aspect in the automatization of the gradient computation and the aRUB itself is that it is general to all neural networks, thus the developer of the neural network does not need to worry about the architecture of the neural network for computing the gradient term. Another noteworthy point is that the loss function (56) is computationally scalable and requires only a reasonable amount of additional memory and computations. Thereby, the training of neural networks with the approximate robust upper bound loss is feasible from a computation standpoint.

4 Experimental Setup

The robustness of adversarial training and the approximate robust upper bound is assessed in practice against adversarial attacks. The tests are performed on three publicly available medical datasets on a common convolutional neural network architecture. Two types of adversarial attacks, the FGSM and the PGD, are used for evaluating the robustness on standard metrics such as accuracy, sensitivity and specificity against different perturbation sizes of the attacks. First, the neural networks are trained, after which the neural networks are tested against adversarial attacks constructed from the test data. The following methods are implemented and tested:

- **aRUB**. The approximate robust upper bound by Bertsimas et al. (2021a,b) with two different norms, $p = 1$ and $p = \infty$ on the uncertainty set $\mathcal{U}_p$.
- **Adversarial training**. Two adversarial trained models with PGD and FGSM on the $\mathcal{U}_\infty$ uncertainty set.
- **Nominal training**. A normally trained model without any defense strategies against adversarial attacks.

All together five different models are tested. The experiments comprise three different medical datasets, which are described in section 4.1. A common computer-vision neural network architecture, described in section 4.2, is used in the tests. The details of the models are explained in section 4.3 and the used attacks and their parameters in section 4.4. In section 4.5, the training process is described and finally, in section 4.6, the practical implementation and used hardware are described.

4.1 Datasets

Three different datasets are used for testing the five models. Each dataset is split into a train and test set with a ratio of 0.8/0.2, that is 80% of the data is used for training the models and 20% is used for testing and evaluating the robustness of the model. The split was done randomly 10 times and each model was trained for each corresponding split, thus one model was trained 10 times for each dataset.

The first dataset is the diabetic retinopathy dataset (Kaggle, 2015), which is a collection of RGB images from the retina, i.e., the back of the eye. *Diabetic retinopathy* is a disease that damages the retina and can eventually cause blindness. As the name suggests, the leading cause of diabetic retinopathy is diabetes, where high blood sugar level causes blockage of blood vessels cutting of the blood supply (Kaschke et al., 2014). This causes the eye to try and grow new blood vessels, which are more likely to not develop properly and are more prone to leaking, thus, damaging the eye (Kaschke et al., 2014). Figure 5 shows an example of the image data for a normal retina and a diabetic retina.

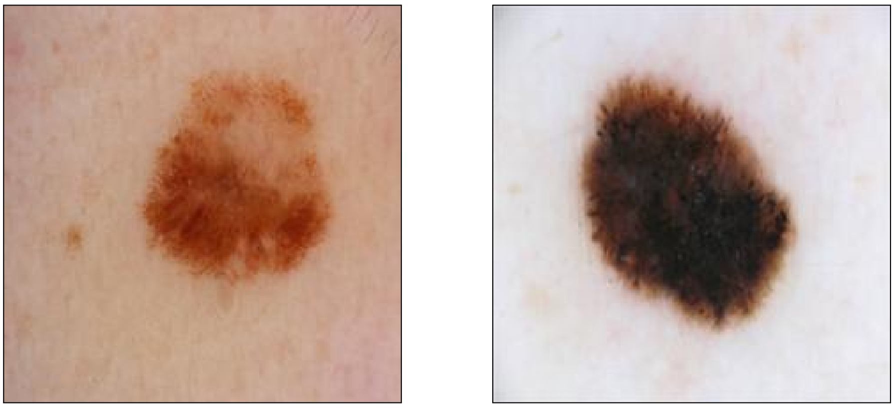
(a) Normal retina.

(b) Damaged retina.

Figure 5: Examples of healthy and damaged retina for the diabetic retinopathy dataset (Kaggle, 2015).

The original image dataset consists of a total of 88 702 images of both the right and left eye. The data was originally labeled with integers in the range of $[0, 4]$, where a label of 0 indicated no damage and a healthy retina, and 4 indicated a severely damaged retina. The values between 0 and 4 indicated the severity. The original dataset had an uneven distribution of classes where the vast majority of labels had a label of 0. To balance the uneven classes, the data were grouped into two classes, where original labels 0 and 1 were considered as 0 (no damage) and original labels 2,3 and 4 were grouped to label 1 (damaged retina). The labels were thereby simplified to "normal=0" and "damaged=1" classes. Even after grouping the five classes into two classes, the data was still unevenly distributed among the two classes and, therefore, the data was augmented by rotations and mirroring of the images. The damaged class was augmented by 90 degrees clockwise and counterclockwise rotations as well as horizontal and vertical image mirroring. The normal class was only augmented with 90 degrees clockwise and counterclockwise rotations. All the image pixel values were scaled to be in the range $[0, 1]$ and all the images resized to 223×223 pixels. The total augmented dataset consisted of 85 770 damaged class images and 143 096 normal class images, equaling a total of 228 866 images. For splitting the test and train data, the data was split in such a way that no augmented image was in the test set if the original or other augmented variations of it were in the train data. This was done to assure that the neural network had not seen the test data in any form before testing.

The second image dataset contains RGB images of nevi (moles) (ISIC, 2019). The original dataset contained nine different diagnoses such as melanoma, melanocytic nevus and benign keratosis as examples. Only melanoma and melanocytic nevus was selected for training and testing. *Melanoma* is a type of skin tumor, which develops from melanocytes and is a malignant type of tumor (Hall and Hall, 2017). *Melanocytic nevus* is a skin lesion or a mole that is benign, i.e., harmless (Hall and Hall, 2017). Figure 6 shows an example image of a melanocytic nevus and a nevus with melanoma.



(a) Benign nevus.

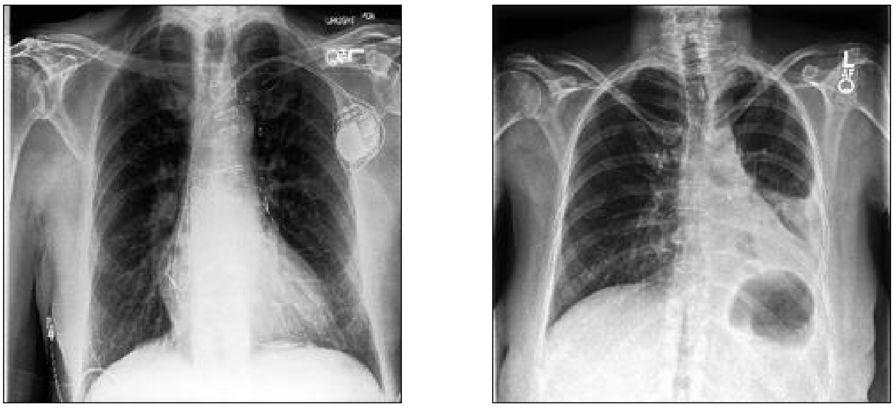
(b) Nevus with melanoma.

Figure 6: Examples of benign nevus and nevus diagnosed with melanoma from the nevus dataset (ISIC, 2019).

The stripped-down dataset for the nevi consisted of 4 522 images of melanoma and 12 875 of melanocytic nevi. Since the data consisted of fewer images of nevi with melanoma, only the first 4 522 melanocytic nevi images were used to balance the classes. The data was given the following class labels "0=melanocytic nevus" and "1=melanoma". For both classes, the data was augmented with 90 degrees clockwise and counterclockwise rotations, vertical and horizontal image mirroring as well as 180 degrees rotations. All the image pixel values were scaled to be in the range of $[0, 1]$ and all the images were resized to 223×223 pixels. The final dataset consisted of a total of 54 264 images.

The third medical image dataset is a set of chest X-ray images of normal lungs and lungs diagnosed with pneumothorax (Irvin et al., 2019). *Pneumothorax* is caused by air trapped outside the lungs and inside the chest in the pleural space, causing the lung to partially collapse (Bourke, 2015). One cause of pneumothorax are respiratory diseases such as chronic obstructive pulmonary disease or cystic fibrosis (Bourke, 2015). Figure 7 shows an example of a normal chest X-ray and a chest X-ray of lungs diagnosed with pneumothorax.

The dataset from Irvin et al. (2019) is called the "cheXpert" dataset and it contains more than 224 316 chest X-rays of 65 240 patients with various labels such as pneumothorax, pleural effusion, pneumonia, etc., as well as "no finding" images. The data was automatically labeled from the statements from medical reports with labels positive, negative and unsure. From all the data, only the images labeled with a positive for pneumothorax or positive for "no finding" were selected. The images with pneumothorax were labeled as 1 and the images with no finding as 0. No augmentation was used as the amount of data was large enough for training a neural network and almost evenly balanced in the two classes. The data consisted of 17 313 pneumothorax images and 16 627 images of no finding, i.e., normal images. All the chest X-ray images were converted to a RGB image from the grayscale image and all



(a) Normal. (b) Pneumothorax.

Figure 7: Examples of a normal chest X-ray and a chest X-ray with pneumothorax (Irvin et al., 2019).

the pixel values were normalized to be in the range $[0, 1]$. All the images were also resized to 223×223 pixels.

4.2 Neural Network

The same neural network model and architecture was used for all five models. In difference to Finlayson et al. (2019), Ma et al. (2021) and Hirano et al. (2021), the used architecture was a ResNet-18 convolutional neural network instead of the ResNet-50 convolutional neural network. The ResNet-50 is a deeper neural network than the ResNet-18 but has a similar base structure. The ResNet-18 was chosen because it has fewer parameters and is therefore faster to train. The ResNet-18 is a convolutional neural network consisting of 5 blocks. Each block consists of two two-dimensional convolutional layers followed by batch normalization after each convolutional layer and with a ReLU activation function between the convolutional layers. After the first block, between each block, there is a skip connection meaning that input vector x is added to the output of the block without any operation on the input vector x, i.e.,

$$z = \mathcal{B}(x) + x, \tag{57}$$

where $\mathcal{B}(x)$ is the output of the block. The output of the ResNet-18 is a vector of size 512. After the ResNet-18 the output is fed to a custom fully connected four-layer neural network with ReLU activation functions between each layer. The details of the full architecture are shown in Table 1 and the details of the original ResNet-18 can be found at He et al. (2016).

In Table 1, the "Conv2D()" function stands for a convolutional layer as in equation (6). The first argument describes the output channels and the second kernel size ([height, width]). Similarly, "Max-Pool()" is the pooling equation defined in (7) and the arguments are the height and width ([height, width]). The "BatchNorm2D()" is the batch normalization operator shown in equation (8), where the same hyperparameter of $\epsilon = 10^{-5}$ is used in all layers. The last fully connected layers are noted as "Linear()" where the first argument is the input dimension and the second the output dimension. Bias terms were used for all fully connected layers. No softmax activation function was used in the last layer as the cross entropy loss or its aRUB variant was used as the loss function. The output dimension of the last layer is two as the neural network is a binary classifier.

Table 1: Model architecture.

Block Name	**Layers**
Block 1	$\begin{bmatrix} \text{Conv2D(64, [7, 7])} \\ \text{BatchNorm2D()} \\ \text{Max-Pool([3, 3])} \end{bmatrix} \times 1$
Block 2	$\begin{bmatrix} \text{Conv2D(64, [3, 3])} \\ \text{BatchNorm2D()} \\ \text{ReLU()} \\ \text{Conv2D(64, [3,3])} \\ \text{BatchNorm2D()} \end{bmatrix} \times 2$
Block 3	$\begin{bmatrix} \text{Conv2D(128, [3, 3])} \\ \text{BatchNorm2D()} \\ \text{ReLU()} \\ \text{Conv2D(128, [3, 3])} \\ \text{BatchNorm2D()} \end{bmatrix} \times 2$
Block 4	$\begin{bmatrix} \text{Conv2D(256, [3, 3])} \\ \text{BatchNorm2D()} \\ \text{ReLU()} \\ \text{Conv2D(256, [3, 3])} \\ \text{BatchNorm2D()} \end{bmatrix} \times 2$
Block 5	$\begin{bmatrix} \text{Conv2D(512, [3, 3])} \\ \text{BatchNorm2D()} \\ \text{ReLU()} \\ \text{Conv2D(512, [3, 3])} \\ \text{BatchNorm2D()} \end{bmatrix} \times 2$
FC	$\begin{bmatrix} \text{Linear(512, 200)} \\ \text{ReLU()} \\ \text{Linear(200, 200)} \\ \text{ReLU()} \\ \text{Linear(200, 200)} \\ \text{ReLU()} \\ \text{Linear(200, 2)} \end{bmatrix} \times 1$

4.3 Models

Five different models were tested for all three datasets. For both the aRUB models and adversarial training models, the perturbation size was set to $\varepsilon = 0.001$ in (15). The five models were:

- **Nominal**. The nominal model is the standard model, which is trained on the clean data (no perturbations in the data) with the normal cross entropy loss in equation (10).
- **aRUB-L_∞**. The model is trained on clean data with the approximate robust upper bound loss function of equation (56) where $q = \infty$. The corresponding uncertainty set is $\mathcal{U}_1$.
- **aRUB-L_1**. The model is trained on clean data with the approximate robust upper bound loss function of equation (56) where $q = 1$. The corresponding uncertainty set is $\mathcal{U}_\infty$.
- **FGSM**. An adversarially trained model with FGSM attacks as in equation (43) in the uncertainty set $\mathcal{U}_\infty$.
- **PGD**. An adversarially trained model with PGD attacks as in equation (45) in the uncertainty set $\mathcal{U}_\infty$. Here, a step size of $\alpha = \varepsilon/5$ and an iteration number of $I = 10$ was used.

4.4 Attacks

The models' robustness with respect to adversarial attacks was determined using attacks generated using the test data. The models had not been trained on any samples, original or augmented, from the test data. The models were tested with two attacks:

- **FGSM**. The attack is constructed with equations (43) and (44).
- **PGD**. The attack is constructed with equation (46) and parameters $\alpha = \varepsilon/5$ and $I = 10$.

For both attacks, the set $\mathcal{U}_\infty$ is used as the underlying domain, i.e., the domain for the inner maximization is the L_∞-sphere. The L_∞-sphere's volume is larger than L_1-sphere's and L_2-sphere's volumes, thus, the L_∞-sphere is a larger search space for the inner maximization when constructing the adversarial attacks, which should result in stronger attacks, at least for the PGD.

4.5 Hyperparameters and Training

All models were trained with the same hyperparameters. The stochastic gradient descent optimizer was used with a momentum of 0.9. The models were trained for 10 epochs with a constant learning rate of $\eta = 0.01$. A batch size of 128 was used.

No fine-tuning of hyperparameters was performed. Instead, reasonable parameter values, which worked on all models, were used for a fair comparison between the models. The training performance metrics are shown in appendix A.

For the convolutional layers, pre-trained weights were used. The pre-trained weights were obtained directly from PyTorch and they were optimized for the ImageNet dataset. The weights for the last four fully connected layers were initialized with the Xavier initialization method (Glorot and Bengio, 2010), i.e., the weights are sampled uniformly from the interval $[-\frac{1}{\sqrt{m}}, \frac{1}{\sqrt{m}}]$, where m is the input dimension.

To help convergence and prevent divergence due to gradient exploding (Goodfellow et al., 2016), gradient clipping was used, where the magnitude of the gradient is constrained by the L_2-norm so that $||\nabla_\theta \mathcal{L}(h(x,\theta),y)||_2 \leq 1$. This method was found to help the aRUB models to converge. In addition, training the aRUB seemed to be more cumbersome than the other models. The aRUB-L_1 was not able to converge directly with the pre-trained ResNet-18 and a Xavier initialization of weights for a perturbation size of $\varepsilon = 0.001$. The aRUB required an initial training of three epochs with one magnitude smaller perturbation size, i.e., $\varepsilon = 0.0001$, where after the model could be trained with the desired perturbation sizw of $\varepsilon = 0.001$ for the rest of the seven epochs. The same kind of difficulties was found for the PGD model for some dataset splits, and, thus, the same method of "hot starting" was done for both aRUB models and the FGSM and PGD models.

4.6 Implementation

The models were implemented in PyTorch v.1.8.1 (Paszke et al., 2019) and all the model building, scripting, data-handling, pre-processing and post-processing were done in Python 3.8 (Van Rossum and Drake, 2009). The models were trained on an Ubuntu desktop using the graphical processing unit (GPU). The GPU was a NVIDIA Quadro P5000 with CUDA version 11.4 (NVIDIA, 2020). The GPUs had 16Gb of memory, which was enough to train all the models with a batch size of 128. The training times of the models are presented in chapter 5.

5 Experimental Results

This chapter shows the results for the three datasets. For each dataset and each model, 10 different splits for the test and train datasets were performed. By doing this, mean values and standard deviations could be calculated, which provided additional statistical confidence to the results. The sample means and biased-corrected standard deviations were used as the main performance indicators. In addition, training times are presented for comparison of the models.

Standard metrics of accuracy, sensitivity and specificity were used to compare the robustness of the models. Given the true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN), the standard *accuracy* is defined as

$$\text{accuracy} = \frac{TP + TN}{TP + FP + TN + FN}, \tag{58}$$

sensitivity or the *true positive rate* as

$$\text{sensitivity} = \frac{TP}{TP + FN}, \tag{59}$$

and *specificity* or *true negative rate* as

$$\text{specificity} = \frac{TN}{TN + FP}. \tag{60}$$

Even though the accuracy, sensitivity and specificity are not guaranteed to be the minimum for all the possible perturbations inside the uncertainty domain $\mathcal{U}_\infty$, it provides a practical insight in the evaluated robustness.

5.1 Diabetic Retinopathy Dataset

The first dataset is the diabetic retinopathy dataset. The training times of the five different models are shown in Table 2 for all 10 splits of the dataset. The average and the standard deviation are shown in the last rows. Clearly, the fastest training method is the nominal training with no robustification, followed up by the FGSM, which is almost as fast. This is to be expected, as FGSM is an one-step method and does not require significant calculation resources. The PGD model required the longest training time and it took approximately 40% and 50% longer to train than aRUB models. The reason for this is that PGD is an iterative method and requires several computational steps and resources, while aRUB requires only a norm computation of the neural network's gradient.

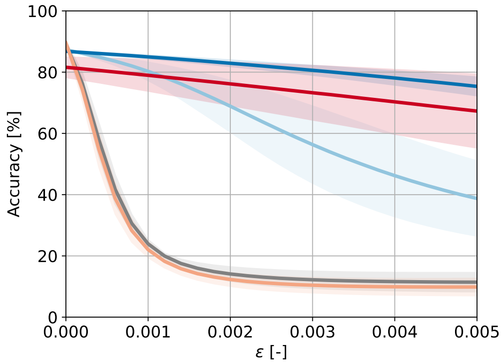
The mean performance metrics of accuracy, sensitivity and specificity are shown in Figure 8 as a function of the perturbation size ε, with the solid line and the shaded area represents one standard deviation from the mean, respectively. The performance metrics for the FGSM attacks are shown on the left side and the PGD attacks on the right side. An attack size of $\varepsilon = 0$ corresponds to clean data.

Table 2: Training times in seconds for the diabetic retinopathy dataset.

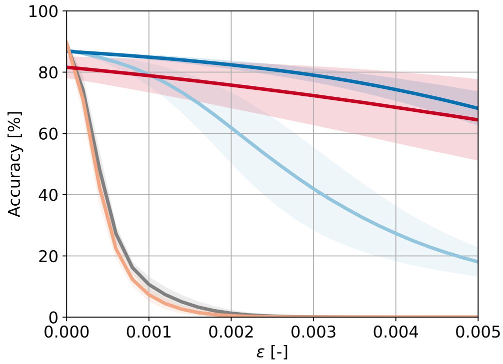
Dataset	Nominal	FGSM	PGD	aRUB-L_∞	aRUB-L_1
1	11288	15356	52447	34617	38041
2	11756	15649	52252	34300	37783
3	11239	15277	52285	34388	37958
4	11248	15384	52606	34725	37959
5	11219	15438	52306	34468	36456
6	11274	15489	52451	34814	38205
7	11294	15802	52588	34445	37914
8	11273	15455	52494	34439	38195
9	11303	15410	52482	34726	37990
10	11727	15402	52360	34441	38083
Mean	11362	15466	52427	34536	37858
Standard Deviation	191	145	117	163	483

Interestingly, the aRUB-L_∞ model performs as the nominal model or slightly worse on all metrics for both attacks. As expected, the FGSM model performs worse than the PGD and aRUB-L_1, but better than the nominal model. The best performer seems to be the PGD model, which is slightly above the aRUB-L_1 model. Both PGD and aRUB-L_1 models provide decent robustness. For example, with the perturbation size $\varepsilon = 0.005$, the accuracy remains above 60% for both PGD and aRUB-L_1, which is a clear improvement when compared to the nominal model. Also, the attack $\varepsilon = 0.005$ is five times larger than the perturbation the models were trained for. A perturbation of $\varepsilon = 0.001$ corresponds to approximately a 0.25/255 pixel value change while $\varepsilon = 0.005$ corresponds to approximately a 1.3/255 pixel value change.

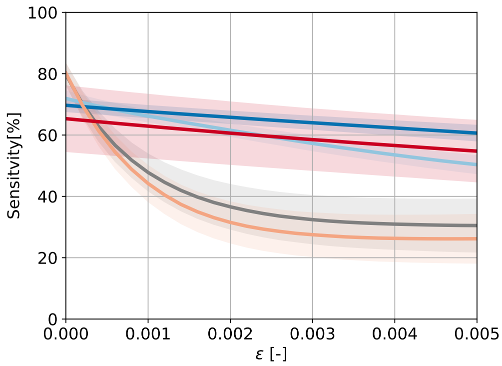
One standard deviation of the mean is shown as a shaded area in Figure 8. For accuracy, the standard deviation is relatively small but increases with perturbation size for the aRUB-L_1 model to around 10%. Also, the FGSM has a high standard deviation in accuracy when compared to the other models. For specificity, the standard deviation is larger than for sensitivity and accuracy for the aRUB-L_1 model, which indicates that there is a variation between different dataset splits on the performance. The explanation for this is that the different splits have most likely learned different features and hence the performance deviations between the splits. PGD on the other hand seems to have a quite small standard deviation compared to aRUB-L_1, making it more consistent with the predictions. For aRUB-L_∞ the standard deviation seems to be quite small compared to the other models on all metrics, but the overall performance against different sized perturbations is poor.



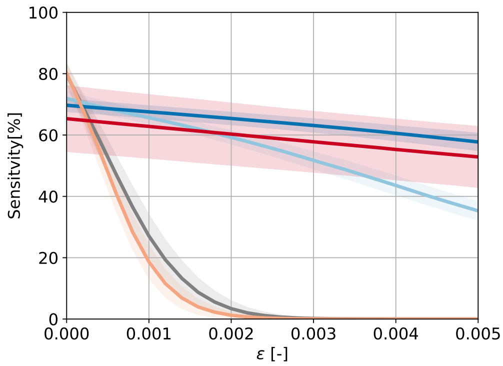
(a) Accuracy on FGSM attacks.



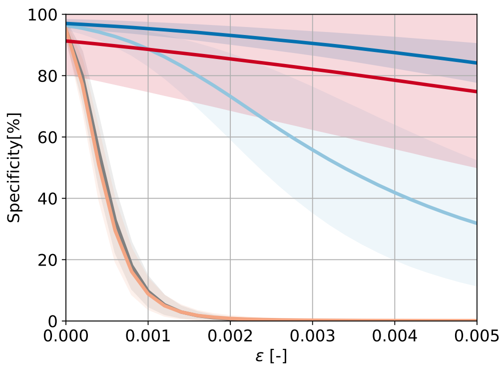
(b) Accuracy on PGD attacks.



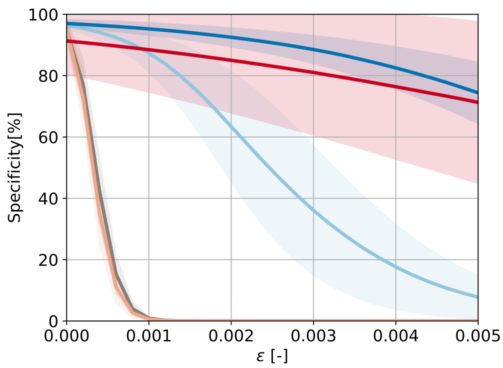
(c) Sensitivity on FGSM attacks.



(d) Sensitivity on PGD attacks.



(e) Specificity on FGSM attacks.



(f) Specificity on PGD attacks.

Figure 8: Accuracy, sensitivity and specificity of the diabetic retinopathy dataset for FGSM and PGD attacks within $\mathcal{U}_\infty$ for different perturbation sizes ε. The attacks are performed on the test set. The solid line corresponds to the mean of the 10 dataset splits and the shaded area corresponds to one standard deviation of the mean.

5.2 Nevus Dataset

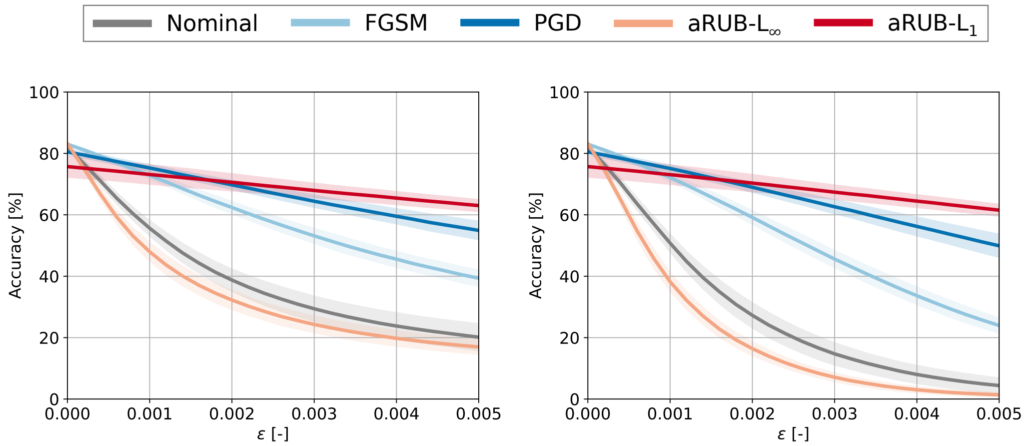
The second dataset is the nevus dataset. The training times of the five different models are shown in Table 3 for all 10 splits of the dataset. Again, the fastest training method is the nominal training with no robustification. The PGD model took approximately 50% longer to train than aRUB models and almost 400% longer than the nominal model. The standard deviations of the training times for each model seem to be small.

Table 3: Training times in seconds for the nevus dataset.

Dataset	Nominal	FGSM	PGD	aRUB-L_∞	aRUB-L_1
1	2786	3651	12567	8312	8174
2	2698	3648	12462	8225	8151
3	2697	3653	12447	8224	8144
4	2728	3644	12460	8229	8153
5	2701	3645	12491	8227	8148
6	2706	3640	12495	8232	8151
7	2706	3648	12587	8270	8192
8	2706	3647	12447	8243	8176
9	2750	3645	12437	8254	8175
10	2703	3654	12439	8242	8169
Average	2718	3647	12483	8245	8163
Standard Deviation	27	4	51	26	15

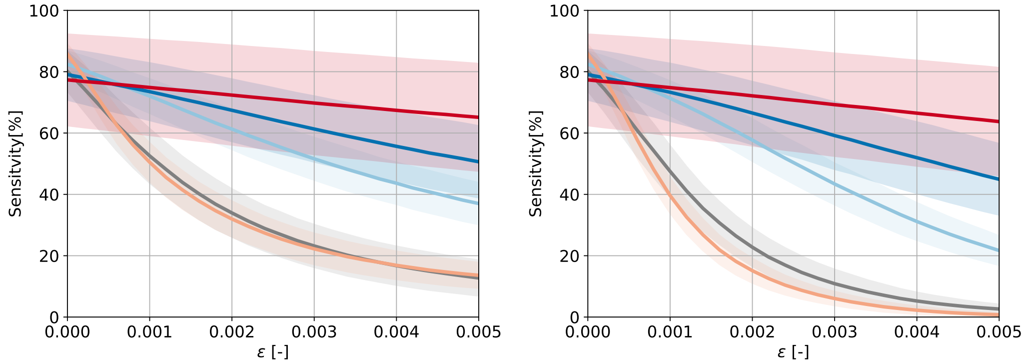
Figure 9 shows the mean performance metrics and the standard deviations for the nevus dataset for different sizes of attacks. Similarly, as for the diabetic retinopathy dataset, the aRUB-L_∞ model performs the worst on all performance metrics. In difference to the first dataset, here, the aRUB-L_1 performs best and even better than PGD for large attacks. The aRUB-L_1 retains an accuracy, sensitivity and specificity of over 60% for perturbations of size $\varepsilon = 0.005$, while the PGD drops under 50%. In difference to the previous diabetic retinopathy dataset, the difference between FGSM and PGD is not as large. The FGSM model performs in a similar manner but the sensitivity and specificity for PGD attacks do not drop as significantly when compared to the diabetic retinopathy dataset.

The standard deviation for sensitivity and specificity of aRUB-L_1 is around 15% and remains almost constant for all perturbation sizes. The nominal, FGSM and PGD models have about the same size of the standard deviation of around 8% for the same metrics, while the aRUB-L_∞ has clearly the lowest standard deviation, which is caused by the poor overall performance. The standard deviation for accuracy on all models appears relatively small.



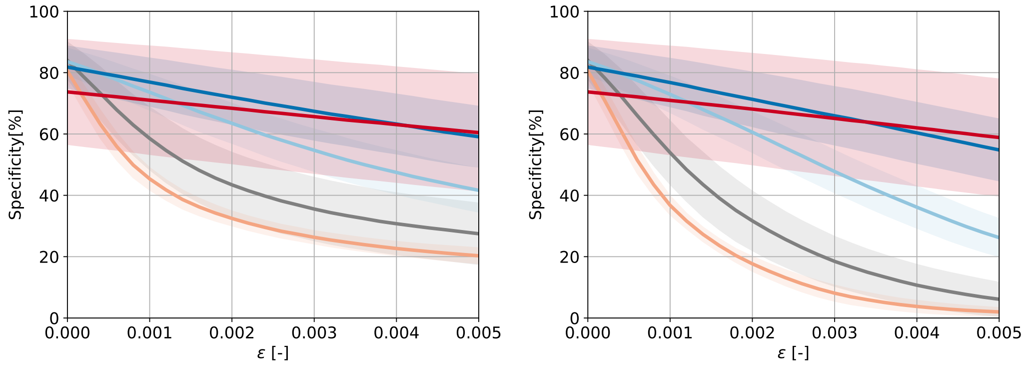
(a) Accuracy on FGSM attacks.

(b) Accuracy on PGD attacks.



(c) Sensitivity on FGSM attacks.

(d) Sensitivity on PGD attacks.



(e) Specificity on FGSM attacks.

(f) Specificity on PGD attacks.

Figure 9: Accuracy, sensitivity and specificity of the nevus dataset for FGSM and PGD attacks within $\mathcal{U}_\infty$ for different perturbation sizes ε. The attacks are performed on the test set. The solid line corresponds to the mean of the 10 dataset splits and the shaded area corresponds to one standard deviation of the mean.

5.3 Chest X-ray Dataset

The last dataset is the chest X-ray dataset. First, in Table 4 the training times are shown for the models for the different dataset splits. Yet again, the nominal model is the fastest and PGD is the slowest to train. In difference to the previous cases, the PGD model is between 55% and 70% slower to train than the aRUB models.

Table 4: Training times in seconds for the chest X-ray dataset.

Dataset	Nominal	FGSM	PGD	aRUB-L_∞	aRUB-L_1
1	1741	2584	8782	5810	5268
2	1728	2562	9098	5772	5225
3	1724	2549	9123	5771	5227
4	1736	2547	9121	5775	5228
5	1721	2548	9135	5771	5223
6	1728	2547	9132	5771	5224
7	1720	2550	9137	5776	5225
8	1723	2546	9138	5779	5227
9	1725	2548	8825	5809	5234
10	1725	2546	8675	6107	5243
Average	1727	2552	9016	5814	5232
Standard Deviation	6	11	171	99	13

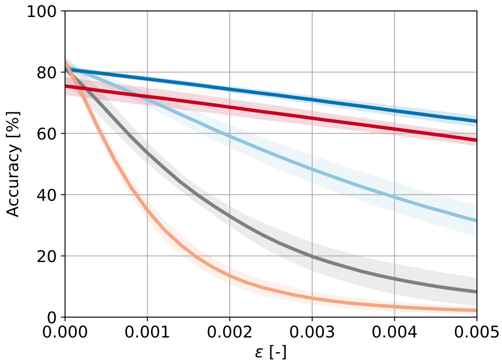
In Figure 10 the mean and standard deviation performance metrics for both attacks on the chest X-ray dataset are shown. The results are similar as for the diabetic retinopathy dataset, i.e., the aRUB-L_∞ performed worst and the PGD performed the best. Similarly, also the aRUB-L_1 performs almost as well as the PGD.

The standard deviation for the accuracy of all models remains quite small while for the sensitivity and specificity the standard deviation increases. Yet again, the aRUB-L_1 model has the highest standard deviation of approximately 15%.

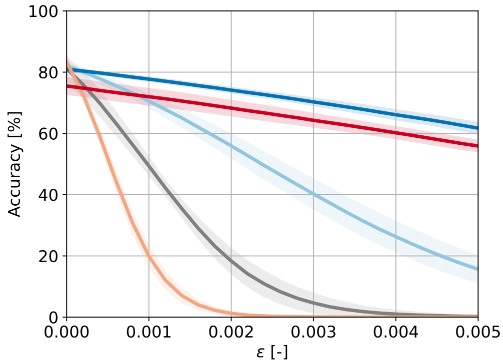
5.4 Discussion

A common observation from the three datasets is that the aRUB-L_1 has the lowest nominal performance in accuracy, sensitivity and specificity. From the training metrics in appendix A, in figures A1e, A2e and A3e, it can be seen that the mean training loss is still decreasing while the training accuracy is not increasing and thereby not overfitting on the data. Hence, the aRUB-L_1 could have benefitted from more epochs and iterations of the optimizer. All other models seemed to have an equal or minimal difference in the nominal performance when compared to each other. Nonetheless, the aRUB-L_1 performs well in robustness as the accuracy, sensitivity and specificity remain on high levels for larger perturbation sizes..

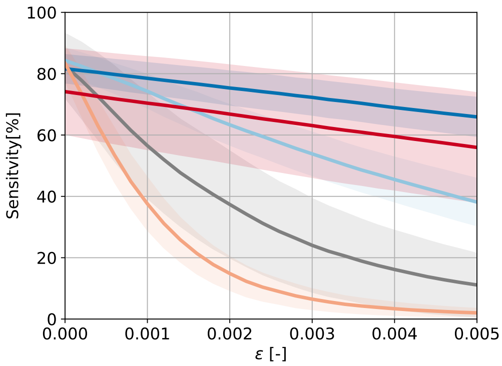
Another interesting common observation from all the datasets is that aRUB-L_1



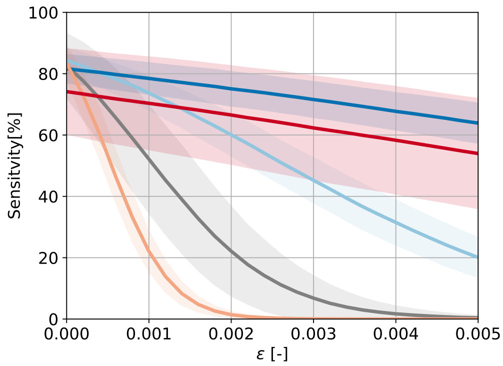
(a) Accuracy on FGSM attacks.



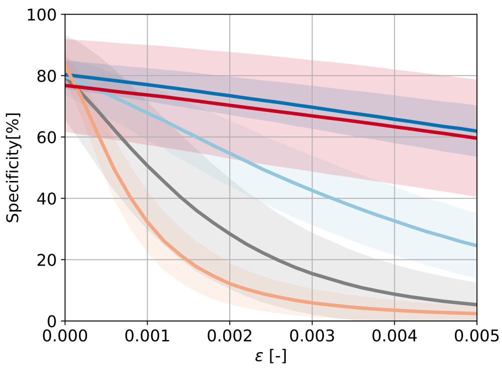
(b) Accuracy on PGD attacks.



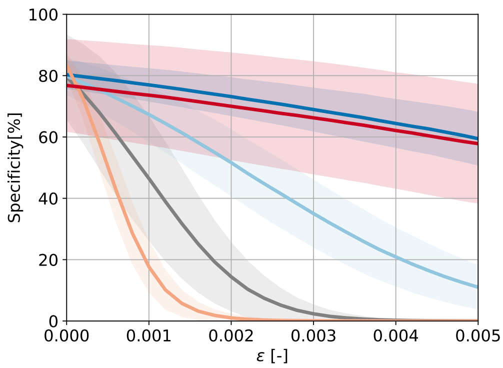
(c) Sensitivity on FGSM attacks.



(d) Sensitivity on PGD attacks.



(e) Specificity on FGSM attacks.



(f) Specificity on PGD attacks.

Figure 10: Accuracy, sensitivity and specificity of the diabetic chest X-ray for FGSM and PGD attacks within $\mathcal{U}_\infty$ for different perturbation sizes ε. The attacks are performed on the test set. The solid line corresponds to the mean of the 10 dataset splits and the shaded area corresponds to one standard deviation of the mean.

has the highest standard deviation, which is clearly visible for sensitivity and specificity. Again, based on the training metrics in appendix A, in figures A1e, A2e and A3e, there is oscillations in the metrics between different sets. Here again, the aRUB-L_1 could have benefited from more epochs and maybe even decreasing the learning rate to reduce the oscillations between each epoch later in the optimization.

The aRUB-L_∞ showed the lowest robust performance on each dataset, indicating that the method is not functioning. The most probable explanation for this is that since the uncertainty set of the attack was the $\mathcal{U}_\infty$, the proper dual norm in the aRUB formulation is then the L_1-norm, not the L_∞-norm. In addition, the norm term in equation (56) acts as a regularizer to the loss. Regularizers are commonly used in training neural networks to avoid overfitting and here the model probably benefits from a larger value of the norm (regularizer) for adversarial attacks. The L_1-norm is always larger than the L_∞-norm for multidimensional vectors. This can be proven by the following statement

$$||x||_\infty = \max_i \{x_i\} \leq \sum_{i=1}^{m} |x_i| = ||x||_1. \tag{61}$$

The L_1-norm acts as a larger regularizer (by value) than the L_∞-norm in this case, which probably also increases the performance against adversarial attacks.

Comparing the FGSM and the PGD models over all the sets, as expected, the FGSM is a weaker form of attack as the FGSM model's performance drops faster than the PGD model's performance for both attacks. Nonetheless, there is no clear difference between the type of attack on the PGD model, i.e., a FGSM attack on the PGD model seems to cause similar results that a PGD attack causes on the PGD model.

Another interesting observation is that data augmentation did not seem to affect the robustness of the models. For the diabetic retinopathy dataset, the positive class samples were augmented five times while only the negative class samples were augmented a single time. This was done to even the amount of positive and negative class samples in the dataset. Also, the nevus dataset's samples were augmented, but both classes were augmented an equal amount. The chest X-ray dataset was not augmented and only clean data was used. The models perform similarly for all datasets and thus no clear effect of augmentation on the performance could be observed.

6 Conclusion

Five different models (nominal, FGSM, PGD, aRUB-L_1 and aRUB-L_∞) were tested on three datasets (diabetic retinopathy, nevus and chest X-ray) for two different adversarial attacks (FGSM and PGD). The results showed good robust performance for both the PGD and the aRUB-L_1 model, with a slightly better performance of the PGD. Both adversarial training and the newly developed aRUB method seem to produce promising results in providing robustness to the model. It is noteworthy that even though the models were only trained for a perturbation size of $\varepsilon = 0.001$, the aRUB-L_1 and PGD were still providing reasonable accuracy at attacks performed with five times the magnitude. Providing robustness even for smaller perturbations is beneficial as identifying adversarial examples with smaller perturbations is harder than for larger perturbations. Therefore providing robustness in smaller perturbations together with an adversarial sample detection algorithm might be enough to secure and provide robustness for the AI algorithms within the medical field.

Even though the PGD model showed better performance than the aRUB-L_1 model on two datasets, the aRUB-L_1 model had a faster training time for all cases. The PGD model took about 50% longer to train. Here, a reasonable number of 10 iterations were performed for the PGD, while other sources (Ma et al., 2021; Xu et al., 2020; Finlayson et al., 2019) usually use a higher number of iterations. The iteration number affects directly the training time for the PGD model and thus the difference here in the training times might be smaller than one would see in practice. Another practical advantage for the aRUB-L_1 model is that it is straightforward to implement as it requires only an implementation of a custom loss function, which can be used in the normal conventional neural network training routines. In the future, when more practical experiments have been conducted for the aRUB, it is to be expected to be implemented as a standard loss function in different deep learning frameworks.

Although the aRUB-L_1 model performed well and was relatively easy to include in the normal training routine of neural networks, it was found difficult to get the model converging and learning. The aRUB-L_1 required a "hot start", meaning that it needed to be trained first with smaller perturbations for three epochs before the desired perturbation could be used. The aRUB-L_1 seemed to get stuck as the training loss was not decreasing and remained constant with small oscillations. Similar phenomenon was also found for the PGD model for some dataset splits. This would indicate that the aRUB-L_1 needs a good initial starting point for training as well as the PGD.

As a conclusion, the aRUB provided competitive results and performance compared to adversarial training, when the correct dual norm to the uncertainty set was used. The aRUB was competitive both in robustness and in training times. In training times the aRUB is faster than adversarial training with strong attacks, such as PGD, making aRUB an attractive alternative to adversarial training.

For future research, the robust optimization approach of providing a computationally tractable robust counterpart, such as the robust upper bound, seems to be a promising lead in robustifying neural networks in practice. Bertsimas et al. (2021a) provided a second exact robust upper bound, the RUB, which was not implemented or further studied in this research. The exact robust upper bound does not scale well, as the number of computations needed increases with the number of dimensions and, hence, it is difficult to implement in practice for real-world problems with large input dimensions. However, implementing the exact robust upper bound together with the model distillation concept of retraining the soft labels with the exact robust upper bound for the last fully connected layers, could provide robustness and prevent the problem with large input dimensions. The benefit of this approach compared to the original model distillation idea is that the proposed method does not require any adversarial training of the last layers, decreasing the training times and hopefully it would provide more robust neural networks. A preliminary attempt was made in this thesis to implement the exact robust upper bound in the last fully connected layers. The weights of the last fully connected layers were trained together with the convolutional layer's parameters, which differs from the model distillation training proposed by Papernot et al. (2016a). In model distillation, first the convolutional neural network is trained with clean data and thereafter only the final layers are trained with adversarial examples. Also, any needed detailing such as the special softmax layer with the regularizing parameters was ignored in this attempt. The results showed poor performance, but the method was not further investigated and, thus, left out of the scope of this thesis. However, with the proper model distillation framework, this idea of robustifying the last fully connected layers with the exact robust upper bound could be a lead for further future development and further development of the robust optimization approach.

Another possible future research idea of the robust optimization approach could be to implement the aRUB onto different loss functions. Here, the cross entropy was used as the loss function. A similar idea could be implemented in other loss functions such as the mean square error loss. The basic idea of the aRUB was to linearize the neural network h instead of the loss function $\mathcal{L}$ with respect to the perturbation δ. Thus, the aRUB is not limited only to the cross entropy loss.

The robustness of AI algorithms, especially neural networks, in the medical field is still an unsolved practical problem. The problem has only just been identified and future development and more research are probably to be expected as robustification methods are being developed as well as the whole AI in the medical field is still developing. Interestingly, it was found from previous research of adversarial attacks within the medical field (Finlayson et al., 2019; Ma et al., 2021; Xu et al., 2020; Bortsova et al., 2021), that the viewpoint for the threat of adversarial attacks comes from an economical perspective rather than patient safety perspective. Hence, the driving factor for the development of research seems to be economical benefit and fraud prevention. Nonetheless, making medical neural networks more robust increases patient safety, no matter what the driving factor may be. For medical devices, FDA has still not included risk management requirements for adversarial attacks. The same lack of regulation applies to the EU as well. This will most likely change in the future, the latest after the first fraudulent cases utilizing adversarial attacks are exposed and detected.

References

Apostolidis, K.D. and Papakostas, G.A., 2021. *A survey on adversarial deep learning robustness in medical image analysis. Electronics*, 10(17), p.2132.

Ben-Tal, A., El Ghaoui, L. and Nemirovski, A., 2009. *Robust optimization.* Princeton university press.

Bernhard, P. and Rapaport, A., 1995. *On a theorem of Danskin with an application to a theorem of Von Neumann-Sion.* Nonlinear Analysis: Theory, Methods Applications, 24(8), pp.1163-1181.

Bertsimas, D., Boussioux, L., Carballo, K.V., Li, M.L., Paskov, A. and Paskov, I., 2021. *Holistic Deep Learning.* arXiv preprint arXiv:2110.15829.

Bertsimas, D., Boix, X., Carballo, K.V. and Hertog, D.D., 2021. *A robust optimization approach to deep learning.* arXiv preprint arXiv:2112.09279.

Bertsekas, D. P., 1999. *Nonlinear Programming.* Ahtena Scientific.

Bortsova, G., González-Gonzalo, C., Wetstein, S.C., Dubost, F., Katramados, I., Hogeweg, L., Liefers, B., van Ginneken, B., Pluim, J.P., Veta, M. and Sánchez, C.I., 2021. *Adversarial attack vulnerability of medical image analysis systems: Unexplored factors.* Medical Image Analysis, 73, p.102141.

Bourke S.J., 2015. *Respiratory Medicine.* John Wiley Sons.

Boyd S. and Vandenberghe L., 2004. *Convex optimization.* Cambridge university press.

Carlini, N. and Wagner, D., 2017, May. *Towards evaluating the robustness of neural networks.* In 2017 ieee symposium on security and privacy (sp) (pp. 39-57). Ieee.

Danskin, J. M., 1967. *The Theory of Max-Min and Its Application to Weapons Allocation Problems.* Springer-Verlag Berlin.

Davenport, T. and Kalakota, R., 2019. *The potential for artificial intelligence in healthcare.* Future healthcare journal, 6(2), p.94.

Deng, J., Dong, W., Socher, R., Li, L.J., Li, K. and Fei-Fei, L., 2009, June. *Imagenet: A large-scale hierarchical image database.* In 2009 IEEE conference on computer vision and pattern recognition (pp. 248-255). Ieee.

Dvijotham, K., Stanforth, R., Gowal, S., Mann, T.A. and Kohli, P., 2018, March. *A Dual Approach to Scalable Verification of Deep Networks.* In UAI (Vol. 1, No. 2, p. 3).

Feinman, R., Curtin, R.R., Shintre, S. and Gardner, A.B., 2017. *Detecting adversarial samples from artifacts.* arXiv preprint arXiv:1703.00410.

Finlayson, S.G., Bowers, J.D., Ito, J., Zittrain, J.L., Beam, A.L. and Kohane, I.S., 2019. *Adversarial attacks on medical machine learning*. Science, 363(6433), pp.1287-1289.

Gee, J., Button, M., 2015. *The financial cost of healthcare fraud 2015: What data from around the world shows*. Available at: https://www.researchgate.net/publication/299378586_The_Financial_Cost_of_Healthcare_Fraud

Gong, Z., Wang, W. and Ku, W.S., 2017. *Adversarial and clean data are not twins*. arXiv preprint arXiv:1704.04960.

Goodfellow, I, Shlens, J. and Szegedy, C., 2014. *Explaining and harnessing adversarial examples*. arXiv preprint arXiv:1412.6572.

Goodfellow, I., Bengio, Y. and Courville, A., 2016. *Deep learning*. MIT press.

Glorot, X. and Bengio, Y., 2010, March. *Understanding the difficulty of training deep feedforward neural networks*. In Proceedings of the thirteenth international conference on artificial intelligence and statistics (pp. 249-256). JMLR Workshop and Conference Proceedings.

Hall J.C. and Hall B.J., 2017. *Sauer's Manual of Skin Diseases*. Wolters Kluwer.

He, K., Zhang, X., Ren, S. and Sun, J., 2016. *Deep residual learning for image recognition*. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

Hirano, H., Minagi, A. and Takemoto, K., 2021. *Universal adversarial attacks on deep neural networks for medical image classification*. BMC medical imaging, 21(1), pp.1-13.

Minagi, A., Hirano, H. and Takemoto, K., 2022. *Natural Images Allow Universal Adversarial Attacks on Medical Image Classification Using Deep Neural Networks with Transfer Learning*. Journal of Imaging, 8(2), p.38.

Ilyas, A., Santurkar, S., Tsipras, D., Engstrom, L., Tran, B. and Madry, A., 2019. Adversarial examples are not bugs, they are features. Advances in neural information processing systems, 32.

Irvin, J., Rajpurkar, P., Ko, M., Yu, Y., Ciurea-Ilcus, S., Chute, C., Marklund, H., Haghgoo, B., Ball, R., Shpanskaya, K. and Seekins, J., 2019, July. *Chexpert: A large chest radiograph dataset with uncertainty labels and expert comparison*. In Proceedings of the AAAI conference on artificial intelligence (Vol. 33, No. 01, pp. 590-597).

ISIC, 2019. *The international skin imaging collaboration*. Available at: https://challenge.isic-archive.com/landing/2019/.

Jain, A., Nundy, S. and Abbasi, K., 2014. *Corruption: medicine's dirty open secret*. BMJ, 348.

Kaggle, 2015. *Kaggle diabetic retinopathy detection challenge.* Available at: https://www.kaggle.com/c/diabetic-retinopathydetection.

Kaggle, 2020. *Br35H :: Brain Tumor Detection 2020.* Available at: https://www.kaggle.com/datasets/ahmedhamada0/brain-tumor-detection?resource=download.

Kaschke, M., Donnerhacke, K. and Rill, M.S., 2014. *Optical Devices in Ophthalmology and Optometry : Technology, Design Principles and Clinical Applications.* John Wiley Sons.

Katz, G., Barrett, C., Dill, D.L., Julian, K. and Kochenderfer, M.J., 2017, July. *Reluplex: An efficient SMT solver for verifying deep neural networks.* In International conference on computer aided verification (pp. 97-117). Springer, Cham.

Kesselheim, A.S. and Brennan, T.A., 2005. *Overbilling vs. downcoding—the battle between physicians and insurers.* New England Journal of Medicine, 352(9), pp.855-857.

Krizhevsky, A., 2009. *Learning multiple layers of features from tiny images.*Available at: https://www.cs.toronto.edu/~kriz/cifar.html.

Kreyszig, E., 1989, *Introductory Functional Analysis with Applications.* John Wiley Sons.

Kurakin, A., Goodfellow, I.J. and Bengio, S., 2018. *Adversarial examples in the physical world.* In Artificial intelligence safety and security (pp. 99-112). Chapman and Hall/CRC.

LeCun, Y., 1998, *The mnist database of handwritten digits.* Available at: http://yann.lecun.com/exdb/mnist/.

Lecuyer, M., Atlidakis, V., Geambasu, R., Hsu, D. and Jana, S., 2019, May. *Certified robustness to adversarial examples with differential privacy.* In 2019 IEEE Symposium on Security and Privacy (SP) (pp. 656-672). IEEE.

Ma, X., Niu, Y., Gu, L., Wang, Y., Zhao, Y., Bailey, J. and Lu, F., 2021. Understanding adversarial attacks on deep learning based medical image analysis systems. Pattern Recognition, 110, p.107332.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D. and Vladu, A., 2017. *Towards deep learning models resistant to adversarial attacks.* arXiv preprint arXiv:1706.06083.

Metzen, J.H., Genewein, T., Fischer, V. and Bischoff, B., 2017. *On detecting adversarial perturbations.* arXiv preprint arXiv:1702.04267.

Muehlematter, U.J., Daniore, P. and Vokinger, K.N., 2021. *Approval of artificial intelligence and machine learning-based medical devices in the USA and Europe (2015–20): a comparative analysis.* The Lancet Digital Health, 3(3), pp.e195-e203.

Moosavi-Dezfooli, S.M., Fawzi, A., Fawzi, O. and Frossard, P., 2017. *Universal adversarial perturbations*. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1765-1773).

NVIDIA, Vingelmann P. and Fitzek F.H.P., 2020. *CUDA*. Release: 11.4, Available at: https://developer.nvidia.com/cuda-toolkit.

Pan, S.J. and Yang, Q., 2009. *A survey on transfer learning*. IEEE Transactions on knowledge and data engineering, 22(10), pp.1345-1359.

Papernot, N., McDaniel, P., Wu, X., Jha, S. and Swami, A., 2016, May. *Distillation as a defense to adversarial perturbations against deep neural networks*. In 2016 IEEE symposium on security and privacy (SP) (pp. 582-597). IEEE.

Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z.B. and Swami, A., 2016, March. *The limitations of deep learning in adversarial settings*. In 2016 IEEE European symposium on security and privacy (EuroSP) (pp. 372-387). IEEE.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L. and Desmaison, A., 2019. *Pytorch: An imperative style, high-performance deep learning library*. Advances in neural information processing systems, 32.

Raghunathan, A., Steinhardt, J. and Liang, P., 2018. *Certified defenses against adversarial examples*. arXiv preprint arXiv:1801.09344.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. and Fergus, R., 2013. *Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199*.

U.S. Food and Drug Administration, 2012. *Computer-Assisted Detection Devices Applied to Radiology Images and Radiology Device Data-Premarket Notification [510 (k)]*. Submissions-Guidance for Industry and Food and Drug Administration Staff. Available at: https://www.fda.gov/regulatory-information/search-fda-guidance-documents/computer-assisted-detection-devices-applied-radiology-images-and-radiology-device-data-premarket.

U.S. Food and Drug Administration, 2014. *Cybersecurity in Medical Devices: Quality System Considerations and Content of Premarket Submissions*. Submissions-Guidance for Industry and Food and Drug Administration Staff. Available at: https://www.fda.gov/regulatory-information/search-fda-guidance-documents/content-premarket-submissions-management-cybersecurity-medical-devices.

U.S. Food and Drug Administration, 2018. *FDA permits marketing of artificial intelligence based device to detect certain diabetes related eye problems*. Press Announcements. Available at: https://www.fda.gov/news-events/press-announcements/fda-permits-marketing-artificial-intelligence-based-device-detect-certain-diabetes-related-eye.

U.S. Food and Drug Administration, 2018. *Software as a Medical Device (SaMD).* Submissions-Guidance for Industry and Food and Drug Administration Staff. Available at: https://www.fda.gov/medical-devices/digital-health-center-excellence/software-medical-device-samd.

U.S. Food and Drug Administration, 2022. *Cybersecurity in Medical Devices: Quality System Considerations and Content of Premarket Submissions.* Submissions-Guidance for Industry and Food and Drug Administration Staff. Available at: https://www.fda.gov/regulatory-information/search-fda-guidance-documents/cybersecurity-medical-devices-quality-system-considerations-and-content-premarket-submissions

Van Rossum, G. Drake, F.L., 2009. *Python 3 Reference Manual.* Scotts Valley, CA: CreateSpace.

Wong, E. and Kolter, Z., 2018, July. *Provable defenses against adversarial examples via the convex outer adversarial polytope.* In International Conference on Machine Learning (pp. 5286-5295). PMLR.

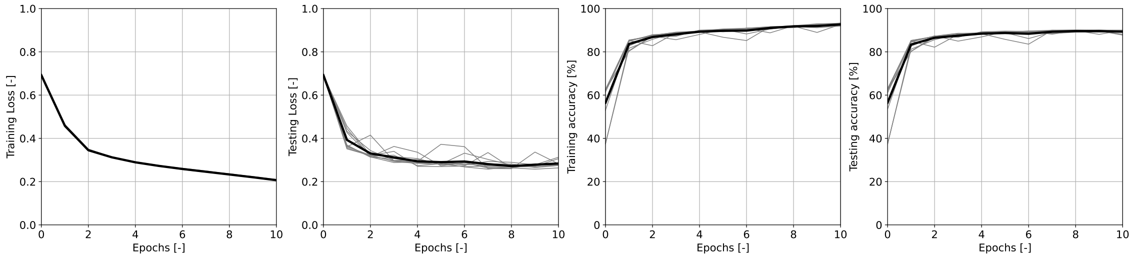
Wynia, M.K., Cummins, D.S., VanGeest, J.B. and Wilson, I.B., 2000. *Physician manipulation of reimbursement rules for patients: between a rock and a hard place.* Jama, 283(14), pp.1858-1865.

Xu, M., Zhang, T., Li, Z., Liu, M. and Zhang, D., 2021. Towards evaluating the robustness of deep diagnostic models by adversarial attack. Medical Image Analysis, 69, p.101977.

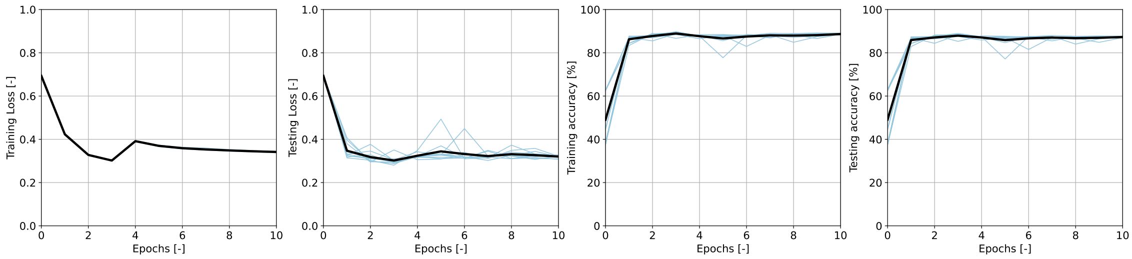
A Training Metrics

The training metrics which are monitored during the training of the neural network are the loss and accuracy evaluated for the training and test set respectively. The evaluation is done after each epoch. Figure A1, A2 and A3 show the loss and accuracy on the test and train set for each dataset split (10 splits in total). The thin-colored lines are the metrics for each split of the test and train set and the solid black line is the mean over the 10 split sets.

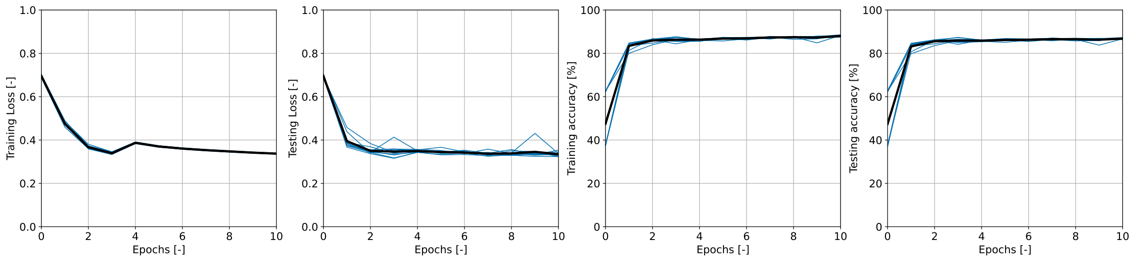
As can be seen from the testing performance in figures A1, A2 and A3, the models seem to converge. The figures show that the obtained training provides reasonable accuracy that could be expected for the current common neural network architecture of ResNet-18. No hyperparameter tuning for each case was done since it is computationally expensive and time-consuming and does not necessarily give additional value. Also, the fixed set of the chosen hyperparameters provides a fair comparison between the models.



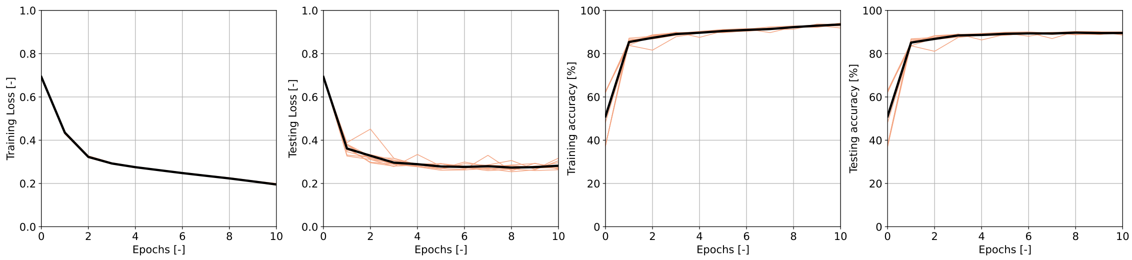
(a) Nominal



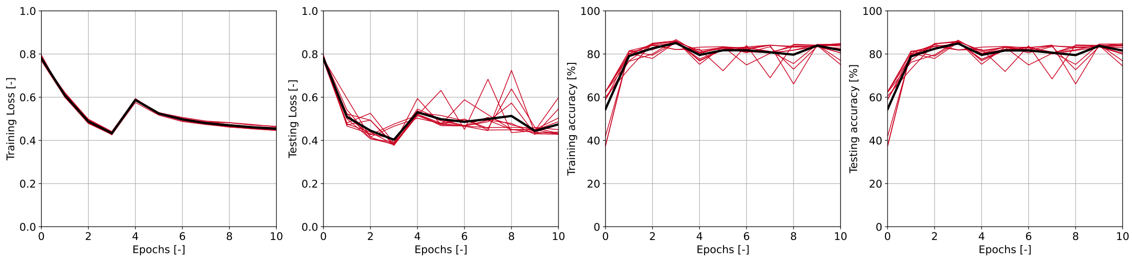
(b) FGSM



(c) PGD

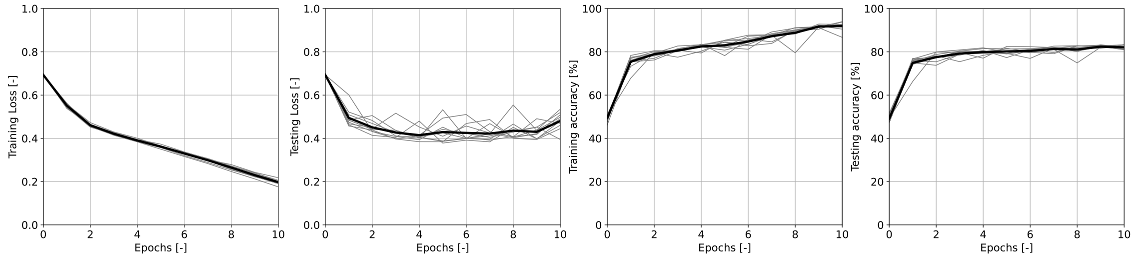


(d) aRUB-L_∞

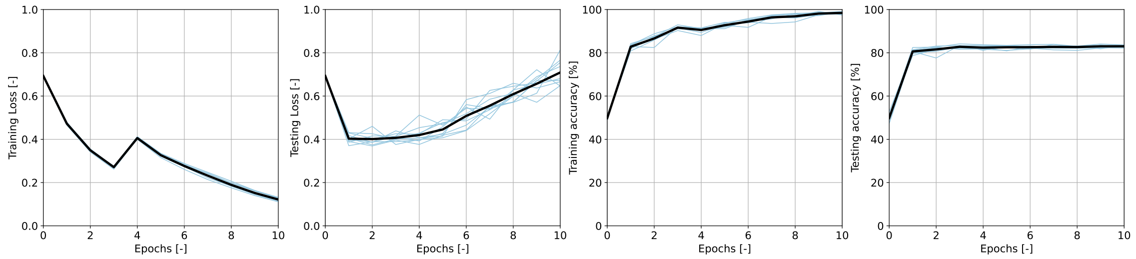


(e) aRUB-L_1

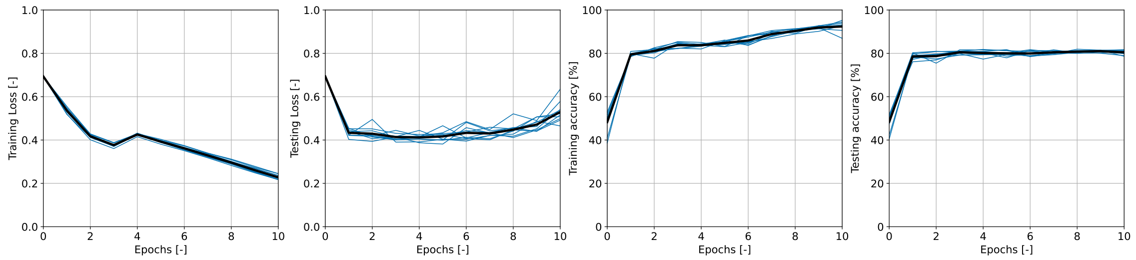
Figure A1: Training metrics of loss and accuracy on both train and test sets for different models on the diabetic retinopathy dataset. The colored thinner lines correspond to one of the 10 sets used and the thicker black line corresponds to the mean of the training data.



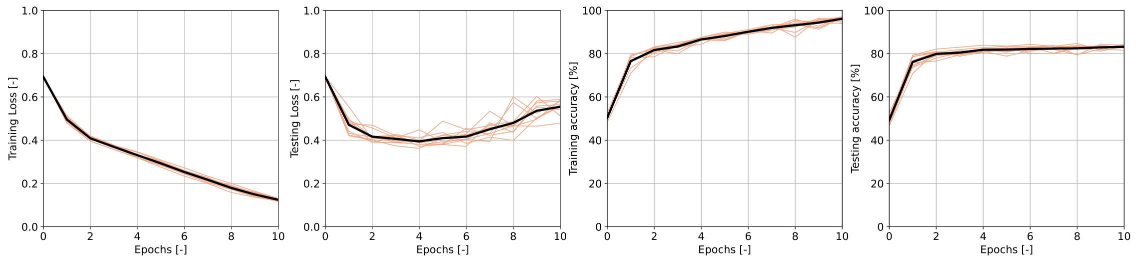
(a) Nominal



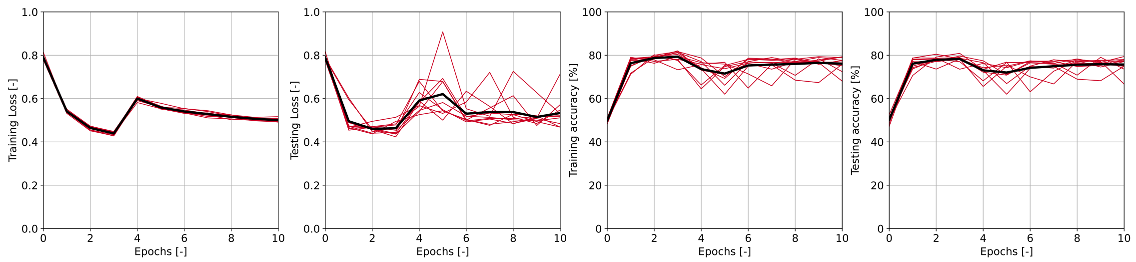
(b) FGSM



(c) PGD

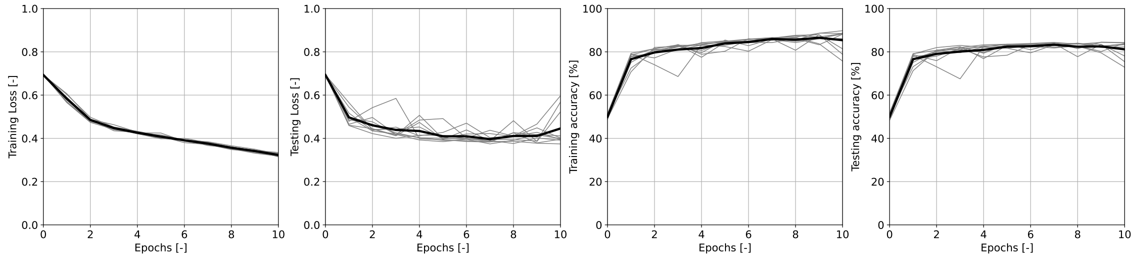


(d) aRUB-L_∞

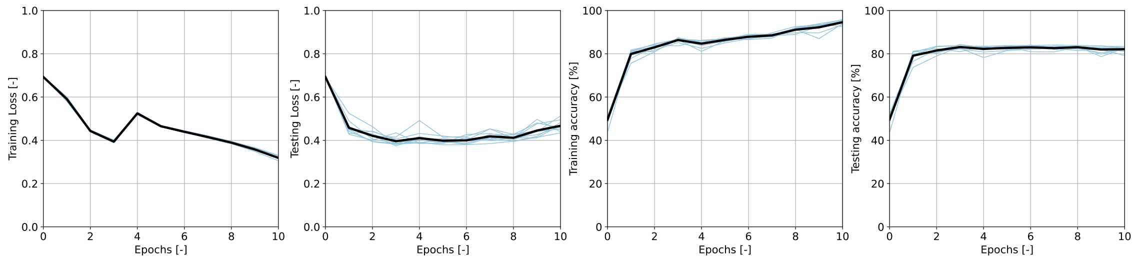


(e) aRUB-L_1

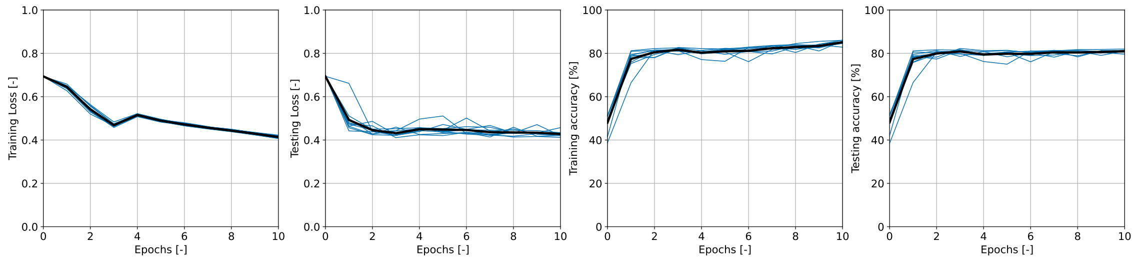
Figure A2: Training metrics of loss and accuracy on both train and test sets for different models on the nevus dataset. The colored thinner lines correspond to one of the 10 sets used and the thicker black line corresponds to the mean of the training data.



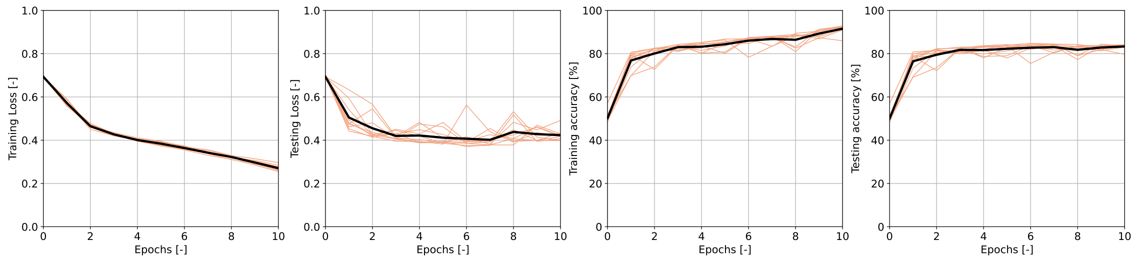
(a) Nominal



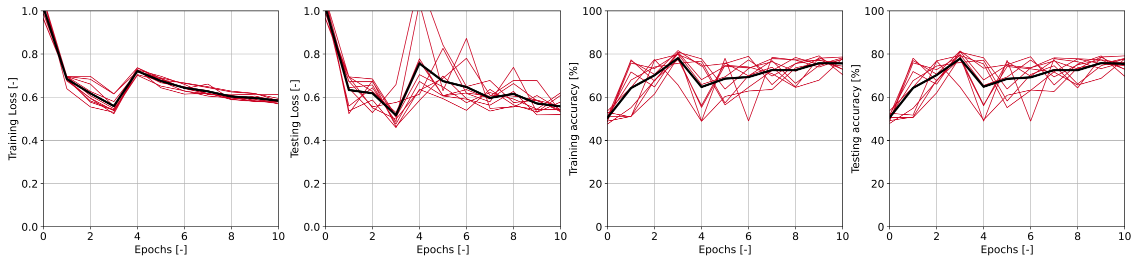
(b) FGSM



(c) PGD



(d) aRUB-L_∞



(e) aRUB-L_1

Figure A3: Training metrics of loss and accuracy on both train and test sets for different models on the chest X-ray dataset. The colored thinner lines correspond to one of the 10 sets used and the thicker black line corresponds to the mean of the training data.