Towards Robust Deep Learning: Lipschitz Continuity and Contractive Layers

A Thesis submitted for the completion of
requirements for the degree of

Master of Science
(Research)

by

Jayesh Kumar Jaiswal
Undergraduate Programme
Indian Institute of Science

Under the supervision of

Prof. Kunal Narayan Chaudhury
Department of Electrical Engineering, Indian Institute of Science

Prof. Ranjan Laha
Department of Physics, Indian Institute of Science

Acknowledgements

I would like to express my sincere gratitude to my thesis supervisor, Prof. Kunal Narayan Chaudhury, for his invaluable guidance and support throughout my research journey. His expertise, patience, and insightful comments have been instrumental in shaping my ideas and helping me to stay on track.

I am also grateful to Prof. Ranjan Laha for his assistance in supervising my thesis. His expert advice and feedback have been invaluable in shaping my research, and I appreciate his willingness to provide guidance when my primary supervisor was unavailable.

My heartfelt thanks go to my friends and family for their encouragement and unwavering belief in my abilities. Their support has helped me to persevere through challenging times and stay motivated to achieve my goals.

Finally, I would like to express my gratitude to all the research participants who generously gave their time and effort to participate in my study. Their willingness to share their experiences and insights has been invaluable in advancing knowledge in my field of research.

Thank you all for your support and guidance.

Declaration

I hereby declare that this work, is my own and was carried out under the supervision of Prof. Kunal Narayan Chaudhury, Associate Professor at the Indian Institute of Science, Bengaluru. It was carried out during my undergraduate studies for a Master of Science (Research) Degree at the Indian Institute of Science from 2023 to 2024.

Wherever I have referred to another author's work, I have rightly attributed it in the references. I have acknowledged the people who have helped me in the lab to carry out my experiments.
This work has not been submitted previously for any other degree or diploma.

Jayesh

Jayesh Kumar Jaiswal
Undergraduate Program
Indian Institute of Science
Bengaluru, PIN: 560012

Certificate

This is to certify that the Master's thesis titled "Towards Robust Deep Learning: Lipschitz Continuity and Contractive Layers" submitted by Jayesh Kumar Jaiswal (Sr No. 11-01-00-10-91-19-1-17509) to Indian Institute of Science, Bangalore towards partial fulfilment of requirements for the award of degree of Master of Science (Research) in Physics is a record of bona fide work carried out by him under my supervision and guidance during Academic Year, 2023-24.

Prof. Kunal Narayan Chaudhury
Associate Professor
Department of Electrical Engineering
Indian Institute of Science
Bengaluru, PIN: 560012

Dr. Ranjan Laha
Assistant Professor
CHEP Department
Indian Institute of Science
Bengaluru, PIN: 560012

Abstract

This research investigates methods for improving the robustness of neural networks against adversarial attacks, focusing on the concept of Lipschitz continuity and the implementation of convex potential layers. We explore the theoretical foundations of adversarial attacks and robustness certification, highlighting the role of the Lipschitz constant as a key metric for evaluating a classifier's resilience to perturbations.

The study delves into the design and implementation of Lipschitz neural networks, employing techniques such as weight clipping, spectral normalization, and Lipschitz activation functions to constrain the model's sensitivity to input variations. We introduce the novel architecture of Convex Potential Layer Networks (CPL-Nets), which incorporate convex potential layers to enhance robustness while maintaining competitive accuracy.

Extensive experiments on the CIFAR-10 dataset demonstrate the effectiveness of CPL-Nets in achieving state-of-the-art certified accuracy and robustness against various adversarial attacks, including AutoAttack and Projected Gradient Descent (PGD). Comparisons with standard ResNet models reveal that while ResNet may achieve higher accuracy, CPL-Nets exhibit superior resilience to adversarial perturbations.

Furthermore, we explore the concept of contractive layers as an extension to 1-Lipschitz layers, proving their theoretical properties and evaluating their impact on model performance. While contractive layers show promise in improving certified accuracy, further investigation is needed to address their limitations under strong adversarial attacks and within deeper architectures.

This research contributes to the ongoing effort of developing robust and reliable neural networks for real-world applications, paving the way for future advancements in adversarial defense mechanisms.

Contents

Chapter 1

Introduction

1.1 Introduction to Neural Networks

Neural networks, inspired by the biological neural networks of the human brain, are computational models designed to simulate the learning process of biological neurons. These models consist of interconnected artificial neurons organized into layers, with each layer responsible for processing specific aspects of input data. Through a process of learning from examples, neural networks can generalize patterns and make predictions on unseen data, making them indispensable tools in modern machine learning.

At its core, a neural network comprises three main types of layers: input, hidden, and output layers. The input layer receives raw data, which is then transformed and processed through one or more hidden layers containing interconnected neurons. Finally, the output layer produces the network's prediction or classification based on the learned features. The connections between neurons are characterized by weights, which are adjusted during the training process to optimize the network's performance

Training a neural network involves presenting it with labeled data and iteratively adjusting its weights and biases to minimize the difference between predicted and actual outputs. This optimization process, often achieved through backpropagation and gradient descent algorithms, allows the network to learn complex patterns and relationships within the data. By updating its parameters based on the calculated error, the network gradually improves its ability to make accurate predictions.

Modern neural networks face susceptibility to imperceptible adversarial perturbations in real-world applications. Addressing this vulnerability is crucial as neural networks are increasingly integrated into production systems. Existing defenses offer theoretical guarantees against adversarial perturbations. Common challenges include tradeoffs between expressivity and robustness, as well as scalability and accuracy. A promising method to fortify classifiers is by enforcing Lipschitzness properties.

In our pursuit of crafting robust defense mechanisms against adversarial attacks, we begin by formally defining adversarial attacks and robustness certification. Our focus lies in a classification task that maps input vectors from a space $X \subset \mathbb{R}^d$ to a label space $Y := \{1, \ldots, K\}$. To this end, our objective is to learn a classifier function $f := (f_1, \ldots, f_K) : X \to \mathbb{R}^K$, where the predicted label for an input x is given by $\arg\max_k f_k(x)$. For a given input-label pair (x, y), we define correct classification as $F(x) = y$.

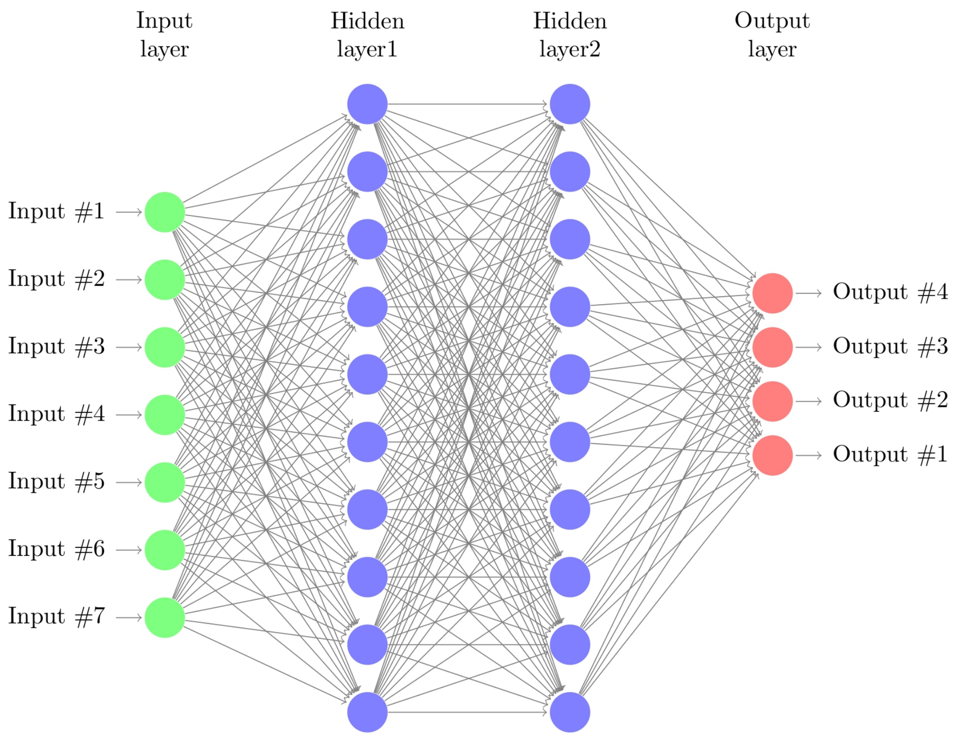


Figure 1.1: Neural Network Architecture.

1.2 Adversarial Attacks

Adversarial attacks exploit the inherent vulnerabilities of neural networks by introducing carefully crafted perturbations to input data, leading to erroneous predictions. These perturbations are often imperceptible to human observers but can significantly alter the network's output.

1. **Untargeted Attacks:**[14] These aim to misclassify the input without specifying a particular target class.
2. **Targeted Attacks:**[15] Here, the attacker intends to force the network to predict a specific target class.
3. **White-box Attacks:**[16] The attacker has complete knowledge of the network architecture and parameters.
4. **Black-box Attacks:**[17] The attacker has limited or no access to the network's internal structure and parameters.

Let $x \in X$, $y \in Y$ be the label of x, and f be a classifier. An adversarial attack at level ε introduces a perturbation τ such that $\|\tau\| \leq \varepsilon$ and results in

$$F(x + \tau) \neq y \tag{1.1}$$

1.3 Robust Certification

Robustness certification is paramount for deploying neural networks in safety-critical applications. It involves validating that the network's predictions remain consistent and reliable in the presence of potential adversarial perturbations within a certain radius

For $x \in X$, $y \in Y$ the label of x, and f as the classifier, we define certifiable robustness at radius $\varepsilon \geq 0$ for a point x. A classifier f is certifiably robust if, for all τ such that $\|\tau\| \leq \varepsilon$, the condition

$$F(x + \tau) = y \tag{1.2}$$

holds.
The task of robust certification revolves around developing methods that guarantee this certifiable robustness property. A pivotal metric in this context is the Lipschitz constant of the classifier.

Chapter 2

Lipschitz Continuity

2.1 Definition

Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be a function. The function f is Lipschitz continuous on a set $D \subseteq \mathbb{R}^n$ if there exists a constant $L \geq 0$ such that for all $x, y \in D$, the following inequality holds:

$$\|f(x) - f(y)\| \leq L\|x - y\| \tag{2.1}$$

Here, $\| \cdot \|$ denotes a norm on $\mathbb{R}^m$, typically the Euclidean norm.

2.2 Lipschitz Constant

The smallest constant L satisfying the Lipschitz condition is called the Lipschitz constant of f on D. If such a constant exists, f is said to be Lipschitz continuous on D.

- The Lipschitz constant L serves as an essential measure of the function's smoothness.
- When $L = 0$, f is identically constant.

2.3 Geometric Interpretation

Lipschitz continuity implies that the function f does not change too rapidly. The Lipschitz constant L bounds the ratio of the change in the output to the change in the input. Geometrically, it means that the graph of f cannot have steep slopes.

1. **Local Lipschitz Continuity:** If f is Lipschitz continuous on a neighborhood of each point in D, then f is locally Lipschitz continuous on D.

2. **Global Lipschitz Continuity:** If f is Lipschitz continuous on D with Lipschitz constant L, then f is globally Lipschitz continuous on D with the same Lipschitz constant.

Chapter 3

Lipschitz Neural Networks

3.1 Introduction

In the context of neural networks, enforcing Lipschitz continuity can bring several advantages:

1. **Robustness**[1, 3, 4]**:** Lipschitz continuity helps make neural networks robust to small changes in input data. This robustness is crucial for applications where the model needs to generalize well to unseen or perturbed data.
2. **Stability**[2, 5]**:** Lipschitz neural networks are more stable during training, and slight changes in the input or parameters lead to bounded changes in the output. This stability is beneficial for convergence during optimization.
3. **Generalization**[6]**:** The Lipschitz property can enhance the generalization capabilities of a neural network by preventing the model from fitting noise or outliers too closely.

3.2 Formal Characterization of Robustness

Proposition[1]**:** For an L-Lipschitz continuous classifier f in the ℓ_2 norm, where $\varepsilon \geq 0$, $x \in \mathcal{X}$, and $y \in \mathcal{Y}$ is the label of x, the margin $M_f(x)$ at point x is defined as:

$$M_f(x) := \max(0, f_y(x) - \max_{y' \neq y} f_{y'}(x)) \tag{3.1}$$

If at point x, the margin $M_f(x)$ satisfies:

$$M_f(x) := \max(0, f_y(x) - \max_{y' \neq y} f_{y'}(x)) > \sqrt{2} L \varepsilon \tag{3.2}$$

then we have for every perturbation τ with $||\tau||_2 \leq \varepsilon$, the following holds:

$$\arg\max_k f_k(x + \tau) = y \tag{3.3}$$

This provides a straightforward method to compute a robustness certificate for a given point. Consequently, in the pursuit of building robust neural networks, it is imperative to ensure a large margin and a small Lipschitz constant to attain optimal guarantees on robustness.[12]

3.3 Architecture

Enforcing Lipschitz continuity in neural networks often involves architectural choices, such as weight clipping, spectral normalization, or using specific activation functions known for their Lipschitz properties (e.g., bounded activation functions like tanh).

1. **Weight Clipping[15]:** Limiting the range of weights within a predefined interval helps control the Lipschitz constant.
2. **Spectral Normalization[18]:** Normalizing weight matrices to have a fixed spectral norm is a technique to ensure Lipschitz continuity in neural networks.
3. **Lipschitz Activation Functions[19]:** Choosing activation functions that satisfy Lipschitz continuity conditions, such as the bounded rectified linear unit (ReLU), can contribute to a Lipschitz neural network.

Chapter 4

Residual Neural Network

The motivation behind ResNet (Residual Neural Network) stems from the observation that as traditional neural networks become deeper, they encounter several challenges such as vanishing gradients and degradation in training accuracy. These challenges arise due to the difficulty of optimizing deep networks effectively.
The main idea behind ResNet is the introduction of residual connections, which allow information to bypass certain layers. This is achieved by adding skip connections that directly connect earlier layers to later layers. In a standard neural network without these connections, each layer is expected to approximate a desired underlying mapping. However, in practice, it's difficult for each layer to precisely do so, especially as the network becomes deeper. ResNet modifies this approach by explicitly encouraging the network to learn residuals (i.e., the difference between the desired mapping and the current approximation), rather than trying to learn the entire mapping from scratch.

The crux of the ResNet architecture lies in the novel residual block, mathematically defined as:

$$y = F(x, \{W_i\}) + x \tag{4.1}$$

$F(x, \{W_i\})$ denotes the learned residual function computed by the layers parameterized by weights $\{W_i\}$. This function captures the residual information required to transform the input x to its desired output. In simpler terms, $F(x, \{W_i\})$ represents the adjustments or deviations from the input necessary to reach the target output. Meanwhile, x stands for the input to the block, which could be the output of a previous layer or the initial input to the network.

If each F returns zero (e.g. because all the weights are 0), then this architecture simply passes the input x through unmodifed i.e., it computes the identity function.

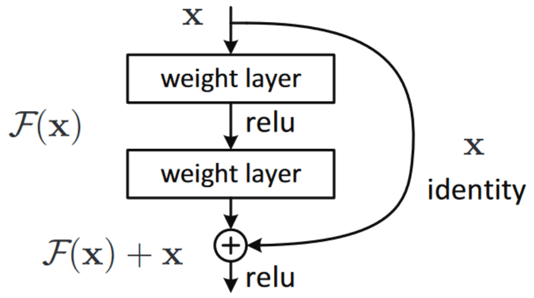
We can also see this algebraically in terms of the backprop equation for a residual block[9]:

$$x^{(l)} = x^{(l+1)} + x^{(l+1)} \cdot \frac{\partial F}{\partial x} = x^{(l+1)}(I + \frac{\partial F}{\partial x}) \tag{4.2}$$

Hence, if $\frac{\partial F}{\partial x} = 0$, the error signals are simply passed through unmodified. As long as $\frac{\partial F}{\partial x}$ is small, the Jacobian for the residual block will be close to the identity, and the error signals won't explode or vanish.

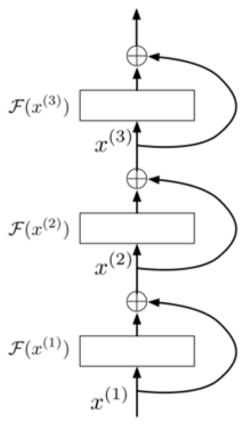
There's one important detail that needs to be mentioned: the input and output to a residual block clearly need to be the same size, because the output is the sum of the input and the residual function. But for conv nets, its important to shrink the images (e.g. using pooling) in order to expand the number of feature maps. ResNets typically achieve this by having a few convolution layers with a stride of 2, so that the dimension of the image is reduced by a factor of 2 along each dimension.

The addition of the input x through the shortcut connection serves as a highway for gradients during backpropagation. This ensures that even if the residual mapping $F(x, \{W_i\})$ approaches zero gradients, the direct connection allows for efficient gradient flow.



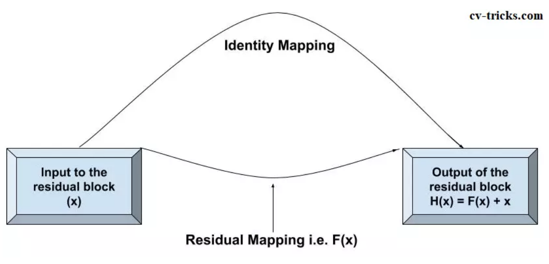
x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

(a) Residual learning: a building block. [8]

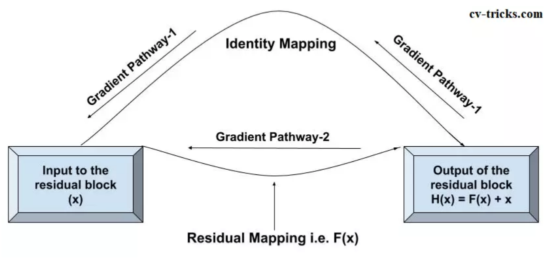


(b) Deep Residual Network [9]

Figure 4.1: Residual Block and Deep Residual Network


cv-tricks.com
Identity Mapping
Input to the residual block (x)
Output of the residual block H(x) = F(x) + x
Residual Mapping i.e. F(x)

(a) Residual Function


cv-tricks.com
Gradient Pathway-1
Identity Mapping
Gradient Pathway-1
Gradient Pathway-2
Input to the residual block (x)
Output of the residual block H(x) = F(x) + x
Residual Mapping i.e. F(x)

(b) Gradient Pathway

Figure 4.2: Representation of Residual Function and Gradient Pathway[10]

[11] During backpropagation, gradients in a residual block have two pathways to travel back to the input layer for weight updates. As illustrated in Figure 2b, they can follow path-1, which involves identity mapping, or path-2, which involves residual mapping. Along path-2, gradients encounter two layers(Figure 1), namely Weight Layer 2 and Weight Layer 1 in the residual network F(x), where the respective kernels or weights (W2 and W1) are updated, potentially leading to the computation of smaller or vanishing gradients, especially for initial layers. The inclusion of identity mapping, or shortcut connections, addresses this issue by allowing gradients to bypass the residual block entirely, thus facilitating more effective learning of the weights. In contrast, when gradients follow path-1, as depicted in Figure 2, they do not encounter any weight layers, thereby remaining unchanged in their computed values. This enables gradients to reach the initial layers directly, circumventing the residual block, and aiding in the proper learning of weights. Notably, in a ResNet basic block, ReLU function is applied after the F(x) + x operation, resulting in modifications to gradient values as they enter the residual block.

Chapter 5

Parametrizing Convex Potential Layers

5.1 Defining the potential function

For any vectors $\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_k \in \mathbb{R}^d$ and bias terms $b_1, b_2, \dots, b_k \in \mathbb{R}$, and for ϕ a convex function, the potential F defined as:

$$F_{(\mathbf{w},\mathbf{b})} : x \in \mathbb{R}^d \mapsto \sum_{i=1}^{k} \phi(\mathbf{w}_i^T x + b_i) \tag{5.1}$$

defines a convex function in x as the composition of a linear and a convex function. Its gradient with respect to its input x is then:

$$\nabla_x F_{(\mathbf{w},\mathbf{b})}(x) = \sum_{i=1}^{k} \mathbf{w}_i \phi'(\mathbf{w}_i^T x + b_i) = \mathbf{W}^T \phi'(\mathbf{W}x + \mathbf{b}) \tag{5.2}$$

with $\mathbf{W} \in \mathbb{R}^{k \times d}$ and $\mathbf{b} \in \mathbb{R}^k$ being respectively the matrix and vector obtained by the concatenation of w_i^T and b_i, and ϕ' is applied element-wise. Moreover, assuming ϕ' is L-Lipschitz, we have that $F_{(\mathbf{w},\mathbf{b})}$ is $L\|\mathbf{W}\|_2^2$-smooth. $\|\mathbf{W}\|_2$ denotes the spectral norm of $\mathbf{W}$, i.e., the greatest singular value of $\mathbf{W}$ defined as:

$$\|\mathbf{W}\|_2 := \max_{x \neq 0} \frac{\|\mathbf{W}x\|_2}{\|x\|_2} \tag{5.3}$$

The reciprocal also holds: if $\sigma : \mathbb{R} \to \mathbb{R}$ is a non-decreasing L-Lipschitz function, $\mathbf{W} \in$

$\mathbb{R}^{k\times d}$ and $\mathbf{b}\in\mathbb{R}^k$, there exists a convex $L\|\mathbf{W}\|_2^2$-smooth function $F_{(\mathbf{w},\mathbf{b})}$ such that

$$\nabla_x F_{(\mathbf{w},\mathbf{b})} = \mathbf{W}^T\sigma(\mathbf{W}x+\mathbf{b}), \tag{5.4}$$

where σ is applied element-wise.
To prove that $F_{(\mathbf{w},\mathbf{b})}(x)$ is differentiable, we need to show that the partial derivatives with respect to each element of x exist and are continuous. The gradient is a vector of these partial derivatives.

Let's denote $F_i(x)=\phi(\mathbf{w}_i^Tx+b_i)$. Then, $F_{(\mathbf{w},\mathbf{b})}(x)=\sum_{i=1}^k F_i(x)$.

Now, let's consider the partial derivative of $F_i(x)$ with respect to x_j (the j-th element of x):

$$\frac{\partial F_i}{\partial x_j}=\frac{\partial\phi(\mathbf{w}_i^Tx+b_i)}{\partial x_j} \tag{5.5}$$

Since $\phi(\cdot)$ is assumed to be convex, we are considering the composition of a convex function with a linear function $(\mathbf{w}_i^Tx+b_i)$. The composition of convex and linear functions is convex, and convex functions are differentiable almost everywhere.
Now, the chain rule tells us:

$$\frac{\partial F_i}{\partial x_j}=\phi'(\mathbf{w}_i^Tx+b_i)\cdot\frac{\partial(\mathbf{w}_i^Tx+b_i)}{\partial x_j}=\phi'(\mathbf{w}_i^Tx+b_i)\cdot w_{ij} \tag{5.6}$$

Here, w_{ij} is the j-th element of the vector $\mathbf{w}_i$.
Since $\phi'(\cdot)$ is assumed to be applied element-wise, and w_{ij} are constants, the derivative is well-defined.
The function $F_{(\mathbf{w},\mathbf{b})}(x)$ is the sum of differentiable functions, and the partial derivatives exist for each component. Therefore, $F_{(\mathbf{w},\mathbf{b})}(x)$ is differentiable.

5.2 Proving the Lipschitzness of the potential function

We know that if a function satisfies $\|F(x)-F(y)\|\leq L\|x-y\|_2$ for all $x,y\in F$, then it is Lipschitz continuous with Lipschitz constant L, where $\|\cdot\|_2$ denotes the L_2-norm.

Given that ϕ' is L-Lipschitz, we can express it as:

$$\|\phi'(x) - \phi'(y)\|_2 \leq L\|x - y\|_2, \tag{5.7}$$

and consequently:

$$\begin{aligned}\|\phi'(Wx + b) - \phi'(Wy + b)\|_2 &\leq L\|Wx - Wy\|_2 \\ &\leq L\|W\|_2\|x - y\|_2.\end{aligned} \tag{5.8}$$

This implies that $\phi'(Wx + b)$ is $L\|W\|_2$-Lipschitz.

Now, considering the gradient:

$$\nabla_x F_{(w,b)}(x) = W^T\phi'(Wx + b), \tag{5.9}$$

we have:

$$\begin{aligned}\|\nabla F_{(w,b)}(x) - \nabla F_{(w,b)}(y)\|_2 &= \|W^T\phi'(Wx + b) - W^T\phi'(Wy + b)\|_2 \\ &\leq \|W^T\|_2 L\|W\|_2\|x - y\|_2 \\ &\leq L\|W\|_2^2\|x - y\|_2,\end{aligned} \tag{5.10}$$

showing that $\nabla_x F_{(w,b)}(x)$ is Lipschitz continuous with Lipschitz constant $L\|W\|_2^2$. This implies that $F_{(w,b)}(x)$ is $L\|W\|_2^2$-smooth.

5.3 Convex Potential Layer

Utilizing the gradient method, we deduce the following convex potential layer:

$$G(x) = x - \frac{2}{L\|W\|_2^2}W^T\sigma(Wx + b) \tag{5.11}$$

This layer is 1-Lipschitz, and to establish this, we aim to demonstrate

$$\|G(x) - G(y)\|_2 \leq \|x - y\|_2 \tag{5.12}$$

To achieve this, we leverage the concept of co-coercivity of a gradient.

Expressing our convex potential layer as:

$$G(x) = x - \frac{2}{L}\nabla_x F_{(w,b)}(x) \tag{5.13}$$

where L is the Lipschitz constant of $\nabla_x F_{(w,b)}(x)$, given by $L\|W\|_2^2$.
First we will check the differentiability of z(x):

Now,

$$\|G(x) - G(y)\|_2^2 = \|x - \frac{2}{L}\nabla F_{(w,b)}(x) - (y - \frac{2}{L}\nabla F_{(w,b)}(y))\|_2^2. \tag{5.14}$$

$$\begin{aligned}\|G(x) - G(y)\|_2^2 = \|x - y\|_2^2 - \frac{4}{L}(\nabla F_{(w,b)}(x) - \nabla F_{(w,b)}(y))^T(x-y) \\ + \frac{4}{L^2}\|\nabla F_{(w,b)}(x) - \nabla F_{(w,b)}(y)\|_2^2.\end{aligned} \tag{5.15}$$

By the co-coercivity of gradient of a smooth function [7], we know:

$$(\nabla f(x) - \nabla f(y))^T(x-y) \geq \frac{1}{L}\|\nabla f(x) - \nabla f(y)\|_2^2 \quad \text{for all } x, y. \tag{5.16}$$

Therefore,

$$(\nabla F_{(w,b)}(x) - \nabla F_{(w,b)}(y))^T(x-y) \geq \frac{1}{L}\|\nabla F_{(w,b)}(x) - \nabla F_{(w,b)}(y)\|_2^2, \tag{5.17}$$

leading to:

$$\begin{aligned}\|G(x) - G(y)\|_2^2 \leq \|x - y\|_2^2 - \frac{4}{L^2}\|\nabla F_{(w,b)}(x) - \nabla F_{(w,b)}(y)\|_2^2 \\ + \frac{4}{L^2}\|\nabla F_{(w,b)}(x) - \nabla F_{(w,b)}(y)\|_2^2.\end{aligned} \tag{5.18}$$

Consequently,

$$\|G(x) - G(y)\|_2^2 \leq \|x - y\|_2^2 \tag{5.19}$$

$\implies$

$$\|G(x) - G(y)\|_2 \leq \|x - y\|_2, \tag{5.20}$$

Hence, z is 1-Lipschitz.

5.4 Computation of Spectral Norm

Within the scope of our research thesis, we strategically leverage the Power Iteration Method (PM) as a pivotal computational technique for estimating the spectral norm of matrices. The PM is a numerical algorithm used to find the dominant eigenvalue and corresponding eigenvector of a square matrix. Mathematically, the Power Iteration Method operates iteratively through the following steps:

1. Initialize a random vector $\mathbf{v}_0$,
2. Iterate $\mathbf{v}_{k+1} = \dfrac{W\mathbf{v}_k}{\|W\mathbf{v}_k\|}$,
3. Repeat until convergence.

Here, W represents the given matrix, and $\mathbf{v}_k$ denotes the vector obtained at the k-th iteration. The Power Iteration Method converges geometrically, and after k iterations, the convergence rate is characterized by $O\left(\left(\frac{\lambda_2}{\lambda_1}\right)^k\right)$, where $\lambda_1 > \lambda_2$ are the two leading singular values of matrix W.

While the PM may seemingly demand a high number of iterations for convergence, during the training phase, we capitalize on the observation that weight matrices W undergo slow changes. Consequently, we opt for a pragmatic approach, executing just one iteration of the PM for each training step. This strategic choice ensures computational efficiency, allowing the algorithm to gradually converge with the training process.

However, during the evaluation phase, particularly when computing certified adversarial robustness, ensuring the PM's convergence becomes imperative. To address this, we perform 100 iterations for each layer at inference time. Importantly, it is worth noting that at this stage, the computation of the spectral norm only necessitates execution once for each layer.

Chapter 6

Experiments

6.1 Dataset

In the course of this thesis, the CIFAR-10 dataset has been utilized as a fundamental component for training, validating, and testing the proposed model.

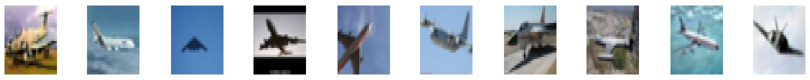
6.1.1 Overview

The CIFAR-10 dataset is a well-known collection of color images used extensively in the field of computer vision and machine learning. Developed by the Canadian Institute for Advanced Research (CIFAR), this dataset serves as a standard benchmark for image classification tasks. It comprises a total of 60,000 32x32-pixel images, distributed across 10 distinct classes. Each class contains 6,000 images, providing a diverse and comprehensive set for training and testing machine learning models.

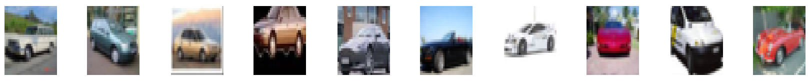
6.1.2 Dataset Composition

The CIFAR-10 dataset is organized into the following 10 classes:

1. Airplane

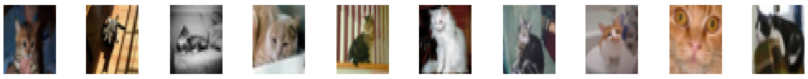


2. Automobile



3. Bird

4. Cat



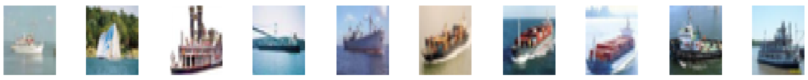
5. Deer

6. Dog

7. Frog

8. Horse

9. Ship



10. Truck

This diversity allows researchers and practitioners to evaluate the ability of their models to discern between a variety of objects and animals commonly encountered in everyday scenes.

6.1.3 Splitting of Data

The dataset is divided into two main subsets:

1. **Training Set:** Consists of 50,000 images, serving as the data used for training machine learning models.
2. **Testing Set:** Comprises 10,000 images reserved for evaluating the performance and generalization ability of trained models.

This split ensures that the model is tested on previously unseen data, providing a reliable measure of its effectiveness in real-world scenarios.

6.1.4 Data Preprocessing and Augmentation

In our project, the CIFAR-10 dataset is subject to preprocessing steps to ensure compatibility with our model architecture. The following procedures are applied:

- **Normalization:** The dataset is normalized using mean and standard deviation values calculated from the training set. This normalization helps in stabilizing the learning process and enhancing model convergence.
- **Padding:** To facilitate border handling and augment the dataset, padding is applied to each image using a reflective mode.
- **Transpose:** Images are transposed to switch from the NHWC (number of images, height, width, channels) format to the NCHW format, which is often preferred in neural network architectures.
- **Data Augmentation:** Various data augmentation techniques are employed, including cropping, flipping, and padding, to increase the diversity of the training dataset and improve the model's generalization.

The mean and standard deviation values used for normalization are $[0.4914, 0.4822, 0.4465]$ and $[0.2471, 0.2435, 0.2616]$, respectively. Additionally, a padding of 4 pixels is applied to each side of the images.

6.1.5 Training and Evaluation

Our training and evaluation process involve several key components:

- **Training Loop:** The model is trained using an optimization loop, where batches of preprocessed data are fed into the model. The model parameters are updated based on the computed loss.
- **Optimization:** The Adam optimizer is employed with a specified learning rate schedule. The learning rate is adjusted dynamically during training to improve convergence.
- **Data Batching:** The training and test datasets are batched using a custom batching class. Data augmentation techniques are selectively applied during training to enhance the model's ability to handle diverse inputs.
- **Training Epochs:** The training process spans multiple epochs, allowing the model to iteratively learn from the dataset. Performance metrics, including accuracy and loss, are monitored throughout the training phase.
- **Model Evaluation:** The trained model is evaluated on a separate test set to assess its generalization performance. Key metrics such as accuracy and loss are reported for comprehensive analysis.

The training and evaluation procedures are carried out for a specified number of epochs, with the Adam optimizer dynamically adjusting the learning rate for optimal convergence.

6.2 Model Architecture

6.2.1 Convex Potential Layer Convolutional Block

The Convex Potential Layer Convolutional Block is a key component of the proposed model. It employs a convex potential layer to introduce non-linearity into the convolutional layers. The layer is initialized with learnable parameters, including a convolutional kernel and bias. The spectral normalization power method is utilized for weight normalization, contributing to the stability of the training process. During training, a dynamic adjustment of the singular value is performed to enhance convergence.

6.2.2 Convex Potential Layer Linear Block

Similarly, the Convex Potential Layer Linear Block applies the convex potential layer to linear transformations. It employs a similar approach to weight initialization and normalization as the convolutional block. The spectral normalization power method ensures stability during training, with a dynamic adjustment of singular values.

6.2.3 Normalization and Transformation

The model incorporates normalization techniques for input data, ensuring standardized and stable processing. The Normalize module is employed to center and scale input data. Furthermore, a Linear Normalization module is utilized, applying spectral normalization to linear layers.

6.2.4 Model Composition

The Convex Potential Layer Network is composed of these Convex Potential Layer Blocks, interleaved with pooling and flattening layers. The network architecture encompasses both convolutional and linear blocks, providing a hierarchical and expressive representation of input data.

6.2.5 Hyperparameters and Configuration

- **Depth**: The depth of the Convex Potential Layer Network, specifying the number of Convex Potential Layer Convolutional Blocks.
- **Depth Linear**: The depth of linear blocks that follow the convolutional layers.
- **Convolutional Size**: The size of the convolutional kernel in Convex Potential Layer Convolutional Blocks.
- **Number of Channels**: The number of channels utilized in convolutional layers.
- **Number of Features**: The dimensionality of the feature space in linear blocks.
- **Use LLN (Last Layer Normalization)**: A binary flag indicating whether Normalization is applied to the final layer.

6.2.6 Training Dynamics

The model is trained using a dynamic adjustment of the singular value during both training and evaluation. The use of spectral normalization power methods contributes to stable training and robust convergence.

6.3 Training Convex Potential Layer Network

In this section, we describe the training procedure for the Convex Potential Layer Network, which is designed to enhance the robustness of our neural network through the incorporation of convex potential layers. Our training approach involves optimizing the model parameters using the Adam optimizer with a margin loss function. The following subsections detail the key components of our training methodology.

6.3.1 Model Initialization

To ensure reproducibility, we set the random seed using `torch.manual_seed` and employ random initialization for the model weights. Additionally, input data is normalized using mean and standard deviation values.

6.3.2 Data Handling

We utilize the `DataClass` for loading and processing the dataset. The input data is normalized with mean and standard deviation values. The data is then split into training and testing batches.

6.3.3 Model Configuration

The Convex Potential Layer Network is configured with specific parameters, including depth, linear depth, number of channels, and more. The model is wrapped with `NormalizedModel` for normalization and then parallelized across multiple GPUs using `DataParallel`. The model is subsequently moved to the GPU for training.

6.3.4 Parameters Training

In our model, the training of parameters involves meticulous procedures aimed at enhancing stability, convergence, and generalization performance.

1. **Parameter Initialization:** Weights are initialized using the Kaiming uniform initialization method, which helps in preventing vanishing or exploding gradients during training. Bias terms are initialized uniformly within a bound determined by the fan-in of the layer.

    (a) **Kaiming Uniform Initialization**: Kaiming initialization, also known as He initialization, is designed to address the issue of vanishing or exploding gradients, which can hinder the training of deep neural networks. This initialization method initializes the weights of the neural network layer according to the following formula:

$$W \sim U\left(-\sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}}}}, \sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}}}}\right) \tag{6.1}$$

Where n_{in} and n_{out} are the number of input and output units of the layer, respectively. By using this initialization scheme, the weights are set to be neither too small nor too large, which helps in stabilizing the training process and preventing gradients from vanishing or exploding during backpropagation. This initialization is based on the assumption that the activation function used in the neural network is symmetric and linear. For activation functions like ReLU (Rectified Linear Unit), which is widely used in deep learning, this initialization method has been shown to be effective in preventing gradient-related issues.

(b) **Bias Initialization**: Biases are initialized uniformly within a bound determined by the fan-in of the layer. The fan-in of a layer refers to the number of inputs to the layer. In the provided code, the bound for bias initialization is set to $\frac{1}{\sqrt{\text{fan\_in}}}$. Initializing biases within a specific range helps in breaking the symmetry in the activation of neurons in the network. By providing each neuron with a slightly different initial bias, the network can start learning from different points in the input space, which can aid in learning diverse representations and improve the capacity of the network to capture complex patterns in the data.

The rationale behind these initialization techniques is to set the initial weights and biases of the neural network in such a way that the network can effectively learn from the data without facing issues such as vanishing or exploding gradients. By providing suitable initial conditions, the training process can be more stable and efficient, leading to better convergence and performance of the neural network.

2. **Spectral Normalization:**

   Spectral normalization is a vital component of our training regimen. Applied to the weights, this technique ensures that the spectral norm of each weight matrix remains bounded throughout the training process. Through spectral normalization, the Lipschitz constant of linear transformations (including both convolutional and linear layers) is effectively constrained, bolstering training stability and thwarting overfitting. During forward propagation, the output of each linear transformation is divided by the spectral norm of the corresponding weight matrix, thus enforcing this crucial constraint.

3. **Dynamic Lipschitz Constant Adjustment:**

The Lipschitz constant of the network undergoes dynamic adjustment during training. This adaptive regulation is facilitated by estimating the spectral norm of the weight matrices utilizing the power iteration method. By dynamically adjusting the Lipschitz constant, we ensure stable and efficient training, while safeguarding generalization performance. The Lipschitz constant is adaptively regulated to strike a delicate balance between stability and performance.

4. **Optimization:**

   We employ the Adam optimizer with weight decay regularization to iteratively update the parameters of the model. This optimizer, coupled with weight decay, aids in preventing overfitting and fine-tuning the model's performance. Additionally, the learning rate is dynamically adjusted using a triangular learning rate scheduler, ensuring smooth convergence and preventing oscillations during training. Alongside optimization, backpropagation serves as the cornerstone of parameter training. During each training iteration, gradients of the loss function with respect to the parameters (both weights and biases) are computed using backpropagation. These gradients guide the adjustment of parameters, minimizing the loss function and driving the model towards optimal performance. Moreover, the use of margin loss with a specified margin parameter further enhances the model's discriminative power, tailoring it to the specific requirements of the target task.

In summary, the training of parameters in our model encompasses a comprehensive approach, integrating initialization strategies, spectral normalization, dynamic Lipschitz constant adjustment, and backpropagation. These meticulously orchestrated processes collectively underpin the stability, convergence, and generalization prowess of the convolutional neural network.

6.3.5 Loss Function

We utilize the margin loss function as the training criterion. The margin loss aims to enforce a margin between the scores of correct and incorrect classes.

Definitions:

- $f(x)$: The output of the neural network for input x.
- y: The true class label.
- m: The margin parameter.

Margin Loss Function: The margin loss $L(x, y)$ is defined as the sum of the hinge losses over all incorrect classes j (where $j \neq y$), ensuring that the score of the correct class y is at least m higher than the score of any incorrect class j:

$$L(x, y) = \sum_{j \neq y} \max(0, f(x)_j - f(x)_y + m) \tag{6.2}$$

Breakdown of Terms:

- $f(x)_j$: predicted score for class j.
- $f(x)_y$: predicted score for the true class y.
- $\max(0, f(x)_j - f(x)_y + m)$: The hinge loss, penalizing cases where the score of the incorrect class (j) is too close to or exceeds the score of the true class (y) by less than the specified margin m.

$$f(x)_y \geq f(x)_j + m \tag{6.3}$$

This relationship ensures that the score of the correct class (y) is at least m greater than the score of any incorrect class (j). In other words, it ensures that the correct class score is sufficiently higher than the scores of incorrect classes by at least the specified margin m. If this condition is not met, the hinge loss term $\max(0, f(x)_j - f(x)_y + m)$ will penalize the prediction.

The margin loss encourages the correct class score to be higher than the scores of incorrect classes by at least a margin m. This loss function is utilized to learn representations that separate instances of different classes by a certain margin, promoting better discrimination between classes during training.

6.3.6 Training Loop

The training loop iterates over batches of training data. The model is set to training mode, and gradients are updated using backpropagation. Learning rate updates are performed based on the learning rate scheduler.

6.3.7 Logging and Monitoring

During training, we log the training loss and print progress updates. This information is crucial for monitoring the model's learning progress.

6.3.8 Evaluation

The model is evaluated on the test dataset to assess its accuracy and robustness against adversarial attacks. Certified accuracy is calculated using Lipschitz constant and epsilon values. Additionally, AutoAttack and PGD attacks are employed for further evaluation.

1. **Accuracy:**
Definition: Accuracy is a metric we used to measure the overall correctness of our classification model. It represents the ratio of correctly predicted instances to the total instances in the dataset. The formula for accuracy is:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \times 100\%$$

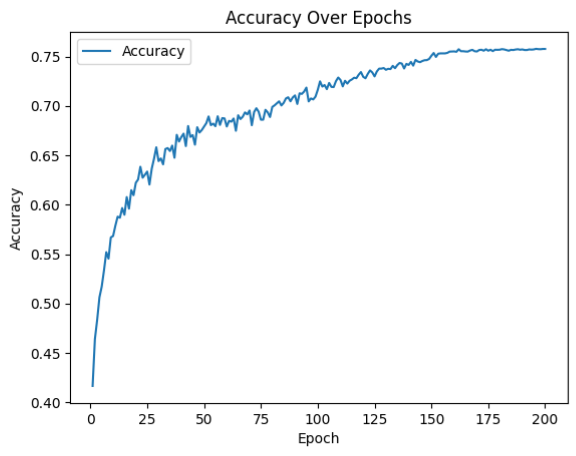
2. **Certified Accuracy:**
Definition: The concept of certified accuracy in the context of our Lipschitz neural network refers to the model's precision under a Lipschitz continuity constraint. The certified accuracy function quantifies this accuracy by evaluating the percentage of correctly classified instances that also satisfy a Lipschitz continuity condition within a specified region. The Lipschitz constraint is determined by comparing the margins between the highest and second-highest predicted class probabilities. The function provides detailed accuracy statistics for both instances meeting the Lipschitz constraint (certified) and those that do not.
3. **AutoAttack Accuracy:**
Definition: AutoAttack is a method we used to evaluate the robustness of our model against adversarial attacks. AutoAttack, an automated procedure, amalgamates various adversarial attacks, including the PGD attack. The function calculates the accuracy of the Lipschitz neural network on the test set when subjected to adversarial perturbations generated by the AutoAttack method. This analysis provides a comprehensive understanding of the model's performance under diverse adversarial scenarios.
4. **PGD (Projected Gradient Descent) Attack Accuracy:**
Definition: PGD attack is a type of white-box adversarial attack we used to evaluate the robustness of our model. This adversarial attack crafts perturbed input samples within a specified L2 ball (L2 norm constraint) around the original inputs while maximizing the model's loss. PGD attack accuracy is a measure of how well our model performs when subjected to such iterative adversarial attacks. It provides insights into the model's susceptibility to more sophisticated adversarial manipulations. The accuracy under PGD attacks is usually lower than the standard accuracy, indicating potential vulnerabilities in the model.

Table 6.1: Architectures description for our Convex Potential Layers (CPL) neural networks with different capacities.

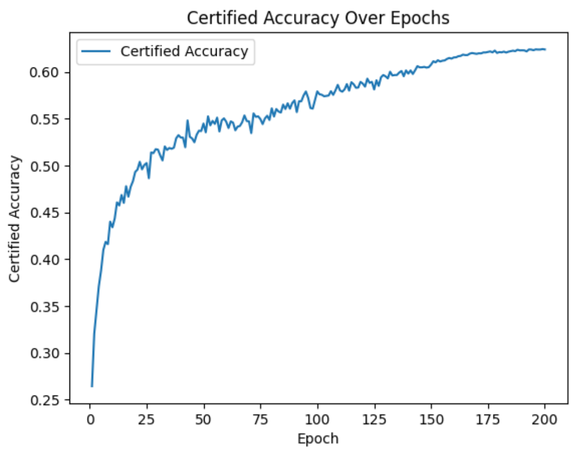
#	Conv. Layers	Channels	Linear Layers	Linear Features
CPL-S	20	45	7	2048
CPL-M	30	60	10	20481

Table 6.2: Results for CPL-S (convex potential layer-small) without using LLN(Last layer normalization) under different epsilon (Perturbations) values.

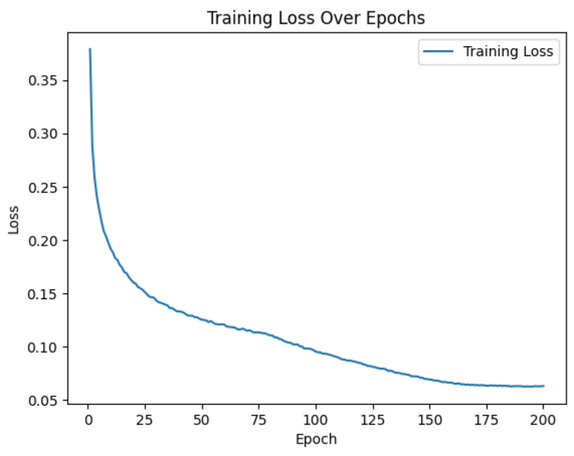
ϵ	Accuracy	Certified Acc.	Autoattack Acc.	PGD Attack Acc.
36/255	75.76	62.41	70.15	71.81
72/255	75.76	46.8	63.41	66.61
108/255	75.76	32.41	56.15	61.73



(a) Accuracy



(b) Certified Accuracy

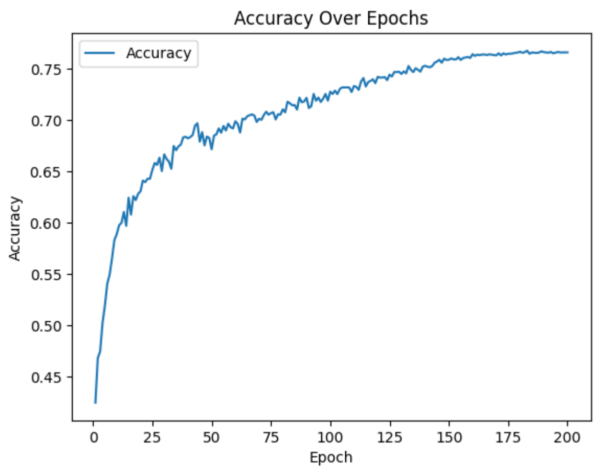


(c) Train Loss

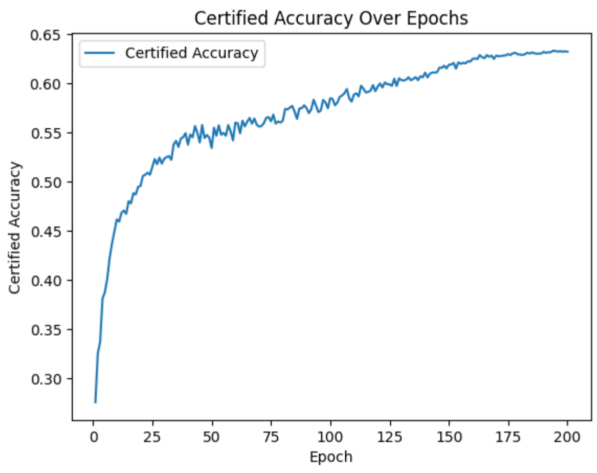
Figure 6.1: CPL-S without using LLN Plots of accuracy, certified accuracy, and train loss.

Table 6.3: Results for CPL-M(convex potential layer-Medium) without using LLN under epsilon (Perturbations) value.

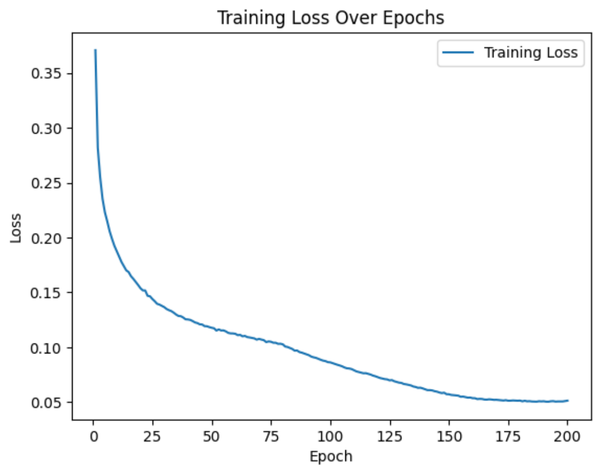
ϵ	**Accuracy**	**Certified Acc.**	**Autoattack Acc.**	**PGD Attack Acc.**
36/255	76.58	63.25	70.48	72.05



(a) Accuracy



(b) Certified Accuracy

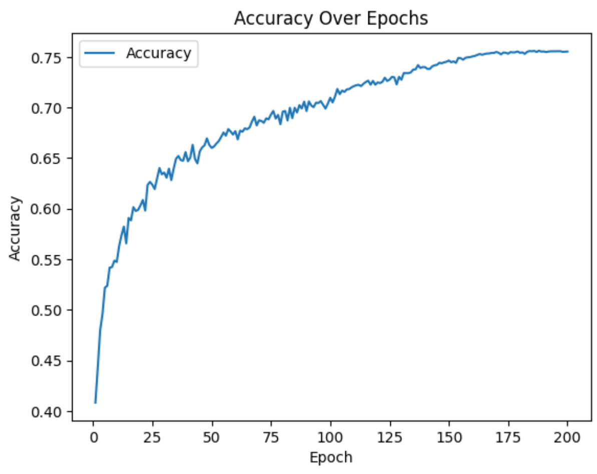


(c) Train Loss

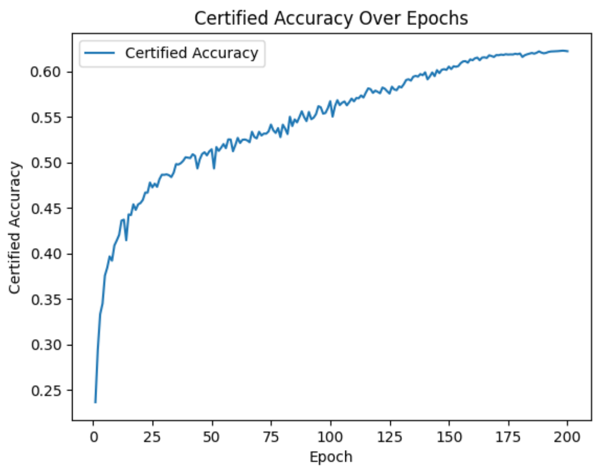
Figure 6.2: CPL-M without using LLN Plots of accuracy, certified accuracy, and train loss.

Table 6.4: Results for CPL-S using LLN under different epsilon (Perturbations) values.

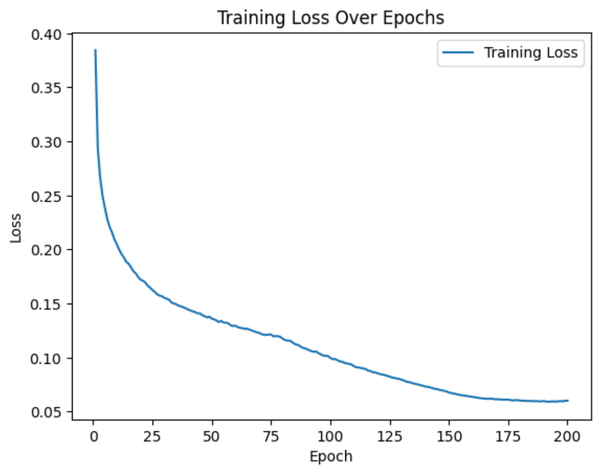
ϵ	**Accuracy**	**Certified Acc.**	**Autoattack Acc.**	**PGD Attack Acc.**
36/255	75.54	62.21	69.68	71.21
72/255	75.54	46.84	63.63	66.73



(a) Accuracy



(b) Certified Accuracy



(c) Train Loss

Figure 6.3: CPL-S using LLN Plots of accuracy, certified accuracy, and train loss.

6.3.9 Results and Analysis

The training of Convex Potential Layer Networks (CPL-Nets) on the CIFAR-10 dataset yielded insightful results, showcasing the impact of convex potential layers on model performance. Here's a breakdown of the key findings:

CPL-S without LLN:

- Achieved a standard accuracy of 75.76%.
- Exhibited strong certified accuracy, particularly for smaller epsilon values (perturbations). For instance, at $\epsilon = 36/255$, the certified accuracy reached 62.41%. This highlights the model's robustness against small adversarial perturbations.
- Maintained reasonable accuracy under AutoAttack (70.15%) and PGD attack (71.81%) for $\epsilon = 36/255$, demonstrating its resilience against these adversarial attack methods.

CPL-M without LLN:

- Demonstrated a slightly higher standard accuracy of 76.58% compared to CPL-S, suggesting the benefit of increased network capacity.
- Showcased improved certified accuracy with 63.25% at $\epsilon = 36/255$, further supporting the positive impact of a larger architecture.
- Performed similarly to CPL-S under AutoAttack (70.48%) and PGD attack (72.05%) for $\epsilon = 36/255$.

CPL-S with LLN:

- Achieved a standard accuracy of 75.54%, similar to the model without LLN.
- Exhibited comparable certified accuracy to the model without LLN, reaching 62.21% at $\epsilon = 36/255$.
- Showed similar performance under AutoAttack (69.68%) and PGD attack (71.21%) for $\epsilon = 36/255$ compared to the model without LLN.

These results demonstrate the effectiveness of convex potential layers in enhancing the robustness of neural networks against adversarial attacks. The observed trade-off between standard accuracy and certified accuracy suggests that CPL-Nets prioritize robustness while maintaining competitive classification performance. Additionally, the findings indicate that increasing network capacity can lead to further improvements in both accuracy and robustness.

Chapter 7

Comparing performance of pretrained models

7.1 Resnet18

In assessing the efficacy of our trained model, we conducted an extensive evaluation using the CIFAR-10 dataset. Our model demonstrated commendable accuracy, achieving competitive results during testing. However, to provide a comprehensive analysis, we compared its performance against two prevalent adversarial attack techniques: AutoAttack and PGDAttack.

Furthermore, to contextualize our findings, we benchmarked our model against a widely used pretrained model, ResNet-18. Notably, while ResNet-18 exhibited superior accuracy on the CIFAR-10 dataset compared to our model, our investigation revealed a substantial discrepancy in performance against adversarial attacks.

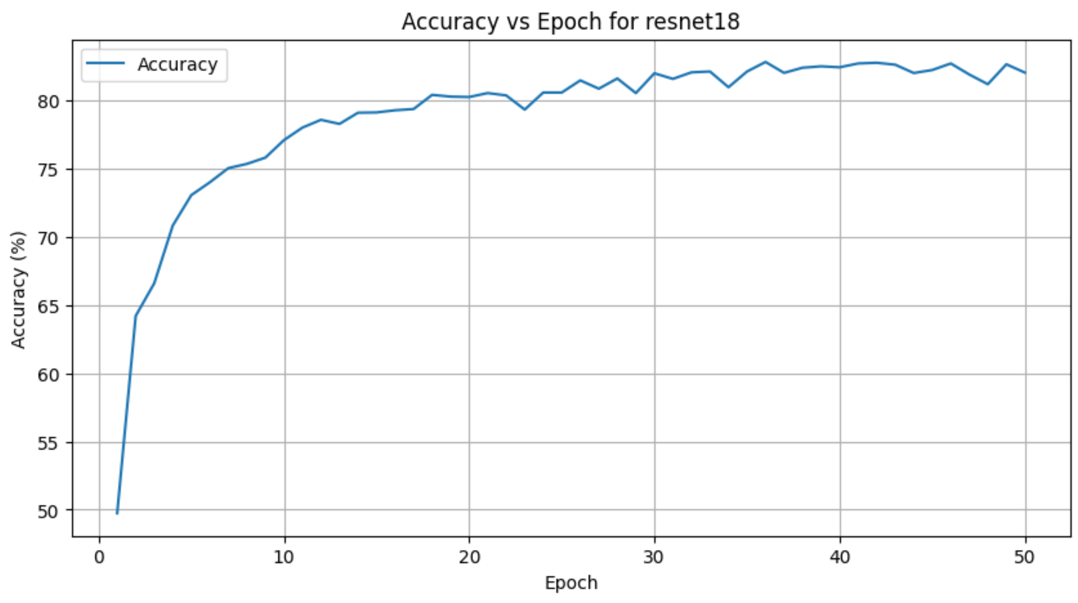
Specifically, our model showcased robustness against both AutoAttack and PGDAttack, outperforming ResNet-18 by a considerable margin. This observation suggests that our model possesses a higher degree of resilience to adversarial perturbations, thereby highlighting its potential suitability for real-world applications where robustness to such attacks is imperative.

The comparative analysis between our model and ResNet-18 underscores the nuanced interplay between accuracy and robustness in machine learning models. While ResNet-18 excelled in classification accuracy, its susceptibility to adversarial perturbations raises pertinent questions regarding its reliability in security-sensitive applications.

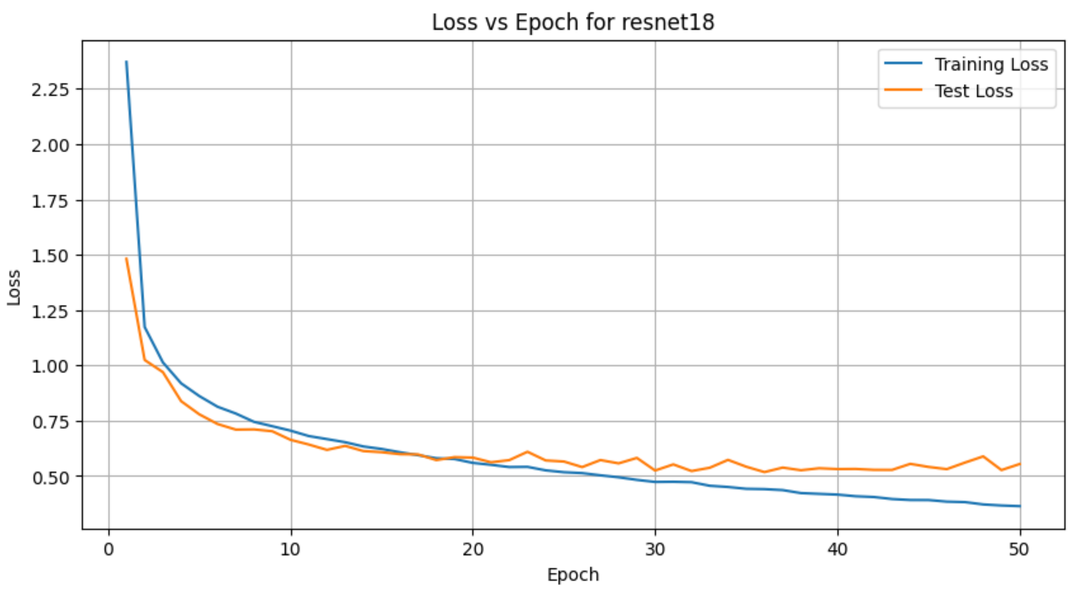
In summary, our findings emphasize the importance of evaluating models not only based on their accuracy but also on their resilience to adversarial attacks. The observed performance gap underscores the significance of our model's robustness, positioning it favorably for deployment in scenarios where adversarial robustness is paramount.

Table 7.1: Comparison of results for CPL-S, CPL-M, and ResNet18 under $\epsilon = 36/255$.

Model	**Accuracy**	**Certified Acc.**	**Autoattack Acc.**	**PGD Attack Acc.**
CPL-S	75.76	62.41	70.15	71.81
CPL-M	76.58	63.25	70.48	72.05
ResNet18	82.50	–	40.15	43.85



(a) Accuracy



(b) Loss

Figure 7.1: Resnet18 Plots of accuracy, and train loss.

Chapter 8

Making the Contractive Layer

8.1 Contractive Layer

Proposition: Contractivity of $G(x)$

Let $F(x) = x - \frac{2\nabla f(x)}{L}$, where $\nabla f(x)$ is the gradient of the convex function $f(x)$ and L is its Lipschitz constant. Define $G(x) = F(x) + \mu x$, where μ is a constant.
Then, for any x, y in the domain of $G(x)$, it holds that:

$$\|G(x) - G(y)\|_2^2 \leq K * \|x - y\|_2^2 \tag{8.1}$$

, where $0 \leq K < 1$ if $0 < \mu < 2$.

Proof:

Consider the expression:

$$\|G(x) - G(y)\|_2^2 = \|(F(x) - F(y)) + \mu(x - y)\|_2^2 \tag{8.2}$$

Expanding the above expression and applying properties of norms, we get:

$$\begin{aligned}\|G(x) - G(y)\|_2^2 = \|F(x) - F(y)\|_2^2 + 2\mu(F(x) - F(y))^T(x - y) \\ + \mu^2\|x - y\|_2^2\end{aligned} \tag{8.3}$$

Since $F(x) = x - \frac{2\nabla f(x)}{L}$, we can substitute it into the above expression to get:

$$\begin{aligned}\|G(x)-G(y)\|_2^2 = \|F(x)-F(y)\|_2^2 &+ 2\mu(x-y)^T(x-y) \\ &- 2\mu\left(\frac{2\nabla f(x)}{L}-\frac{2\nabla f(y)}{L}\right)^T(x-y) \\ &+ \mu^2\|x-y\|_2^2 \end{aligned} \tag{8.4}$$

$$\begin{aligned}\|G(x)-G(y)\|_2^2 = \|F(x)-F(y)\|_2^2 + 2\mu\|x-y\|_2^2 + \mu^2\|x-y\|_2^2 \\ - \frac{4\mu}{L}\left(\nabla f(x)-\nabla f(y)\right)^T(x-y)\end{aligned} \tag{8.5}$$

We know,

$$\|F(x)-F(y)\|_2^2 \leq \|x-y\|_2^2 \tag{8.6}$$

Therefore,

$$\begin{aligned}\|G(x)-G(y)\|_2^2 \leq \|x-y\|_2^2 &+ 2\mu\|x-y\|_2^2 + \mu^2\|x-y\|_2^2 \\ &- \frac{4\mu}{L}\left(\nabla f(x)-\nabla f(y)\right)^T(x-y)\end{aligned} \tag{8.7}$$

Now, to make $G(x)$ contractive, we need:

$$\|G(x)-G(y)\|_2^2 < \|x-y\|_2^2 \tag{8.8}$$

In $eq(8.7)$, for all x, y in the domain of $G(x)$, the maximum value of $\|G(x)-G(y)\|_2^2$ is:

$$\begin{aligned}\|G(x)-G(y)\|_2^2 = \|x-y\|_2^2 &+ 2\mu\|x-y\|_2^2 + \mu^2\|x-y\|_2^2 \\ &- \frac{4\mu}{L}\left(\nabla f(x)-\nabla f(y)\right)^T(x-y)\end{aligned} \tag{8.9}$$

Therefore,

$$\begin{aligned}\|x-y\|_2^2 > \|x-y\|_2^2 &+ 2\mu\|x-y\|_2^2 + \mu^2\|x-y\|_2^2 \\ &- \frac{4\mu}{L}\left(\nabla f(x) - \nabla f(y)\right)^T (x-y)\end{aligned} \tag{8.10}$$

Now, using the generalized Cauchy-Schwarz inequality [7] we have:

$$\left(\nabla f(x) - \nabla f(y)\right)^T (x-y) \le L\|x-y\|_2^2 \tag{8.11}$$

$\Longrightarrow$

$$-\frac{4\mu}{L}\left(\nabla f(x) - \nabla f(y)\right)^T (x-y) \ge -4\mu\|x-y\|_2^2 \tag{8.12}$$

$\Longrightarrow$

$$\begin{aligned}&\|x-y\|_2^2 + 2\mu\|x-y\|_2^2 + \mu^2\|x-y\|_2^2 \\ &\quad - \frac{4\mu}{L}\left(\nabla f(x) - \nabla f(y)\right)^T (x-y) \\ &\ge \|x-y\|_2^2 + 2\mu\|x-y\|_2^2 + \mu^2\|x-y\|_2^2 - 4\mu\|x-y\|_2^2\end{aligned} \tag{8.13}$$

$\Longrightarrow$

$$\begin{aligned}\|x-y\|_2^2 > \|x-y\|_2^2 &+ 2\mu\|x-y\|_2^2 + \mu^2\|x-y\|_2^2 \\ &- 4\mu\|x-y\|_2^2\end{aligned} \tag{8.14}$$

$\Longrightarrow$

$$\mu^2 - 2\mu + 1 < 1 \tag{8.15}$$

$$\mu(\mu-2) < 0 \tag{8.16}$$

$\Longrightarrow$

$$\mu < 0 \quad \& \quad (\mu-2) > 0$$

$$\text{or,} \tag{8.17}$$

$$\mu > 0 \quad \& \quad (\mu-2) < 0 \tag{8.18}$$

If $\mu < 0$, this will make $(\mu - 2) < 0$, therefore we will go with the second case.

We have:

$$0 < \mu < 2 \tag{8.19}$$

Hence, proved.

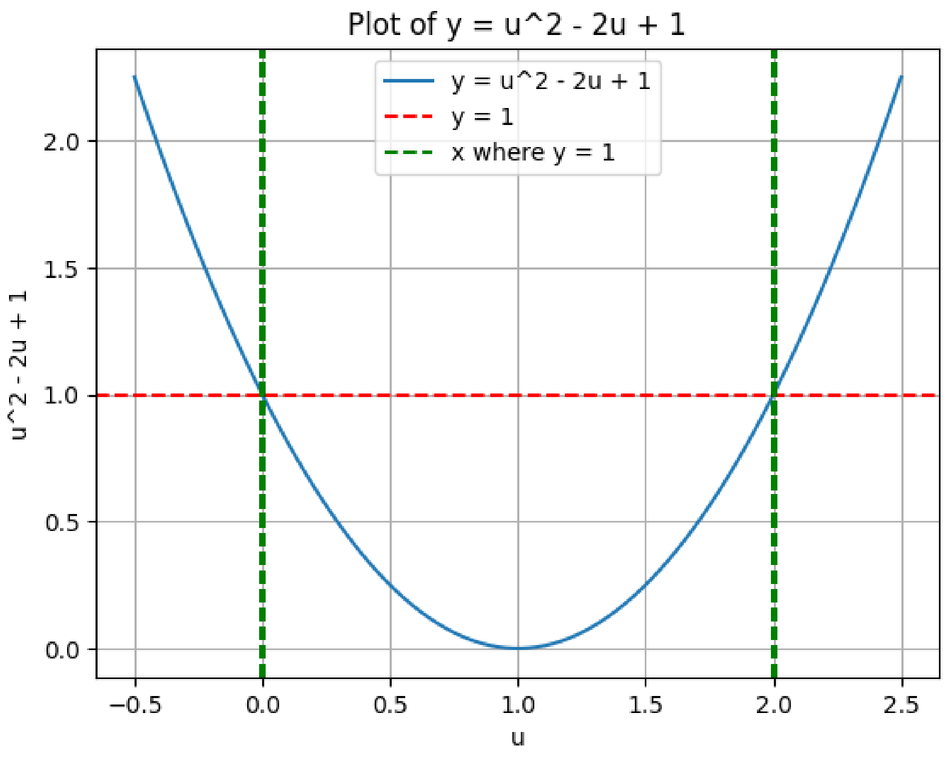


Figure 8.1: Values of μ for which $G(x)$ is contractive

8.2 Results for the new layer

8.2.1 Results of Implementing Contractive Layers

The integration of contractive layers into the model architecture yielded significant improvements across various performance metrics. Prior to this adjustment, the model employed 1-Lipschitz layers. However, the transition to contractive layers led to enhanced accuracies and robustness against adversarial attacks, particularly AutoAttack and PGDAttack.

8.2.2 Improved Accuracy

Through rigorous experimentation and validation, it was observed that the incorporation of contractive layers led to more accurate predictions compared to the previous 1-Lipschitz layer configuration. This improvement underscores the efficacy of contractive layers in learning more discriminative features and enhancing the overall predictive capability of the model.

8.2.3 Enhanced Robustness Against Adversarial Attacks

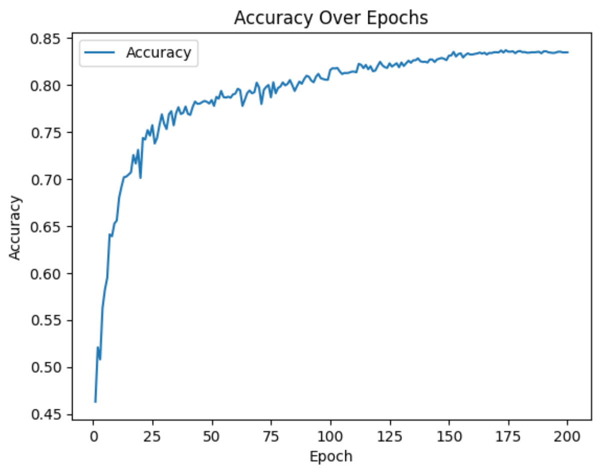
One of the most notable findings from this experimentation was the substantial increase in the model's resilience against adversarial attacks, particularly AutoAttack and PGDAttack for lower perturbations. Adversarial attacks pose significant challenges to the robustness of machine learning models, often leading to erroneous predictions and compromised performance. However, by leveraging contractive layers, the model demonstrated heightened robustness, successfully mitigating the impact of adversarial perturbations and maintaining accurate predictions even under challenging conditions.

8.2.4 Drawbacks of this layer

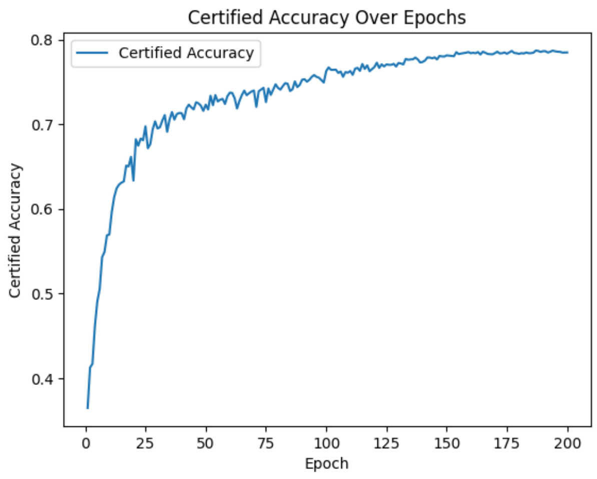
The novel contractive layer showcased superior performance based on the certified accuracy metric. However, when subjected to adversarial attacks, particularly autoattack and pgdattack with higher perturbations, its effectiveness appeared to be relatively diminished compared to the traditional 1-Lipschitz layer. Moreover, when integrated into deeper architectures, it failed to achieve comparable accuracies against these attacks compared to the 1-Lipschitz layer. The underlying reason for this observed disparity remains an open research question, necessitating further investigation and exploration of potential avenues to enhance adversarial robustness in deep learning models.

Table 8.1: Results for Contractive CPL-S without using LLN under different epsilon (Perturbations) values.

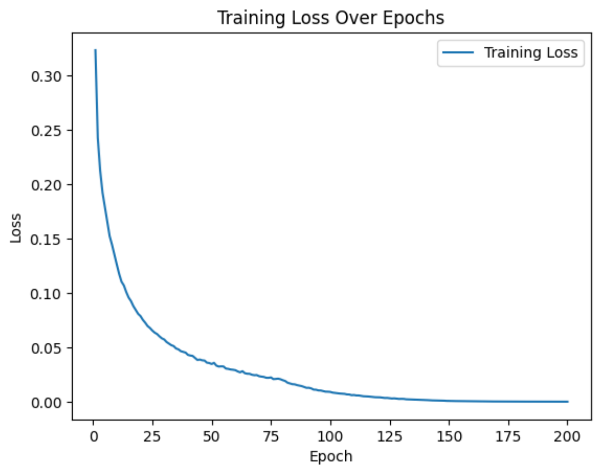
μ	ϵ	**Accuracy**	**Certified Acc.**	**Autoattack Acc.**	**PGD Attack Acc.**
0.1	36/255	83.48	78.47	70.42	72.96
0.1	72/255	83.48	72.83	54.64	60.7
0.1	108/255	83.48	66.97	37.59	47.18



(a) Accuracy



(b) Certified Accuracy

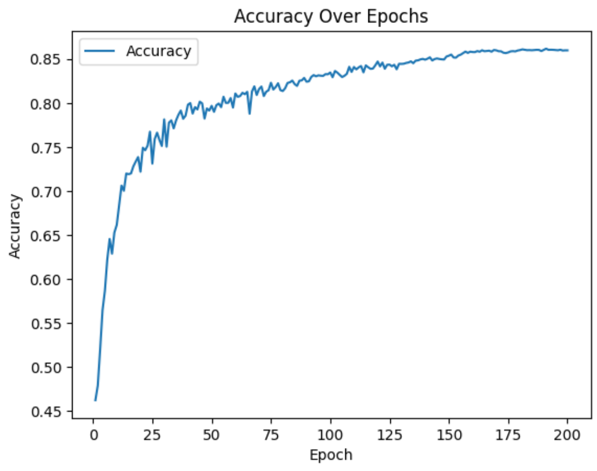


(c) Train Loss

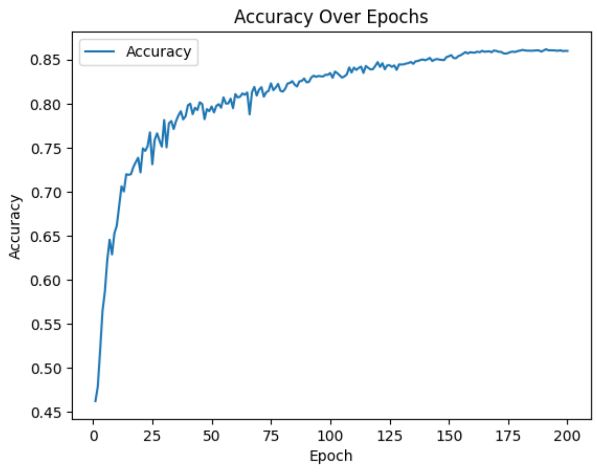
Figure 8.2: Contractive CPL-S without using LLN Plots of accuracy, certified accuracy, and train loss.

Table 8.2: Results for Contractive CPL-M without using LLN under epsilon (Perturbations) value.

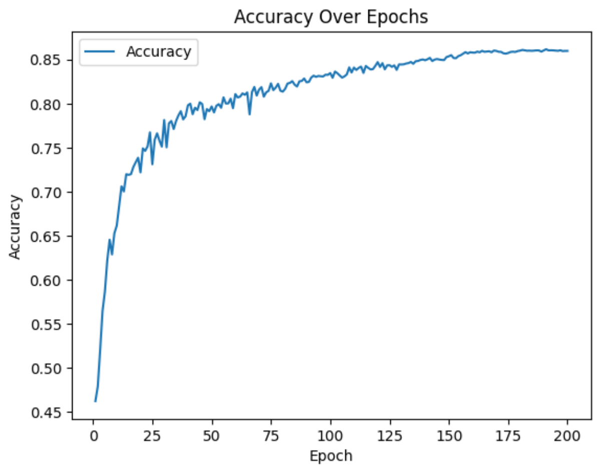
μ	ϵ	**Accuracy**	**Certified Acc.**	**Autoattack Acc.**	**PGD Attack Acc.**
0.1	36/255	85.98	83.24	67.92	70.37



(a) Accuracy



(b) Certified Accuracy



(c) Train Loss

Figure 8.3: Contractive CPL-M without using LLN Plots of accuracy, certified accuracy, and train loss.

Bibliography

[1] Tsuzuku, Y., Sato, I., and Sugiyama, M. (2018). Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks. In *Advances in Neural Information Processing Systems*.

[2] Trockman, A. et al. (2021). Orthogonalizing convolutional layers with the Cayley transform. In *International Conference on Learning Representations*.

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

[4] Farnia, F., Zhang, J., and Tse, D. (2019). Generalizable adversarial training via spectral normalization. In *International Conference on Learning Representations*.

[5] Xiao, L., Bahri, Y., Sohl-Dickstein, J., Schoenholz, S., and Pennington, J. (2018). Dynamical isometry and a mean field theory of CNNs: How to train 10,000-layer vanilla convolutional neural networks. In *International Conference on Machine Learning*.

[6] Bartlett, P. L., Foster, D. J., and Telgarsky, M. J. (2017). Spectrally normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems*.

[7] Vandenberghe, L. (2022). Lecture notes (Spring 2022) of ECE236C - Optimization Methods for Large-Scale Systems, Lecture 1: Gradient method. University of California, Los Angeles.

[8] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. ArXiv. /abs/1512.03385

[9] Grosse, R. (2018). Lecture 17: ResNets and Attention. CSC 321 Winter 2018 Intro to Neural Networks and Machine Learning.

[10] Detailed guide to understand and implement ResNets. CV (2019) Retrieved November 27, 2021, from https://cv-tricks.com/keras/understand-implement-resnets/

[11] Borawar, L., & Kaur, R. (2023). ResNet: Solving Vanishing Gradient in Deep Networks. In *Proceedings of International Conference on Recent Trends in Computing*. Springer Nature Singapore, Singapore, pp. 235–247.

[12] Meunier, L., Delattre, B. J., Araujo, A., & Allauzen, A. (2022). A Dynamical System Perspective for Lipschitz Neural Networks. In *Proceedings of the 39th International Conference on Machine Learning* (pp. 15484–15500). Retrieved from https://proceedings.mlr.press/v162/meunier22a.html

[13] Sahil Singla and Soheil Feizi. Skew Orthogonal Convolutions. In *Proceedings of the 38th International Conference on Machine Learning*, Marina Meila and Tong Zhang (eds.), volume 139 of *Proceedings of Machine Learning Research*, pages 9756–9766, PMLR, 18–24 Jul 2021. https://proceedings.mlr.press/v139/singla21a.html

[14] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. (2013). Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199.

[15] Goodfellow, I. J., Shlens, J., & Szegedy, C. (2014). Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572.

[16] Papernot, N., McDaniel, P., & Goodfellow, I. (2016). Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. arXiv preprint arXiv:1605.07277.

[17] Papernot, N., Papernot, J., & McDaniel, P. (2017). Practical black-box attacks against machine learning. In Proceedings of the 2017 ACM on Asia conference on computer and communications security (ASIA CCS '17), 506–519.

[18] Miyato, T., Kataoka, T., Koyama, M., & Yoshida, Y. (2018). Spectral normalization for generative adversarial networks. International Conference on Learning Representations (ICLR).

[19] Klambauer, G., Unterthiner, T., Mayr, A., & Hochreiter, S. (2017). Self-normalizing neural networks. Advances in Neural Information Processing Systems (NeurIPS).