Design and Evaluation of GAN-based Models for Adversarial Training Robustness in Deep Learning

by

Weimin Zhao

A thesis submitted to the
School of Graduate and Postdoctoral Studies in partial
fulfillment of the requirements for the degree of

Master of Applied Science in Electrical and Computer Engineering

Department of Electrical, Computer, and Software Engineering

Faculty of Engineering and Applied Science

University of Ontario Institute of Technology (Ontario Tech University)

Oshawa, Ontario, Canada

April 2023

THESIS EXAMINATION INFORMATION

Submitted by: **Weimin Zhao**

Master of Applied Science in **Electrical and Computer Engineering**

Thesis title: Design and Evaluation of GAN-based Models for Adversarial Training Robustness in Deep Learning

An oral defense of this thesis took place on April 3rd, 2023 in front of the following examining committee:

Examining Committee:

Chair of Examining Committee	Dr. Masoud Makrehchi
Research Supervisor	Dr. Qusay H. Mahmoud
Research Co-supervisor	Dr. Sanaa Alwidian
Examining Committee Member	Dr. Akramul Azim
Thesis Examiner	Dr. Khalil El-Khatib, Ontario Tech University

The above committee determined that the thesis is acceptable in form and content and that a satisfactory knowledge of the field covered by the thesis was demonstrated by the candidate during an oral examination. A signed copy of the Certificate of Approval is available from the School of Graduate and Postdoctoral Studies.

ABSTRACT

Design and Evaluation of GAN-based Models for Adversarial Training Robustness in Deep Learning

Weimin Zhao Advisor

Ontario Tech University, 2023 Dr. Qusay H. Mahmoud

Dr. Sanaa Alwidian

Adversarial attacks show one of the generalization issues of current deep learning models on special distribution shifted data. The adversarial samples generated by the attack algorithm can introduce malicious behavior to any deep learning system that affects the consistency of the deep learning model. This thesis presents the design and evaluation of multiple possible component architectures of a GAN that can provide a new direction for training a robust convolution classifier. Each component is related to a different aspect of the GAN that impacts the generalization and the robustness outcomes. The best formulation can achieve around 45% accuracy under 8/255 L_∞ PGD attack and 60% accuracy under 128/255 L_2 PGD attack that outperforms L_2 PGD adversarial training. The other contributions include the research on gradient masking, robustness transferability across the constraints and the generalization limitations.

Keywords: Adversarial attacks; adversarial samples; adversarial robustness; adversarial training; generative adversarial networks

AUTHOR'S DECLARATION

I hereby declare that this thesis consists of original work of which I have authored. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize the University of Ontario Institute of Technology (Ontario Tech University) to lend this thesis to other institutions or individuals for the purpose of scholarly research. I further authorize University of Ontario Institute of Technology (Ontario Tech University) to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research. I understand that my thesis will be made electronically available to the public.

_____ Weimin Zhao

STATEMENT OF CONTRIBUTIONS

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication. I have used standard referencing practices to acknowledge ideas, research techniques, or other materials that belong to others. Furthermore, I hereby certify that I am the sole source of the creative works and/or inventive knowledge described in this thesis.

Results from this thesis research have been disseminated in the following publications:

- W. Zhao, Q. H. Mahmoud, and S. Alwidian, "Evaluation of GAN-based Adversarial Training" Sensors, vol. 23, no. 5, p. 2697, Jan. 2023, doi: 10.3390/s23052697.
- W. Zhao, S. Alwidian, and Q. H. Mahmoud, "Evaluation of GAN Architectures for Adversarial Robustness of Convolution Classifier" The AAAI-23 Workshop on Artificial Intelligence Safety (SafeAI 2023), Washington, DC, Feb 2023.
- W. Zhao, S. Alwidian, and Q. H. Mahmoud, "Adversarial Training Methods for Deep Learning: A Systematic Review," Algorithms, vol. 15, no. 8, p. 283, Aug. 2022, doi: 10.3390/a15080283.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my thesis supervisors, Dr. Qusay H. Mahmoud and Dr. Sanaa Alwidian, for their support, patience, and guidance throughout my graduate studies. They provided me with resourceful guidance and support that allowed me to successfully conduct the research. I also want to thank all my course instructors for their valuable guidance and knowledge during the master's degree. Finally, I also would like to thank my family and Ontario Tech University for giving me good support and a valuable platform to conduct the research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

API	Application Programming Interface
ART	Adversarial Robustness Toolbox
BIM	Basic Iterative Method
CIFAR	Canadian Institute for Advanced Research
eFGSM	Enhanced Fast Gradient Sign Method
FGSM	Fast Gradient Sign Method
GAN	Generative Adversarial Network
IFGSM	Iterative Fast Gradient Sign Method
JSMA	Jacobian-based Saliency Map Attack
KL divergence	Kullback-Leibler divergence
MNIST	Modified National Institute of Standards and Technology
PGD	Projected Gradient Descent
RGB	Red Green Blue
SIM	Single-step epoch-Iterative Method
TRADES	TRadeoff-inspired Adversarial DEfense via Surrogate-loss minimization
VGG	Visual Geometry Group

Chapter 1. Introduction

Deep learning is utilized for various applications, such as image classification, language recognition, signal transformation, and sample generation [1]. Deep learning models have also played an important role in the development of self-driving vehicles, fault detection, and other mission-critical and safety-critical systems. The recent expansion of deep learning models adds more concerns and challenges related to the consistency and security of machine learning models. Deep neural networks are referred to as black-box machine learning algorithms in which large quantities of parameters and complex structures are highly uninterpretable [1]. The behavior of these deep learning models can be impossible to explain or understand by humans.

Adversarial samples expose one of the black-box properties of deep learning models and highlight the limitations of the interpretability of the deep learning system [1], [2]. These samples typically have different representations or meanings when they are interpreted by human learning systems and deep learning systems. Adversarial sample is one of the anomalies that is humanly crafted and targets machine learning or deep learning model themselves. Hence, cyber attackers could exploit these adversarial samples to perform attacks on critical systems that build upon deep learning models to conduct malicious activities. From the researcher's perspective, the existence of adversarial samples means that the current understanding of the deep learning models remains limited. The current research suggests that the adversarial samples represent one type of shifted data distributions, often referred to as *adversarial distribution shift* and deep learning models have failed to generalize on this adversarial distribution shifted data samples [3].

The discovery of the integrity properties of the models are important to resolve future optimization problems.

The algorithm for finding adversarial samples is called *adversarial attacks*. The attack algorithms can be categorized based on their optimization methods, required access information, and constraint types [1]. One of the most efficient methods to generate adversarial samples can involve using a *multi-step gradient descent algorithm* to update the input vectors of the model based on the loss functions of the model [4], [5]. There are also multiple algorithms that work on different objective functions or that are not required to access inner model information [6].

The current research primarily focuses on using detection methods [7], denoise models [8], [9], [10], certified methods [11], and adversarial training [5], [12] to defend adversarial samples or to improve the adversarial robustness of deep learning models. However, most of these methods cannot provide a direct robustness increase with the original classifier. Some of the methods are also limited in terms of effectiveness [13] and scalability [1] and may require additional model deployments.

Adversarial training is one of the defense methods that directly improves the adversarial robustness of the deep learning model [12]. The advantages of adversarial training include the simplicity of facilitating an understanding and implementation of the formulation, and the high effectiveness in enabling the current state-of-the-art adversarial robustness [5], [12]. Adversarial training focuses on data augmentation and trains the model on additional generated data to improve the overall data generalization of the model. Adversarial attack algorithms are typically implemented to generate the augmentation data by computing the adversarial noises from the training data set [5]. Both clean training data

and adversarially-perturbed data are used in the adversarial training process. Thus, deep learning models can generalize the distribution of the data based on clean data samples and adversarially-perturbed samples. It is essential that the implemented attack algorithm is effective. Hence, higher-complexity iterative algorithms are generally used in the training processes to maximize the possibility of finding precise adversarial noises [5]. However, the limitations include high training complexity, across-the-board generalization, and accuracy-robustness trade-offs [12].

Generative models have recently provided a new solution to discover more data samples in a hidden distributional space. Some research has demonstrated that it is possible to implement these generative models to produce more training data or to perform data augmentation to improve model training [14]. In adversarial machine learning settings, the generative model can be used to produce high-quality adversarial samples when the generative model is optimized in attack settings [15].

Generative adversarial networks (GANs) are one of the approaches to realize the generative architecture. In defensive cases, both the generative properties of GANs and the adversarial training properties of GANs can be utilized together to construct an effective framework to formulate a similar schema to adversarial training. The generator of GANs can learn to generate adversarial noises by optimizing with gradient descent or Adam optimizers [15], [16], and the generated adversarial noises can be used to train a classifier to improve the classification robustness on defending adversarial attacks [16].

Compared to conventional adversarial training, GAN architecture can be implemented in a low training complexity formulation that can achieve similar training results for small and medium sized models [16]. In this thesis, the size of models refer to

the deep learning models around or less than ten million parameters. The output layers of the generator are also interchangeable, as they switch the adversarial noise outputs based on the required norm constraint. Hence, the deep learning classifier can be generalized on different constrained adversarial samples without implementing an additional attack algorithm. The model trained with the GANs augmentation method could improve its generalization on a wide range of adversarial samples without significant architecture changes and increased complexity.

The challenge of GAN training is that GANs are limited to generalizing on the gradient of the classifier. The generalization of GANs is heavily affected by the generator design and other hyperparameters. Hence, a low-capability set of generators and parameters cannot consistently estimate the worst-case loss of the classifier, resulting in the low robustness of the overall training results. Therefore, an improved GAN is required to resolve the training complexity issue of the conventional adversarial training and to improve the generalization of GAN methods.

1.1 Motivation

The primary motivation of the proposed architecture is to reformulate conventional adversarial training and to address the limitations of adversarial training regarding the training complexity, flexibility, and data generalization. The secondary motivation is to utilize GAN methods to generate adversarial distribution shifted data points and to formulate the problem within the deep learning model itself to compare the differences between the adversarial samples found by deep learning models and conventional attack algorithms.

Adversarial samples can be obtained by using multiple attack algorithms with different optimization formulations. The most common attacks are *gradient-based attacks,* which backpropagate the gradient of the loss back to the input space to calculate the adversarial noises by using gradient descent [4], [5]. The other method involves using metaheuristic optimization or other optimizers to find adversarial samples [6]. In general, attackers aim to maximize the loss of the target model while minimizing the information loss to humans. Hence, adversarial samples can refer to any intentionally crafted data sample that results in significant differences in predictions between human and machine learning models [1], [3]. This thesis mainly focuses on one particular case of crafting methodology, since it is primarily utilized by gradient-based attack algorithms and is easily accessible within white-box situations or transferred attack situations. This crafting methodology is predicated upon an assumption of an adversarial sample as $x+\boldsymbol{\delta}$, where x is a selected sample from the data set, and δ is a small adversarial perturbation or adversarial noise vector that is added to the sample x while minimizing the human perception loss. The attack model uses gradient descent to generate adversarial perturbation δ based on a sample x and a given deep learning model in one or iterative process. Then the perturbation δ is added to sample x to generate an adversarial sample that has a high possibility of causing the deep learning model to misclassify the sample. Hence, the consistency of the classification of the sample within the data set are the first priority of the proposed defense method. Under this assumption, the adversarial noise vector is constrained by a norm size under a distance metric to be mathematically small to minimize the human perception differences. Hence, the adversarial sample is assumed to be located close to a possible data sample within a selected data set under a distance metric.

This thesis focuses mainly on adversarial attack defenses against deep learning image classifiers. In particular, deep convolution image classifiers. The research uses convolution classifier model as a baseline deep learning model to evaluate the effectiveness and the performance of the proposed defensive methodology because these models are well-studied in the related research field [3]. Furthermore, the GAN models are commonly used for image synthesis [15]; hence, the thesis capitalized on this model to augment image data that is well suited for convolution image classification.

In image classification, L_0, L_2, and L_∞ norm metrics are commonly used to minimize the adversarial vector size. The L_0 norm constraint limits the upper pixel number that can be changed, whereas the L_2 norm constraint limits the adversarial vector size in Euclidean space. The L_∞ constraint applies an upper bound to the maximum value change across all the pixels. This thesis explores the use of GANs to improve the adversarial robustness of a deep learning convolution classifier under the assumption of the adversarial sample being $x+\delta$. The primary constraints considered in this work are L_∞ and L_2 constraint adversarial samples. The L_0 constraint is more challenging to formulate with differentiable gradient optimization methods, and it requires more research to implement in the GAN architecture.

1.2 Research Scope

The scope of this research primarily encompasses a small to a medium-size deep learning image classifier that categorizes any user input image in (Red Green Blue) RGB channels according to a specified set of label classes. The size of the classifier depends on the parameter numbers of the model, and the main focus of this thesis is the classifier model with less than ten million parameters. The target image classifier primarily trains with a supervised learning technique to generalize on every image-label pair with training set data

and to provide an accurate classification of any given same-dimensional images. Ideally, this training architecture can be applied to any deep learning image classifier with supervised training. However, it is particularly beneficial for small and medium image classifiers to reduce adversarial training time and improve adversarial robustness. This research addresses adversarial attacks within L_∞ and L_2 norm constraints.

The primary attack algorithms considered in this thesis are the single and the multi-step white-box gradient attacks, which are the attack algorithms that exploit the loss gradient of the model to generate adversarial perturbation. The scenario entails a situation in which attackers gain access to the information of the target model or when the attackers use a transfer-based gradient attack to generate adversarial samples. The primary focus of the solution is to resolve the problem when the correct classification is affected by an attack algorithm that is converted into a false classification.

1.3 Contributions

The main *contribution* of this thesis is an improved GAN-based architecture to augment the training of deep learning image classifiers' adversarial robustness against adversarial attacks under L_∞ and L_2 norm constraints. The architecture uses a deep neural network generator to produce adversarial perturbation solutions for a target classifier for data augmentation. A target classifier can be adversarially-trained with the generator to learn to generalize on such adversarial noise that can provide general robustness against adversarial attacks under the same constraints. In the process of training, the generator captures the loss gradient of the classifier and learns to determine the vulnerability of the target classifier in its loss landscape. The generated noises can be used for later study. The architecture is used on a modified Visual Geometry Group (VGG) classifier with the 10 classes of data

samples from the Modified National Institute of Standards and Technology database (MNIST) and the Canadian Institute for Advanced Research (CIFAR). The following research contributions are presented:

- A GAN architecture that realizes data augmentation for training robust deep learning image classifiers against gradient-based adversarial attacks under L_∞ and L_2 constraints.
- Evaluations of different implementations of the GAN to enable a discussion of the effectiveness of each implementation.
- A GAN adversarial training methodology that facilitates low-complexity adversarial training and achieves improved accuracies against gradient-based attacks. The best accuracies are around 45% against 8/255 L_∞ projected gradient descent adversarial samples and 60% against 128/255 L_2 projected gradient descent adversarial samples.
- An attack-independent adversarial training architecture to provide flexible robustness against two selected norm constraints (L_∞ and L_2) without significant modification requirements.
- A demonstration of the performance of GAN regarding the transferability between constraints and a visualization of the generated adversarial samples.

The research results from this thesis have been disseminated in the papers [17]-[19].

1.4 Thesis Outline

The remainder of the thesis is organized as follows.

Chapter 2 introduces the main concepts and the research related to adversarial attacks and adversarial training. The chapter also introduces the concept of the generative adversarial network (GAN) and its relationship to adversarial training for adversarial attack defenses.

Chapter 3 provides a detailed formulation of the proposed defensive strategy against gradient-based adversarial attacks. The chapter introduces the basic formulation and several detailed components of the proposed model that are used for evaluating the models' performances.

Chapter 4 presents the implementation tools and libraries. The chapter also describes the realization of the model's components and the implementation of the training logic. Furthermore, the chapter elucidates the experimental settings for evaluating the attack algorithm used and presents the datasets utilized for the experiments. The model parameters used for each dataset and experiment are also discussed in this chapter.

Chapter 5 discusses the experimental evaluation and results. The chapter is divided into two main sections for the different datasets. The more advanced dataset is divided into subsections. The experiments related to different model components are reported in the corresponding subsections.

Finally, **Chapter 6** presents the conclusion and the potential future direction of this research.

1.5 Summary

This chapter includes a brief discussion of the current challenges regarding the adversarial samples in a deep learning context. It also presents an outline and the scope of the thesis. The primary goal of the thesis is to implement a novel GAN architecture to generate

augmentation adversarial samples and to provide adversarial training to a given image classifier to improve its generalization on adversarial samples. Chapter 2 discusses the related works and introduces other important background information regarding adversarial machine learning and adversarial training.

Chapter 2. Background and Related Work

This chapter surveys the related research regarding adversarial samples, adversarial attacks, and adversarial training algorithms. The concepts related to gradient-based adversarial attack algorithms are presented, and the state-of-the-art adversarial training frameworks are reviewed in detail. The other defensive strategies are not closely related to the approach in this thesis; hence, they are not reviewed. The gradient-based attacks are categorized based on their computational cost and the constraint types. The adversarial training is categorized based on the components used for data augmentation, and more specifically, the attack algorithms used to generate adversarial samples. The major contents of this chapter have been published as a systematic literature review [17].

2.1 Background

Deep learning is one of the machine learning methodologies that utilizes deep-stacked neuron networks to realize function transformation and feature extraction. Recent advances within deep learning have been applied to a wide range of applications, including image recognition, language recognition, generative models, and other domains. To realize these domain applications, multiple architectural designs have been proposed, including convolution networks, recurrent networks, transformers, and many others [20]. Within image recognition models, the convolution neuron network is one of the commonly used architectures that has realized state-of-the-art performance in image classification and image synthesis. Furthermore, the generative adversarial network (GAN) is one of the generative model architectures that can utilize convolution neuron networks for image generation and synthesis. The image synthesis in GAN typically combines the components of convolution neuron networks and the knowledge of image classification and data

synthesis to realize its functionality. This thesis focuses on issues related to the deep learning convolution classifier and the utilization of GAN to address the vulnerability of these types of classifiers. Hence, this section provides basic information relating to GAN and the related attack algorithms for convolution image classifiers. The overview of the relationship between machine learning, deep learning, and GAN is illustrated in Figure 2.1.

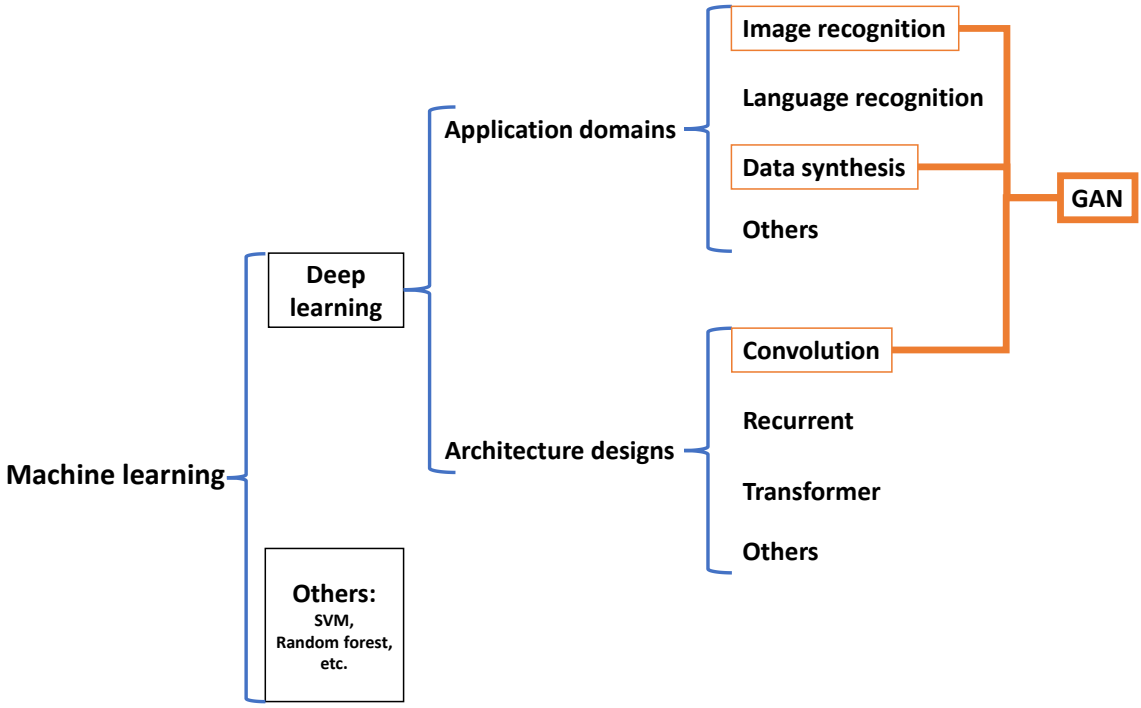

Figure 2.1: Relationship between machine learning, deep learning, and GAN.

2.1.1 Generative Adversarial Network

A generative Adversarial Network (GAN) is a specific architecture that incorporates a generator model and a discriminator to form an adversarial training architecture [21]. The architecture realizes a self-supervised learning formation through the min-max optimization of two network models. The generator is typically responsible for learning to generate high-quality synthetic data samples, and the discriminator is used to differentiate the generated synthesis data samples and the real data samples and to guide the optimization process of the generator. A standard architecture of GAN is illustrated in

Figure 2.2. In image synthesis tasks, a convolution GAN is commonly utilized, where the generator and the discriminator are both implemented with convolution neuron networks. The generator usually learns to transfer an input vector to the synthetic images and the discriminator, as a binary classifier outputs a label to indicate whether its input samples are real or fake. The overall GAN formulation can be described as min-max optimization between the generator and discriminator networks. The objective function can be described using Equation (2.1):

$$min_G max_D \, \mathbb{E}_{x\sim p(x)}[\log D(x)] + \mathbb{E}_{z\sim q(z)}[\log (1 - D(G(z)))] \qquad (2.1)$$

where discriminator *D* maximizes the probability to distinguish the real and synthetic data, and generator *G* minimizes the discriminator's probability of it. The term *x* and *z* represent the real sample and the latent vector, respectively. Both networks can be optimized using gradient descent or a more advanced optimizer, such as Adam. The recent application of GAN has entailed generating augmentation data for training other learning models [14], [16].

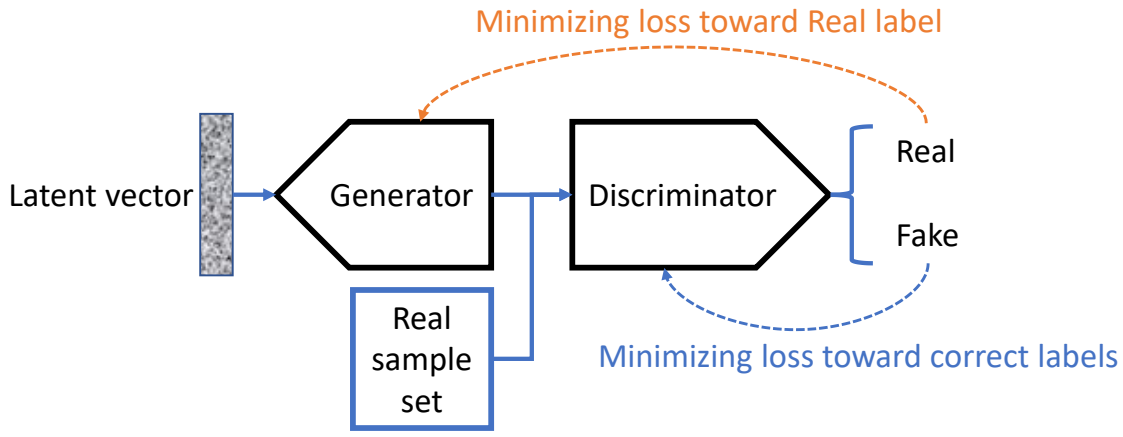

Figure 2.2: A GAN architecture.

2.1.2 Adversarial Attacks on Image Classifiers

An adversarial sample of an image classifier is a type of image sample that is classified differently by the model classifier and a human [1]. In adversarial machine learning studies, the common example is typically a sample that exists close to a selected evaluation sample that is indistinguishable by humans but that has a significantly different model classification compared to the selected evaluation sample. Several distance metrics are used to measure the adversarial distortion size in a mathematical manner. Generally, it is assumed that a small mathematical distance measurement is sufficient to make the distortion indistinguishable from human visions. Hence, the general assumption can be written as the Equation (2.2) for an adversarial sample generated from an evaluation sample *x*:

$$x_{adv} = x + \delta \tag{2.2}$$

where δ is the adversarial perturbation vector or adversarial distortion noise, and x_{adv} is an adversarial sample generated from the sample *x*. The common distance metrics used to constrain the mathematical norm size of δ include L_0, L_2, and L_∞ norm constraints. The formulation of the L_p norm is written as Equation (2.3):

$$\big||x|\big|p = \sum_{i=1}^{n} |x|^p \; p \geq 1 \tag{2.3}$$

where *x* is a given vector, and *p* is a real number greater than 1. When *p* equals zero, the L_0 norm of a vector is the maximum value of the vector that is not equal to zero. When p equals 2, Equation (2.3) is the Euclidean length of the vector, and when p equals infinity, Equation (2.3) is the maximum value within the vector *x*.

An adversarial attack that targets a visual classifier could implement any of the norm constraints to generate an adversarial sample. However, the adversarial sample generated from different constraints has a different distribution. Since the L_0 norm size computation is not differentiable, L_0 constraint adversarial samples are typically generated from black-box algorithms that use metaheuristic optimizers to determine the effective adversarial distortion [3]. These types of algorithms are more flexible in their optimization methods and can create adversarial samples without any differentiable formulation. Within the scope of this thesis, gradient-based attacks are the primary focus; hence, the L_2 and L_∞ norms are mostly considered in this work.

2.1.3 Gradient Descent

Gradient Descent is one of the optimization algorithms that is widely used in deep learning optimizations [3], [20], including training, finetuning, and adversarial attacks. The algorithm of gradient descent can be described using Equation (2.4) [20]:

$$\theta_{n+1} = \theta_n - \alpha \nabla L(\theta_n) \tag{2.4}$$

where the θ_{n+1} and θ_n represent the optimizing parameter of a differentiable model, and L represents the loss function of the optimization. The gradient descent algorithm computes the gradient of the loss function ∇L regarding the parameter θ_n and multiplies the gradient with a learning rate parameter α to determine the value update on the parameter θ_n. θ_{n+1} is the updated parameter after gradient descent.

In the case of the training and finetuning process of a deep learning model, the gradient descent usually pairs with backpropagation to update the model parameters within neural network layers [20]. The training algorithm of a deep learning model requires the

definition of a loss function L and a set of sample-label pairs of training data (x_i, y_i). The gradient of the loss function regarding the inner parameters of the model is computed from the errors of the model outputs based on training sample-label pairs. The error values of the outputs can be backpropagated to the shallower layers of the deep learning model to obtain the gradient of the loss with respect to all the parameters. Hence, all the parameters of the model can be optimized using the gradient descent algorithm.

In the case of adversarial attacks, the algorithms using gradient descent to optimize the adversarial samples are referred to as gradient-based attacks [3], [5]. Assuming a deep learning model with sample-label pairs (x_i, y_i) and loss function L, the gradient-based attack optimizing sample x_i based on the gradient of the loss function regarding each sample-label pair (x_i, y_i). The details of the gradient-based attack algorithms will be introduced in subsequent sections.

In this thesis, gradient descent is used as the primary optimization algorithm. The implementation of gradient descent is included in all the model optimization and attacks from the thesis, such as GAN training process, classifier training process, and all gradient-based attack algorithms.

2.1.4 Fast Gradient Method Adversarial Attacks

Fast gradient method attacks are one of the gradient-based attack algorithms that can target convolution classifier models. These attacks use the backpropagated gradient information of a target model to compute adversarial noise. Hence, this requires one gradient backpropagation to perform the attack and necessitates access to a selected target model's inner parameter. When the attack operates in the L_∞ norm constraint, the sign of the loss

gradient is used to keep the signed direction of the gradient for constraining the L_∞ norm. This version of the attack is called the fast gradient sign method (FGSM) [4]. The FGSM attack corresponds to Equation (2.5):

$$x' = x + \varepsilon \cdot sign\big(\nabla x\, L(x, y)\big) \tag{2.5}$$

where *x'* is the adversarial sample of an original sample *x*. The $\varepsilon \cdot sign(\nabla x\, L(x,y))$ is the computational vector or adversarial perturbation vector. The $\nabla x\, L(x,y)$ represents the loss gradient of the sample-label pair (*x*,*y*) from the target classifier, and the sign function constrains the gradient vector into the L_∞ norm. The parameter ε is a scalar parameter that limits the maximum norm size of the perturbation vector. The method can also be used in the L_2 norm by simply removing the sign function.

2.1.5 Iterative Gradient Descent Methods

The *iterative gradient descent method* [1], [3], [5] includes multiple gradient-based attack algorithms that utilize multi-step gradient descent and backpropagation to generate adversarial samples; the algorithm is more effective in attacking convolution classifiers. Similar to FGSM, these methods require the model's inner parameters to perform a successful adversarial sample generation.

Projected gradient descent (PGD) [5] is a common optimization algorithm that uses gradient descent as a core optimization strategy. In adversarial machine learning, PGD is an effective iterative gradient descent attack method that incorporates vector projection and random initialization. The selected data sample is added by a random perturbation before the start of the gradient descent, and projection is performed when the solution runs out of feasible space. The basic formulation is written in Equation (2.6):

$$x_{t+1} = \prod_{x+s}(x' + \alpha\, sign(\nabla x\, L(x, y))) \quad (2.6)$$

Compared to the FGSM formula, the PGD includes the iterative addition of adversarial perturbation and projection. The x_{t+1} represents the current result of an adversarial sample, where the x' is the previous iteration's adversarial sample. The overall sample vector is projected to the feasible space to compute the current sample x_{t+1}. Similar to FGSM, the sign function transforms the perturbation into the L_∞ constraint. The formulation can also be implemented in the L_2 constraint by removing the sign function from the formula.

2.2 Related Work

This section provides details about the state-of-the-art adversarial training methods that are used to improve the adversarial robustness against gradient-based adversarial attack algorithms. The general formulation can be described using Equation (2.7):

$$\min \sum \max L(f(G)), y_i) \quad (2.7)$$

The algorithm trains the classifier model to generalize on the adversarial samples produced by a selected adversarial attack. The formulation uses the adversarial attack to generate adversarial samples for maximizing the loss of the target classifier. The target classifier then trains with these samples to minimize the loss to provide generalization on the adversarial samples. The selected attack algorithm determines the type of robustness of the trained classifier model. The quality of the generated adversarial sample impacts the robustness level of the trained model. Hence, the adversarial sample generation algorithm should be carefully selected for adversarial training to provide a proper type of robustness, and generalization depends upon scenarios of potential threat.

Table 2.1 lists the adversarial sample generation algorithms (i.e., the attack algorithms) used in adversarial training. Each type of augmentation algorithm is systematically reviewed, and the advantages and limitations are listed and discussed.

Table 2.1: Overview of adversarial training.

Category	**Related works**	**Description**	**Advantage**	**Limitation**
Gradient-based single step algorithm	[22]-[41]	FGSM/eFGSM/SIM	Efficient and fast with low training complexity compared to iterative methods	Low precision; causes overfitting and provides poor generalization of robustness
Gradient-based multi-step algorithm	[5], [22], [31]-[34], [36], [42]-[73]	PGD/IFGSM/BIM/JSMA	High precision and accurate adversarial sample generation; can provide more robustness compared to single-step algorithms	High complexity; overfitting; poor generalization on clean data samples
Generative models	[16], [62], [74]-[83]	Auto-encode-decoder and GANs	Semi-supervised or self-supervised learning; more efficient for implementation and for training complexity compared to multi-step methods	Performance varies from different generative models; low transferability and catastrophic forgetting and overfitting
Ensemble methods	[22], [84]-[86]	Combining ensemble models	Lower the transferability of adversarial samples across different models with	Require pretrained models; provide less robustness to target model itself

			efficient training time	
Our proposed method	N/A	Contributions: • Improving upon GAN model and utilizing it for adversarial training • Providing more advanced formulations to enhance GAN's augmentation ability to train a robust convolution classifier • Providing low-complexity training methodology • Constructing an attack-independent adversarial training method with L_∞ and L_2 constraints and providing insight into the transferability of robustness between the constraints • Providing insight into the overfitting problem of GAN and discovering the effect of training epochs The limitation of the proposed method: • Overfitting of generator and classifier • Accuracy and robustness tradeoffs		

2.2.1 Gradient-Based Single-Step Algorithm

Adversarial training frequently uses gradient backpropagation to generate adversarial samples for data augmentation. This category of the gradient-based method incorporates one-time computation to obtain the adversarial vector that is used to modify the original data. The basic idea concept incorporates the gradient descent algorithm to optimize the loss gradient onto the input data. The inner part of the adversarial training maximization is generally implemented by the FGSM algorithm or by a similar algorithm for generating augmentation data. The benefit of this single-step generation is that only one additional gradient descent is required for each training step. Hence, the overall training complexity is not heavily affected. This type of adversarial training can provide baseline robustness for the trained model to defend against similar single-step gradient descent attacks. However, single-step gradient attacks are known to be ineffective in finding the worst-case

adversarial sample [5]. This means that the model trained with a single-step attack algorithm cannot guarantee robustness against the worst-case adversarial samples [5]. This ineffectiveness can also be applied to smaller models. Hence, in most cases, single-step adversarial training is less practical, since any attacker can use multiple gradient descents to generate adversarial samples. Conversely, the model trained with a single-step attack still has adversarial samples that exist in false decision boundaries; therefore, the model cannot be trusted for capturing robust features of the data. The other problems include overfitting [28] and accuracy-robustness trade-offs [50].

Several studies have targeted the improvement of this type of adversarial training to provide an enhanced robustness generalization and retain the advantages associated with low computational complexity. The improved versions include the implementation of dropout scheduling to reduce overfitting [23] and optimization regularization to provide a better estimate of adversarial direction [24]-[41]. Fast adversarial training [26], [27], [31] is one of the most well-known efficient adversarial training concepts that relates to implementing adversarial training with a single-step gradient attack. This method uses random initialization incorporated with a single-step attack to generate a similar adversarial sample compared to other multi-step attacks. However, these improvements are also reported to have problems with overfitting [31], and they still do not promote the accuracy against adversarial samples compared to the current state-of-the-art accuracy with regular classification tasks.

2.2.2 Gradient-Based Multi-Step Algorithm

The adversarial training that uses a multi-step gradient attack has been developed from single-step attack adversarial training. Rather than using simple one-step backpropagation,

this type of adversarial training uses multiple backpropagation gradient descents to generate adversarial samples in each training step [5]. The commonly implemented multi-step attacks are variations of PGD or basic iterative method (BIM) attacks. With the repetitive gradient descent, the loss gradient of the model can be accurately explored to produce more precise and effective adversarial samples [5]. Hence, the model trained with this augmented data can yield enhanced overall robustness compared to single-step algorithms [5]. However, the disadvantage of the multi-step algorithm is that it includes another iterative loop within each training step; hence, it can cause heavy computational overhead, particularly with a larger model [12]. This increased complexity can impact the practical efficiency of training a robust model. Furthermore, the robustness-accuracy trade-offs are also presented in the trained model with these attacks [50]. The other problem also involves overfitting [12], which demonstrates the limitation of the generalization. The other challenge related to the usage of PGD attacks is that it is typically more effective in generating L infinity norm adversarial samples [5]. Hence, robustness against other norm-constraint adversarial samples cannot be guaranteed.

PGD adversarial training [5] is one of the most popular adversarial training methods. Hence, various studies have addressed the prospect of improving PGD adversarial training. The general improvement direction encompasses reducing overfit, reducing training complexity, improving attack effectiveness, progressing toward better training objectives, and developing solutions to data limitations. The research related to these improvements is presented in Table 2.2, with a summary of the improved direction.

Table 2.2: PGD adversarial training subcategories.

PGD AT improved versions	**Related works**	**Improvements**

Curriculum training	[42], [43], [44]	Reducing overfit and improving generalization
Adaptive training	[45], [46], [47]	Reducing overfit and improving generalization
Efficient training	[48], [49], [31]	Reducing training complexity and training time
Improved regularization	[50], [51], [52], [53], [54]	Improving loss functions and improving generalization
Unsupervised training	[55], [56], [57], [58]	Providing more data and improving generalization

Curriculum adversarial training primarily resolves issues of overfitting. It incorporates an accuracy monitoring step and adjusts the attack strength based on the current performance of the model [42]. The entire process of curriculum training involves a gradually increasing norm size of the adversarial sample; there is also a version of curriculum adversarial training with an early stop [43], [44].

Adaptive adversarial training applies different attack norm sizes for different data samples within the training data set [45]-[47]. The adaptive training prevents the adversarial attack augmentation from overshooting the decision boundary around the training data and applies an adjustable adversarial perturbation based on the distance between the data point and the decision boundary.

Efficient training [31], [48], [49] is another improvement associated with PGD training that aims to reduce the training complexity of PGD training. Free adversarial training [48] is one of the proposals that could improve the gradient efficiency of the training and modify the training schemas to include model parameter updates and the single gradient descent adversarial sample generation within a one-step update. Hence, the training step and the single iteration of the adversarial attack are shared in one iteration to

improve the efficiency of training. An enhancement of this training has been proposed that uses a layer-wised heuristic learning method to further improve the efficiency [49]. Other methods [31] combine the single-step attack and multi-step attack for different stages of the training to accelerate the adversarial training.

There are various methods, including the modification of objective functions and optimization terms. These methods [50]-[54] typically modify or include a regularization function for better training performance. The TRadeoff-inspired Adversarial DEfense via Surrogate-loss minimization (TRADES) regularization [50] is one of the methods that proposes a theoretical optimization trade-off between accuracy and robustness and that applies the trade-off as an objective in a regularization function. The training using this regularization term has a new training objective to reach a balance point between natural errors and adversarial errors. The other regularization proposals include logit pairing [51], sample correctness regularization [52], and triplet loss [53], [54].

Unsupervised and semi-unsupervised adversarial training have been proposed to leverage the data-hungry problem of adversarial training. Zhai et al. [57] suggest that more data is necessary to achieve robust results. Hence, some research [55]-[58] has proposed adversarial training with unlabeled data samples.

Aside from these categories of PGD adversarial training, several other proposals have been provided with other benefits regarding robustness. Maini et al. and Stutz et al. [59], [60] developed a training method to address the problem of the unseen adversarial samples. These adversarial samples can be concealed beyond the normal norm size of the attacks and cannot be addressed by the usual adversarial training. The other method incorporates other unique regularization methods and addresses issues with real-life

adversarial samples [61]-[73]. Despite these modifications and improvements, the limitations of adversarial training still exist and generally cannot be fully tackled via one solution.

2.2.3 Generative Models

Generative models represent one of the useful types of models of architectures for data generation and synthesis. Multiple proposals have considered the use of these generative networks for robust adversarial training purposes. The architectures of these generative models include GANs, auto-encode-decoder, and diffusion models. Of these architectures, the GANs and auto-encode-decoder fit the adversarial training schema; hence, these two types of models are reviewed in this section.

The GANs model used for adversarial attack defense typically focuses on one of two major concepts: *distribution transferring* and *data augmentation*. Distribution transferring GANs [77], [78], [83] capitalize on the encode-decoder generative architectures, such as cycle GAN and U-net, to transfer the adversarial distributional data to the normal data distribution to mitigate the effects. The benefit of this method is that it utilizes an efficient architecture, and the generative structures can be interchanged to work with different classifiers. On the other hand, the limitation of this method is that it can only be effective with the specific adversarial samples it trained for and cannot be universally generalized. It also requires a heavy adversarial sample generating process similar to PGD adversarial training, and it does not improve the original model's robustness. Additionally, extra models must be deployed in real time to realize the adversarial denoise process. Another type of GAN [79]-[82] utilizes a self-supervised learning method that provides sample reconstruction to realize a simpler cleansing effect. It generally does not require attack

method implementation during the training. However, the method shares the limitation of additional model deployment and also does not address the original classifier's robustness.

Data augmentation GANs [16], [62] enhance the conventional adversarial training using GAN architecture or directly implement GAN for data augmentation. Liu et al. [62] use the GANs model to enhance the adversarial training process, wherein an extra discriminator is involved during the training to guide the optimization of the target classifier model toward the robustness parameters. However, this model does not address the issue of training complexity and introduces more parameters into the training process. Another GAN defense strategy [16] directly utilizes the original idea of GAN [21] and is significantly similar to conventional adversarial training. This model uses the generator to capture the loss gradient of the adversarially-trained classifier and to produce maximized loss samples; it then makes the classifier generalize on these samples. The benefit of this type of GANs is that it is generally less computationally complex compared to PGD adversarial training, and it can be implemented without a specific attack method. The generator can also effectively produce more data samples during the training process, which leverages the problem of data-hungry limitations. However, the training result of this type of model heavily depends upon the generator formulations and the capability of the generator to capture the proper gradient information [87]. One challenge of formulating the generator to realize different constraints of adversarial samples is that most lectures only include one type of generation constraint [16], [62], [83]. The other limitations include overfitting [87] and gradient saturation [87], which are the traditional challenges involved in GAN training. This thesis extends the GAN augmentation models and aims to provide more insight into the potential details of GAN used for adversarial robustness purposes.

The thesis evaluates the different components of GAN and particularly its adversarial sample generator network to provide improvement directions regarding the overall training performance of the GAN model.

The auto-encode-decoder [74]-[76] can also be used to generate adversarial samples. The general formulations include a similar min-max formulation compared to GAN [75] and the classifier latent space decoder [74], [76] for adversarial sample generation and training. The major disadvantages are similar to GAN, in that the performance depends upon the generative model's capabilities.

2.2.4 Ensemble Models

Ensemble adversarial training provides a different approach compared to other adversarial training methods. Ensemble adversarial training mainly focuses on decreasing the transferability of an adversarial attack across multiple different models [22], [84]-[86]. The method generates adversarial samples from a pre-trained model and includes these samples to train other independent models. Hence, the new model can generalize on these adversarial samples that the adversarial sample cannot simultaneously target both models. Ensemble adversarial training is highly effective for defending against transfer-based black-box adversarial attacks [22]. However, if the attack algorithm directly operates on the target classifier, the ensemble adversarial training cannot provide such robustness against these attacks [22]. Another benefit is that this training method does not require real-time adversarial sample generation, which results in heavy training overhead.

2.2.5 Limitations

The overall limitations of the current algorithm-based adversarial training frameworks for adversarial attack defenses include generalization problems, overfitting, computational

complexity, and limitations related to unseen distributional shifted data [1], [12]. There are also limited strategies and information to understand the effects of these adversarial samples. Hence, a flexible and universally applicable method is necessary to provide more usability in terms of generalization on different constraint attacks, to diminish the training complexity, and to reduce other negative effects of adversarial training.

The GANs augmentation frameworks may not provide state-of-the-art training results in terms of accuracy against adversarial sample and overfitting, since these frameworks are limited to the conventional problems related to GAN, such as gradient saturation and overfitting. They are also limited to their architectural design and additional parameters of the generators involved during the training process. However, the GAN model can provide some unique benefits for formulating the adversarial training framework. With proper design, GAN can typically be more backpropagate efficient compared to PGD adversarial training. The generator can also be modified to adapt it to different formulations for different constraints of adversarial sample generation with minimal changes. The generated adversarial sample also provides additional data points for classifier generalization. This generator can also be regarded as an alternative adversarial sample generation method for further research exploring the differences between generator-captured adversarial samples and adversarial samples from other algorithms.

2.3 Summary

This thesis focuses on the benefits of GAN models and proposes a GAN adversarial training architecture with several novel formulations. The formulations are experimented with and compared to improve the training performance and the robustness of the GAN training method. Furthermore, the proposed method enables low-complexity adversarial

training and can also defend more complex attacks. The method also does not require specific attack algorithm implementation and is attack-independent compared to conventional adversarial training. The thesis provides a deeper evaluation of training epochs and the overfitting effect of the model to identify the hidden relationships. The summarized information is also presented in Table 2.1.

This chapter has presented the necessary background information and related research to understand adversarial attacks and adversarial training. The limitations of the current research indicate the need for more research regarding adversarial training strategies and a further exploration of adversarial samples. Chapter 3 discusses the proposed architecture and the different considerations regarding the generator formulations and constraint settings of GAN for robust data augmentation.

Chapter 3. Proposed GAN-Based Architecture

This chapter discusses the proposed architecture and formulation of the GAN model for data augmentation. The basic architecture is presented first. The basic generation and training strategy are also described and discussed. Subsequently, the different variations of generator formulations are listed. These formulations will be used in later experiments for performance comparison. Finally, the chapter introduces the basic classifier architecture and different types of constraint GANs. The architecture and the formulations of the proposed GAN have been published in paper [18], [19].

3.1 Basic Formulations

The proposed architecture involves a standard composition of GAN with a generator and discriminator min-max formulation. Since the GAN is used for robust classifier training, the discriminator is referred to as a classifier in later sections. The generator of the architecture learns the gradient from the classifier and provides the adversarially-generated data. The classifier then learns to classify the clean sample data and the adversarially-generated data using the correct labels. The optimization function of the overall architecture is based on the traditional GAN's objective function [21] and the previous work [16], and adapted as Equation (3.1):

$$min_D max_G \sum L\,(D(x_i), y_i) + \sum L(D(x_i + \varepsilon N(G(\boldsymbol{I}))), y_i) \qquad (3.1)$$

where x_i represents the original clean data sample, and the generator G produces a synthesis data sample under the limitation of norm size ε from a selected input I. The N function is a L_p constraint function that limits the output vector of G into a certain L_p constraint norm vector; as a result, the scalar ε can restrict the vector from N into a vector with a length of

ε in any L_p constraint. The generator learns to generate the synthesis data that maximizes the loss of the classifier *D*. The classifier *D* learns to minimize its loss on both synthesis data and clean data *x*. The loss functions *L* for both are the categorical cross-entropy loss, and can be described as Equation (3.2):

$$-\sum y_{true} * \log (y_{pred}) \tag{3.2}$$

where y_{true} is the expected output, and y_{pred} is the predicted output. This loss function represents the Kullback-Leibler (KL) divergence with the GAN model [21]. In this case, the generator maximizes the loss of the classifier; hence, the loss value is reversed for the generator by multiplying a negative one value. There is a more advanced loss, and divergence can be used for GAN models (i.e., the Jensen-Shannon divergence) [88]. However, the Jensen-Shannon divergence does not suit this type of training purpose, which includes classifier training.

A dual-generator formulation has been introduced to improve the overall stability of the architecture. The dual-generator formulation can provide a regularization for the GAN to generate more diverse data and to improve the overall quality of the synthesis data [89]. This formulation can also compromise the stability issue of using KL divergence. Figure 3.1 illustrates the overall architecture of the proposed GAN. With this dual-generator design, the input vector *I* of the generator is simultaneously fed into the generator, and each generator provides its output on the same input *I*. The output of each generator is scaled with the ε and added with the corresponding sample from the training data set simultaneously. The addition samples are the adversarial augmentation samples learned by the generator models. Then these addition samples are mixed with original training samples

and used for classifier training. The thesis implements two generators as the architecture used in experiments. The architecture performs well within the evaluation and is sufficient with the implemented datasets. Hence, all experiments are conducted with this design. The generator population can have an impact on training performance. However, it is not the focus of the thesis, and further research is required to conclude the impact of the generator population.

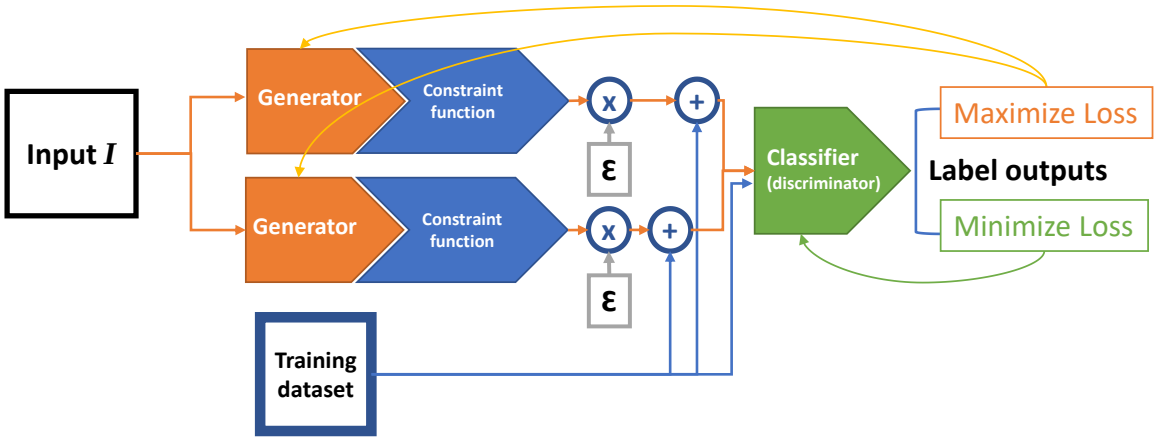

Figure 3.1: The proposed GAN-based Architecture.

Several symbols from the basic formula can be extended and described in detail. These include the input *I*, the constraint function *N*, the architecture of the generator *G*, and the classifier *D*. These components represent each of the adjustable parts that are evaluated independently in this thesis.

To evaluate the impact of each component, two major types of GANs are proposed for each L_∞ and L_2 constraint. These two constraint GANs require the different structures of the generator to realize L_∞ and L_2 vector generation. Hence, different constraint functions *N* are added to the overall formulation of GAN to transfer the output vector of the generator into different constraint vectors. As a result, the overall architecture of the

generator can be consistent, and only the switching in constraint function *N* are necessary to transform the augmentation into different constraints.

The second difference between the two types of GAN is scalar value ε. The scalar ε is used for limiting the maximum output vector size of each generator under their constraint settings. A classifier can have varying accuracies against same-sized adversarial perturbation under different constraints. Hence, it is necessary to select a scalar to limit the perturbation vector within the classifier generalization compatibility. However, the selection of the scalar value also depends upon the dataset dimensions. Hence, the exact scalar values are introduced in Chapter 4's "Dataset" section. The method of selecting the scalars was developed from extensive experiments and testing.

The third difference between the types is the generator architecture. Both types of GAN involved an image-to-image generator architecture; however, the two generator architectures were implemented in a slightly different manner. Since the L_∞ constraint adversarial is a more challenging problem, the generator uses a more flexible design so that the parameters can be adjusted based on the demand. Conversely, the L_2 constraint generator contains long-range skip connect to maximize the gradient backpropagation and to avoid potential gradient masking problems. However, the two designs can be alternated and switched for each type with minor differences. Additionally, the input vector *I* and the architecture of the classifier is shared with both types. This chapter discusses the shared components first and then introduces the components of each GAN type. The following sections introduce input formulation *I*, classifier architecture, and the two types of GANs in order.

3.2 Input Formulations of the Generator

This section lists the different formulations of the generator's input that are considered for evaluation. Equation (3.1) from Section 3.1 includes the generator symbol *G*. *G* can be written as a function estimation of an input *I* and an output *O* as in Equation (3.3):

$$O = G(I) \tag{3.3}$$

where *O* is the final estimated adversarial perturbation vector based on input *I*. The generator *G* must provide a reasonable estimation learned from the classifier's gradient. It is necessary to provide generator *G* with an input *I* that maximizes the possibility of successful estimation. In traditional GANs, the synthesis data frequently uses a latent noise z as an input *I* to provide a randomized diverse output sample [21]. However, the generator in the proposed architecture must transfer the input *I* into an adversarial perturbation vector of an original sample *x*. The generator is required to learn based on the current gradient of the classifier and the provided input *I*. Hence, an independent latent vector is insufficient to provide deterministic information for mapping the relationship between sample *x* and the adversarial perturbation output vector. In previous research [16], an encode-decoder generator was used to generate an adversarial sample based on the input of the clean sample *x*. Based on this paper and the gradient attack properties, four different input formulations have been proposed for evaluation. The included formulations are given as Equations (3.4), (3.5), (3.6), and (3.7):

$$I = x \tag{3.4}$$

$$I = concat(x, z) \tag{3.5}$$

$$I = sign(\nabla) \tag{3.6}$$

$$I = concat(sign(\nabla), x) \tag{3.7}$$

where the four formulations represent different conditions, assumptions, and learning objectives of the adversarial sample generation:

- Equation (3.4): Formulation provides the input *I* as the vector of the original clean sample *x*. It assumes that the adversarial perturbation is the direct transformation of the original sample *x*;
- Equation (3.5): Formulation provides the input *I* as the original data vector *x* and the combination of a latent vector *z*. It assumes that aside from the original sample, a randomized vector *z* can help to generate more diverse perturbation vectors;
- Equation (3.6): Formulation provides the input *I* as a one-step signed gradient vector from the current state of the classifier. It assumes that the more complex adversarial perturbation is the function transformation from the current state loss gradient of the classifier. To prevent the model overfit of the simple gradient vector, the additional sign function is applied on the gradient vector to only keep the sign direction of the vector;
- Equation (3.7): Formulation provides the input *I* as the combination of the one-step signed gradient vector and the original sample vector *x*. It assumes that the adversarial perturbation is related to both the current state classifier gradient and the original sample.

The generator *G* must use one of these pieces of input information to transfer the input vector into an adversarial vector. The input formulations are significant for this work and are necessary considerations to achieve an effective result in terms of robustness and

accuracy. The experiments compare and evaluate these formulations to determine which formulation is the optimal input to estimate the adversarial perturbation direction. Theoretically, these four types of inputs I can be used for all of the suggested types of models (L_∞, L_2, and small model). However, during the implementation and experiment, the L_∞ constraint GAN will evaluate all the inputs I, and rest of the model types will consider the best solution found by using the L_∞ constraint GAN. The reason for this is that the L_∞ adversarial samples from gradient-based attacks are more difficult to defend compared to the L_2 constraint, and L_2 constraint attacks are less effective against any classifier because of gradient masking [5]. Hence, the L_2 constraint GAN is used to evaluate the effectiveness compared to conventional adversarial training and the transferability of the robustness between the different types of models.

3.3 Classifier Architecture

The classifier architecture is a modified design based on the Visual Geometry Group (VGG) classifier model [93]. The VGG model is a famous deep learning architecture classifier for high-resolution image classification. The reason why a more advanced architecture such as ResNet is not considered is because the dimensions of data used in this thesis are low in sizes, and deeper architecture is not required to provide generalization. Hence, a simpler classifier model is preferred to conduct an efficient evaluation. The architecture of the model is modified and adapted for the proposed GAN's classifier. Figure 3.2 shows the general design of the classifier. Compared to the original VGG classifier, the proposed classifier has a shallower layer number and reduces the dense layer number. The model involves ten convolution 2D layers and one dense output layer. Each convolution layer uses a three-by-three kernel size and one stride size. There are four max pooling operations

between the layers. Each max pooling layer reduces the image size by half and increases the channel by the power of two.

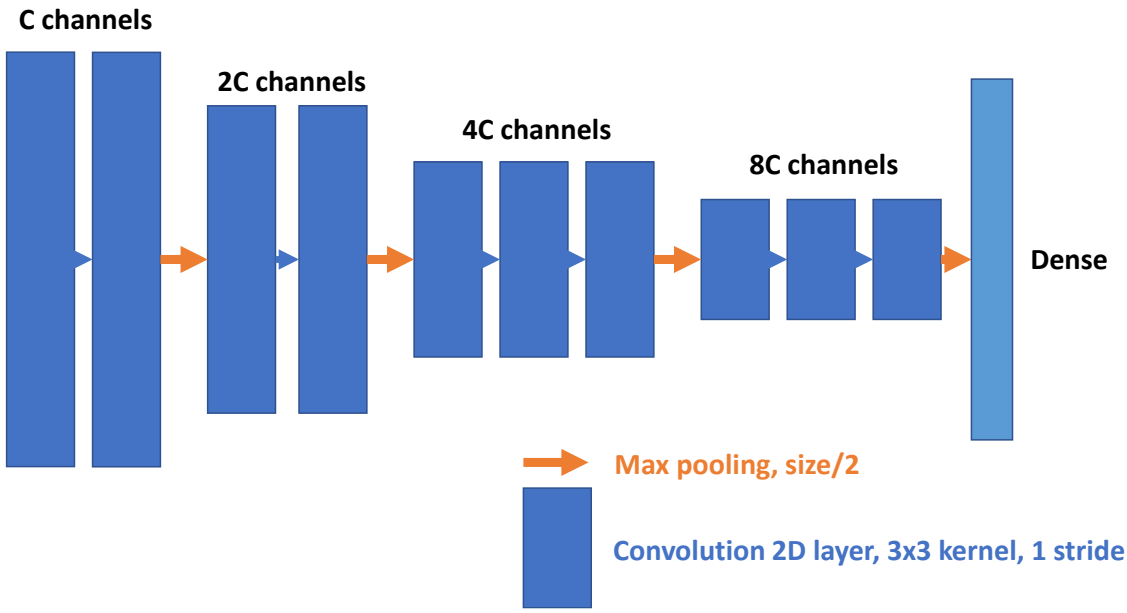

Figure 3.2: Modified small VGG model.

Other deep learning classifier architectures are also possible to use in the proposed GAN model; however, this thesis focuses on a comparison of the components of GANs. Hence, this standardized classifier architecture is used to control the variable.

3.4 L_∞ GAN

In this section, the detailed architecture of the L_∞ type GAN is presented. This section first discusses the constraint function that realizes the L_∞ norm vector generation and then presents the generator design. At the end, a summarized plot is presented to represent the overall design of this type.

3.4.1 L_∞ Constraint Function

The L_∞ constraint limits the maximum value across all the elements of a vector. Hence, the constraint function of the L_∞ constraint requires a value limitation of the output vector from

generator *G*. Hence, the tanh activation function is used to provide a soft value limitation with a properly defined gradient. The output after the tanh function is guaranteed to be within the value range [-1,1]. This tanh function can constrain the adversarial perturbation from the generator to be an L_∞ constraint with a maximum value of 1. The scalar ϵ can then multiply the vector into a vector with any L_∞ norm size of ϵ.

3.4.2 L_∞ Generator Design

The generator of the proposed GAN is built to transfer the selected input vector to the augmentation adversarial perturbation. All four of the input vectors proposed involve a consistent dimensional vector transformation, which means that an encode-decoder or image-to-image generator architecture is preferred. This thesis considers two main architectures for the generator. The first architecture is the Cycle-Gan [90] generator with a residual network backbone. This type of architecture is illustrated in Figure 3.3.

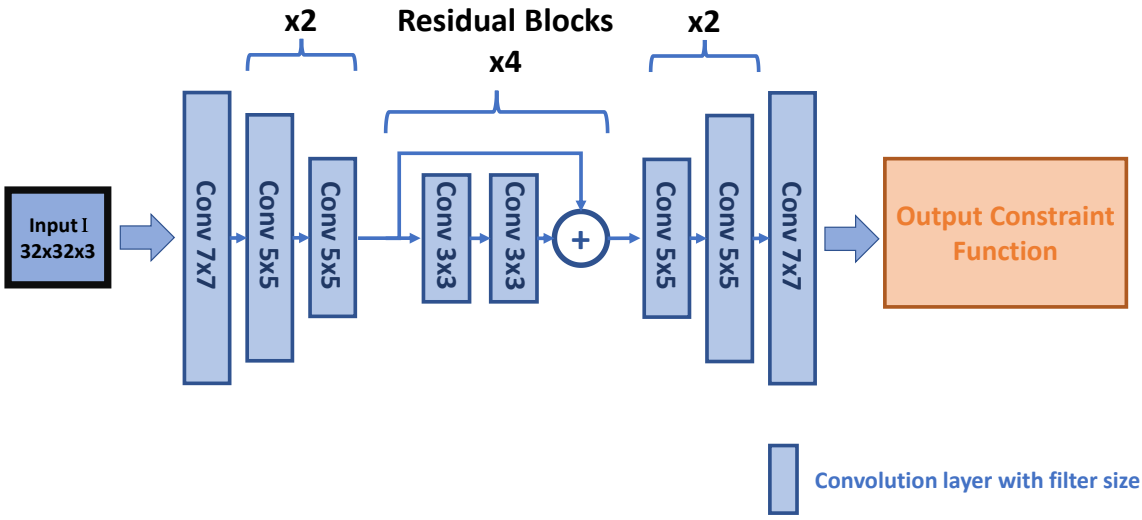

Figure 3.3: Residual generator model.

This architecture is well-known for texture and image style transformation [90]. The uniqueness of the residual backbone is that it uses the modular design of a residual block

to construct the main body of the model. The benefit of this design is that the skip connection of each residual block can promote gradient backpropagation [95], and the stacked residual block numbers can be easily adjusted to change the model layer numbers on demand. The overall design of the L_∞ GAN is illustrated in Figure 3.4.

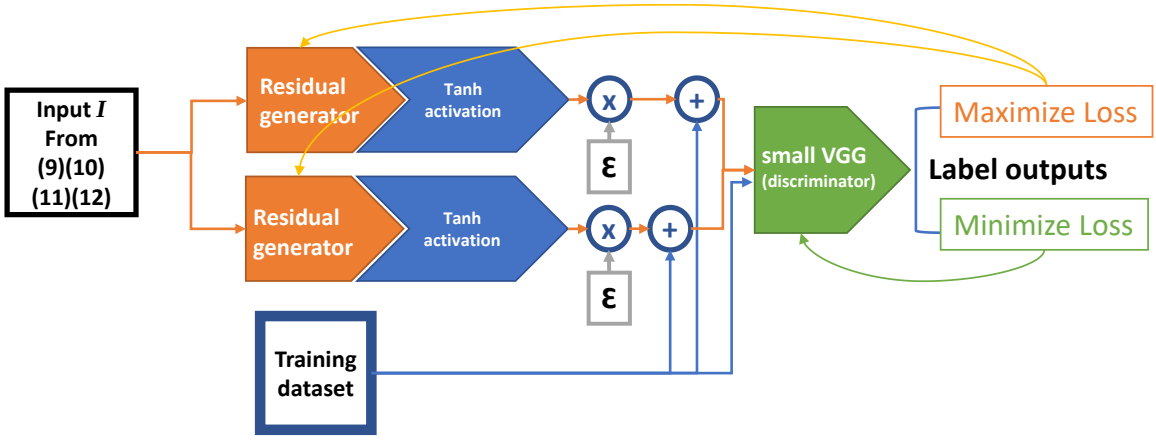

Figure 3.4: L_∞ GAN overview.

3.5 L_2 GAN

In this section, the detailed architecture of the L_2 type GAN is presented. The section follows a same order with the L_∞ type GAN section, introducing the constraint function *N*, scalar value, generator design, and a summary.

3.5.1 L_2 Constraint Function

The L_2 constraint limits the overall length of the vector in Euclidean space. Different from the L_∞ constraint function, there is no activation function that can limit the overall size of a vector. Hence, an L_2 normalization is implemented for the L_2 constraint function to convert the output vector of the generator to an L_2 constraint vector. The formulation of the L_2 normalization is written as Equation (3.8):

$$\mathcal{E} * N(G(I)) = \varepsilon \frac{G(I)}{||G(I)||_2} \tag{3.8}$$

The *G(I)* is the generator output vector, and *I* is any input vector. In the Equation (3.8), the L_2 norm size of the *G(I)* is calculated, and the *G(I)* is divided by the L_2 norm of itself to limit the overall L_2 norm size of *G(I)* to a value of 1. In this case, *N* as the constraint function transforms the vector output *G(I)* into an L_2 unit vector. In the subsequent process, the scalar ε can modify the unit vector of *G(I)* into an L_2 vector of a size ε. One disadvantage of this process is that the L_2 norm size of the resulting vector always has a fixed size of ε. However, the learning objective of the generator can be simplified, since the generator is only required to discover the perturbation direction aligned with the unit vector.

3.5.2 L_2 Generator Design

The L_2 generator architecture is a U-net architecture, which is illustrated in Figure 3.5.

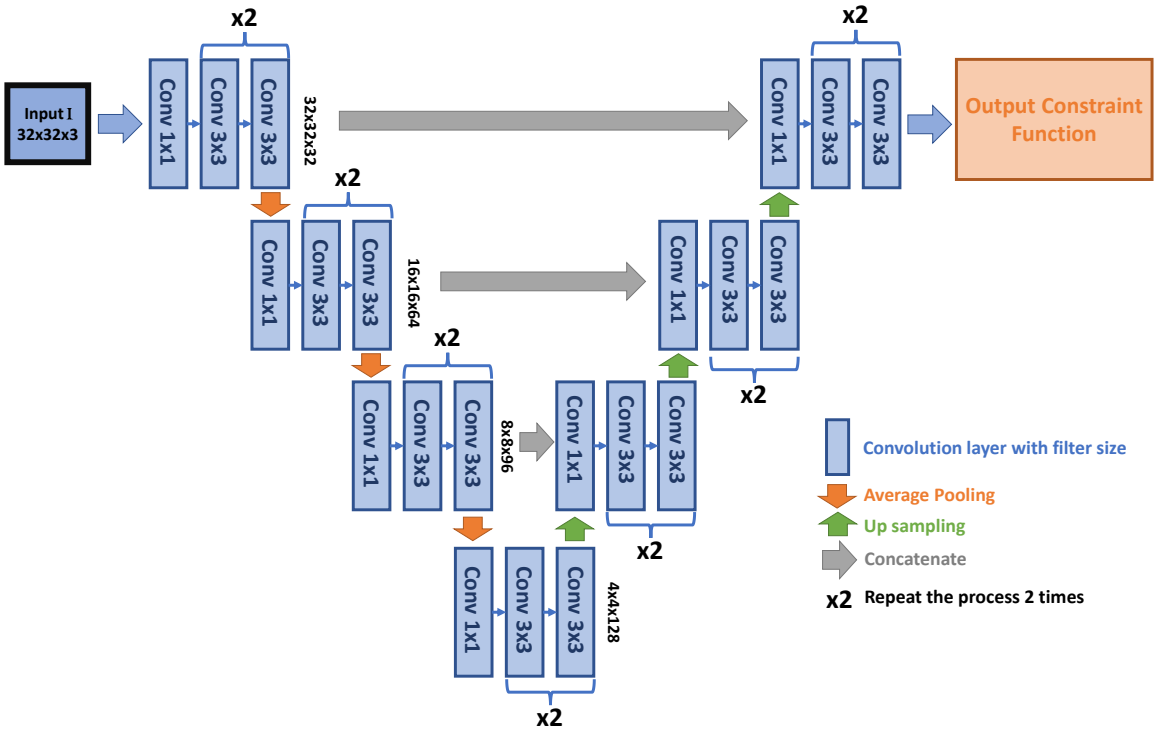

Figure 3.5: U-net generator model.

U-net architecture is well-known for instance segmentation [91] as well as diffusion models [92]. Different from the previous residual backbone generator, the U-net architecture does not include residual block modular design, which means that it is more challenging to modify such architecture for parameter tweaking. However, the U-net generator uses a long-range concatenate connection that formulate a symmetric style of structure. This long-range connection can further leverage the gradient problem of the generation process [92] and is more beneficial for generating L_2 constraint adversarial perturbation. Figure 3.6 depicts the overall design of the L_2 GAN.

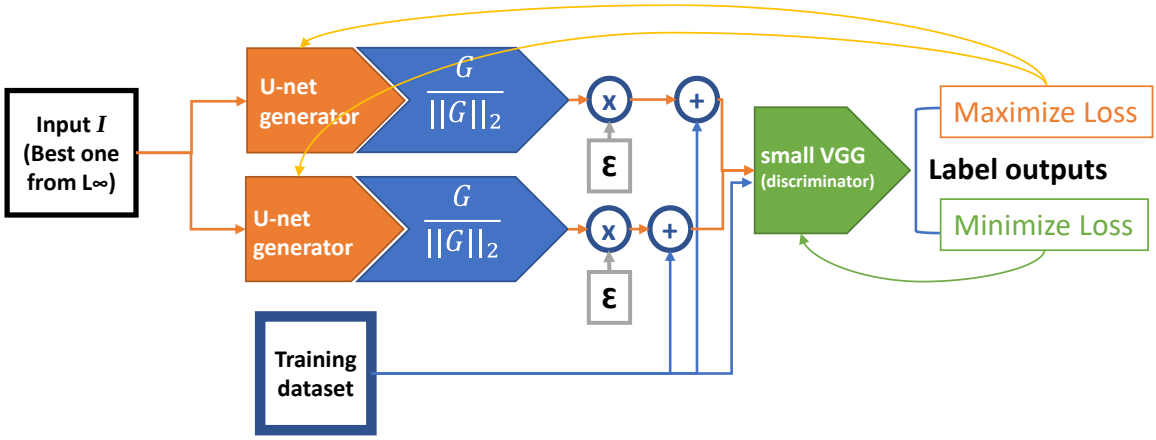

Figure 3.6: L_2 GAN overview.

3.6 Summary

This chapter has described the proposed architecture of GAN for adversarial data augmentation. It discussed the basic components of GAN, including the input formulations, constraint function, and generator and classifier architectures. Chapter 4 elucidates the details of implementation using the TensorFlow framework and the case studies performed to evaluate the GAN architecture.

Chapter 4. Implementation

This chapter introduces the libraries and frameworks used to implement the proposed GAN. The tools and attacks used for evaluation are also discussed. In subsequent sections, the implementation of the generator and classifier, training hyperparameters, and datasets are addressed.

4.1 Libraries and Tools

This section presents all the libraries and tools used for building the models and evaluation. The library used for building and training the model is primarily *TensorFlow 2* [98], and the tool involved in the evaluation is the *Adversarial Robustness Toolbox (ART)* [99].

4.1.1 TensorFlow 2

TensorFlow 2 [98] is an open-source deep learning library provided by Google Brain. The library can use the Python programming language and provide a broad range of supported features for building, training, and evaluating deep neural networks. The dynamic graph-building features are designed to support common deep learning architectures, including VGG net, residual net, GAN, and diffusion models. The built-in layer modules also provide seamless access to common deep learning operations, such as convolution, recurrent, and multiheaded attention operations. Multiple loss functions are provided for different learning objectives of deep learning models, including mean squared error loss and cross-entropy loss for regression and classification. The library also provides various optimizers, including the stochastic gradient descent and Adam optimizer with changeable hyperparameters, which renders the model's implementation and training convenient and accessible.

The proposed generators and classifier implementations use TensorFlow 2's layer API. The layer API consists of the basic building blocks to construct a deep neural network model with tensor process computation functions. The layer functions include the basic convolution layer, dense layer, pooling layer, and activation function layers. Each layer function can use the "call" function to stack and connect to the prior layer's output or any input tensor. Figure 4.1 demonstrates the process of the layer object.

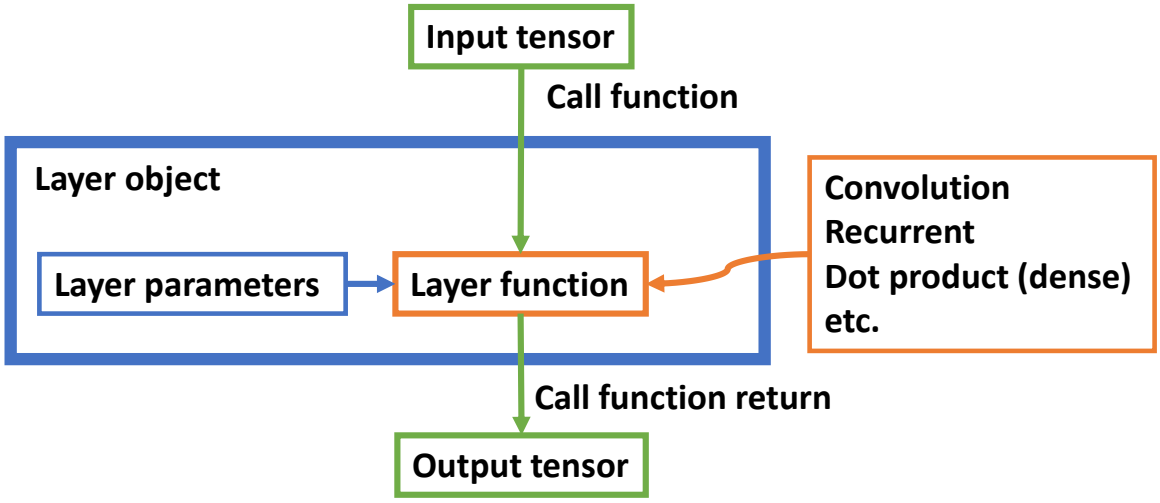

Figure 4.1: Layer object.

There are other options for the deep learning library. However, due to the high functionality coverage and the user experience of the researcher, TensorFlow 2 is the library that we have opted to utilize for the implementation of the models in this thesis.

4.1.2 Adversarial Robustness Toolbox

The Adversarial Robustness Toolbox is an open-source library that provides various attacks and defensive strategies for adversarial machine learning research [99]. The library includes evasion, poisoning, extraction, and inference attack types, which can be used to

exploit the vulnerability of any deep learning system. The gradient-based adversarial attack algorithms are included in the evasion attack category that directly attacks the input sample of a given classifier. The defensive strategies provided by the library include the preprocessor, postprocessor, trainer, transformer, and detector. However, this thesis implements its own strategy, and no defensive strategy is used from the library. The adversarial robustness toolbox also provides a high degree of compatibility with the TensorFlow 2 frameworks. The attack and defense strategies can be easily used for any TensorFlow 2 model.

4.2 Implementation Details

This section presents the implementation details regarding the GAN-based architecture to augment and train a robust deep learning classifier model. In particular, this section focuses on explaining the detailed implementation of the different generator formulations, output constraint functions, and the model parameters. The input layers' implementations and the classifier parameters are shared by both types of GANs, and they are discussed first. The output constraint function implementation and generator parameters are then presented in separate sections. Finally, the data processing method and the adversarial attack implemented for evaluation are discussed at the end.

4.2.1 Input Implementation

Several different layers have been built using the TensorFlow layer API to implement a flexible input formulation. The basic input layers for the image sample, latent noise vector, and gradient vector use the `layer.Input()` function to initiate the input tensor with a symbolic tensor object. The input shape for the image sample and gradient vector is the image's dimension (e.g., the image with 32×32 pixels and three color channels will result

in an input shape of 32×32×3). The input shape for the latent noise is set at 128×1, since the generator takes in 128-dimensional randomly distributed real numbers as latent vectors. The combination of the input formulation is implemented using the `layer.Concatenate()` function to concatenate selected input tensors in channel-wised concatenation. For example, the image with three color channels will concatenate with its gradient vector with three channels to produce a new tensor with six channels. The 128-dimension latent vector will first connect to a linear dense layer and reshape into a shape consistent with the image dimension to realize the concatenation. After all the inputs are defined, the generator's main graph can select any input formulation that the user desires during the runtime. Only the inputs used will be realized for the runtime objective. Figure 4.2 illustrates the realization of the inputs.

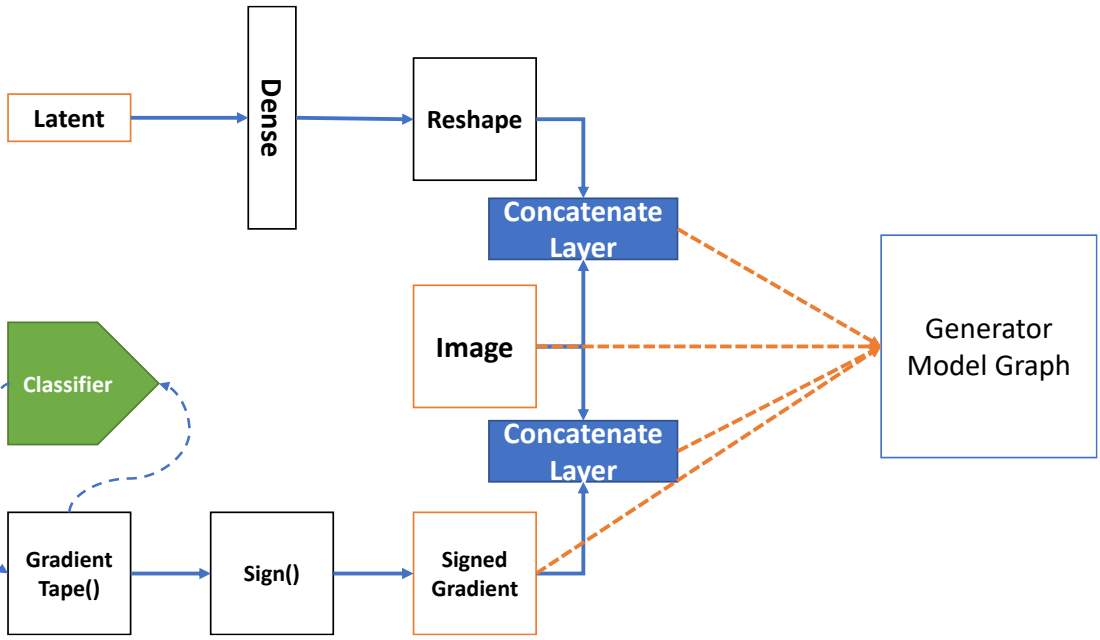

Figure 4.2: Generator input implementation diagram. The orange dotted line represents the import relationship of each input formulation.

To obtain the gradient vector, the proposed model pre-calculates the loss gradient using TensorFlow 2's auto-differentiation function `GradientTape()`. The function first computes the loss gradient regarding the current input image to the classifier. After the loss gradient is obtained, the `tf.sign()` function is used only to keep the vector sign of the gradient. This signed vector is then used as one of the inputs of the generator.

4.2.2 Classifier Parameters

This thesis uses a modified VGG model as the classifier model to demonstrate the function and performance of GAN augmentation. The VGG net is capable of classifying large-resolution image data [93] and is widely used in image classification applications. This thesis primarily concerns small-image datasets (32×32×3 maximum in pixel dimensions); hence, a reduced-size version of VGG net has been implemented. In real applications, this classifier network can be implemented with another common convolution classifier model, such as Resnet [95]. The parameters of implementation are presented in Table 4.1. The table includes the layers' output vector sizes, convolution kernel sizes and filter numbers. The output vector sizes are formatted by "height×width×channels" in values. The layer's kernel sizes and filter numbers are presented within squared brackets as "[kernel size×kernel size, filter number]". The values behind the squared brackets indicate there are duplication of the same parameterized layers in sequence. The stride size used for all the convolution layers is 1; hence it is omitted in the table. In addition, dropout regularization layers are also applied after each convolution layer to reduce the overfitting effect. The parameter for dropout is 0.3 for the first two convolution layers and 0.4 for the rest of the layers.

Table 4.1: VGG classifier parameters.

Output size	Discriminator
32×32×32	[3×3, 32] × 2
16×16×32	Max pooling
16×16×64	[3×3, 64] × 2
8×8×64	Max pooling
8×8×128	[3×3, 128] × 2
4×4×128	Max pooling
4×4×256	[3×3, 256] × 2
2×2×256	Max pooling
1024	Flatten
10	Dense (10)

4.2.3 L_∞ GAN

This section presents the implementation of the L_∞ GAN model. The implementation details include the API used for the output constraint function and the L_∞ constraint generator parameter settings.

4.2.3.1 L_∞ GAN Output Constraint Function

The constraint function relates to the generator's graph output tensors. After the final generator's main layers, several different computation layers are added according to the constraint settings of the generator to realize the different constraint functions.

The L_∞ norm constraint function accepts the output tensor from the generator's main graph and converts it into a constraint size vector under the L_∞ norm. The process entails tanh activation and multiplication. The tanh activation is implemented using the activation layer object with the built-in tanh activation function from TensorFlow 2. The tensor from the final main layer of the generator should be soft-clipped by this tanh activation layer and constrained within the value range of [-1,1]. The element-wised multiplication is then used to scale up the tensor by the scalar ε. The final tensor vector will generate the adversarial

perturbation from the generator and add to the original corresponding image vector to produce an adversarial augmentation image.

4.2.3.2 L_∞ GAN Generator Parameters

This section presents the L_∞ generator parameters settings in Table 4.2. The table includes the dimensions of each layer's output tensors, the filter size of each convolution layer, and the stride size of each convolution layer. Each layer's output is presented as "height×width×channels" format and each layer's parameters are presented as "filter size×filter size, filter number, stride number" format. The squared bracket indicates the implementation of residual blocks, and the value after the "×" represents the repetition time of the residual blocks in sequence. The stride size for residual block layers is always one; hence it is omitted in these layers. In addition to these basic parameters, a width value is implemented to adjust the global filter numbers of the layers in the generator. Table 4.2 presents the residual generator parameters.

Table 4.2: Residual generator parameters.

Output size	**Generator filters (×width)**
32×32×32	7×7, 32, stride 1
16×16×64	5×5, 64 × width, stride 2
8×8×128	5×5, 128 × width, stride 2
8×8×128	$\begin{bmatrix} 3\times3,\ 128\times\text{width} \\ 3\times3,\ 128\times\text{width} \end{bmatrix} \times 4$
16×16×64	5×5, 64 × width, stride 2
32×32×32	5×5, 32 × width, stride 2
32×32×3	7×7, 3, stride 1

4.2.4 L_2 GAN

This section presents the implementation of the L_2 GAN model. The implementation details include the API used for the output constraint function and the L_2 constraint generator's parameter settings.

4.2.4.1 L_2 GAN Output Constraint Function

The L_2 norm constraint function calls the built-in function `tf.math.l2_normalize` from TensorFlow 2. The calculation formula of the function is represented in Section 3.5.1. The output tensor for the L_2 constraint is normalized by the function to determine the unit vector of the tensor. The unit vector then multiplies with a scalar ε by using elementwise multiplication. The scaled output tensor is finally used as the adversarial perturbation for the L_2 constraint and is added onto the original corresponding image vector to produce an adversarial augmentation image.

4.2.4.2 L_2 GAN Generator Parameters

The L_2 constraint generator uses U-net architecture. The parameter settings are displayed in Table 4.3. The table includes the dimensions of each layer's output tensors and the filter size of each convolution layer. The output sizes are presented as "height×width×channels" format and the layer parameters are presented as "filter size×filter size, filter number" format. The squared bracket indicates a unit of building block of the model. In contrast to the L_∞ generator, the sampling methods used in this architecture include average pooling and bilinear up sampling, and the rest of the convolution 2D layers use the same stride size of one. Hence, the strides' values are omitted in this table.

Table 4.3: U-net generator parameters.

Output size	Generator
32×32×32	$[1\times1, 32] \times 1$ $\begin{bmatrix}3\times3, 32\\3\times3, 32\end{bmatrix}\times2$
16×16×32	Average pooling
16×16×64	$[1\times1, 64] \times 1$ $\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$
8×8×64	Average pooling

8×8×96	$\begin{bmatrix} 1\times1, 96 \\ 3\times3, 96 \\ 3\times3, 96 \end{bmatrix} \times 2$
4×4×96	Average pooling
4×4×128	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$
8×8×128	Up sampling
8×8×64	$\begin{bmatrix} 1\times1, 96 \\ 3\times3, 96 \\ 3\times3, 96 \end{bmatrix} \times 2$
16×16×64	Up sampling
16×16×32	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$
32×32×32	Up sampling
32×32×32	$\begin{bmatrix} 1\times1, 32 \\ 3\times3, 32 \\ 3\times3, 32 \end{bmatrix} \times 2$
32×32×3	[1×1, 3] × 1

4.2.5 Training Methodology Implementation

The training process of the proposed GAN architecture involved the adversarial training of the generator and the classifier. The training schema was directly modified from the TensorFlow 2 model API with two training loops. The outside training loop is called the "`fit()`" function and controls the maximum training epochs of the entire training process. Within this loop function, the data is split into different batches. The batch size of the data is established before the training starts. This thesis uses 512 as the standard batch size. The batch of the training data is sent to the inner training loop that represents the training step of each batch of data. Within each training step, the "`train_step()`" function is called, and the parameters of the generator and classifier are updated accordingly. The inner training loop ends when all batches of data are finished or updated. The new epoch then starts with a new update of the batches. The outer training loop ends until an established maximum epoch is reached. The details of the maximum training epochs depend on the

case studies of the thesis. The overall process is illustrated in Figure 4.3 and the pseudocodes is provided in Appendix A: Training Algorithms.

In each training step, the generators first produce an adversarial sample as additional training data. These adversarial samples mix with normal samples that feed into the classifier for the forward pass process. The loss gradient is calculated based on the mixed dataset, and the optimizer can use the gradient to update the parameters of the classifier to minimize the classifier's loss on both the adversarial sample and the normal sample. In the section half of the training step, the same procedure is performed for the forward pass process. However, this time, the generator's parameters are updated based on the loss gradient of the classifier.

The training step implementation is directly modified from the "`train_step`" function provided by TensorFlow 2. The training function of the GAN accepts three network models (dual generators and one classifier) to perform the training process illustrated in Figure 4.3. Additionally, during each training epoch loop, the random flip and shift augmentation method is used, as mentioned in Chapter 3. The random shift augmentation is set to shift the maximum 0.1 fractions of the image size in a random direction.

The optimizer selected for training is the Adam optimizer, which is widely used in GAN models [96]. Adam optimizer is an extension of the gradient descent [96] and its adaptive properties can accelerate the training process of the complex model with more advanced training objectives. The learning rate for the classifier is set at 0.0001, with 0.5 $beta_1$ and 0.9 $beta_2$. The learning rate for the generator is 0.0002, with the same beta values. The learning rate for the generator is larger because, during the adversarial training, the

generator should capture the loss gradient of the classifier first to generate an effective adversarial sample for augmentation. The larger learning rate can accelerate the learning speed of the generator.

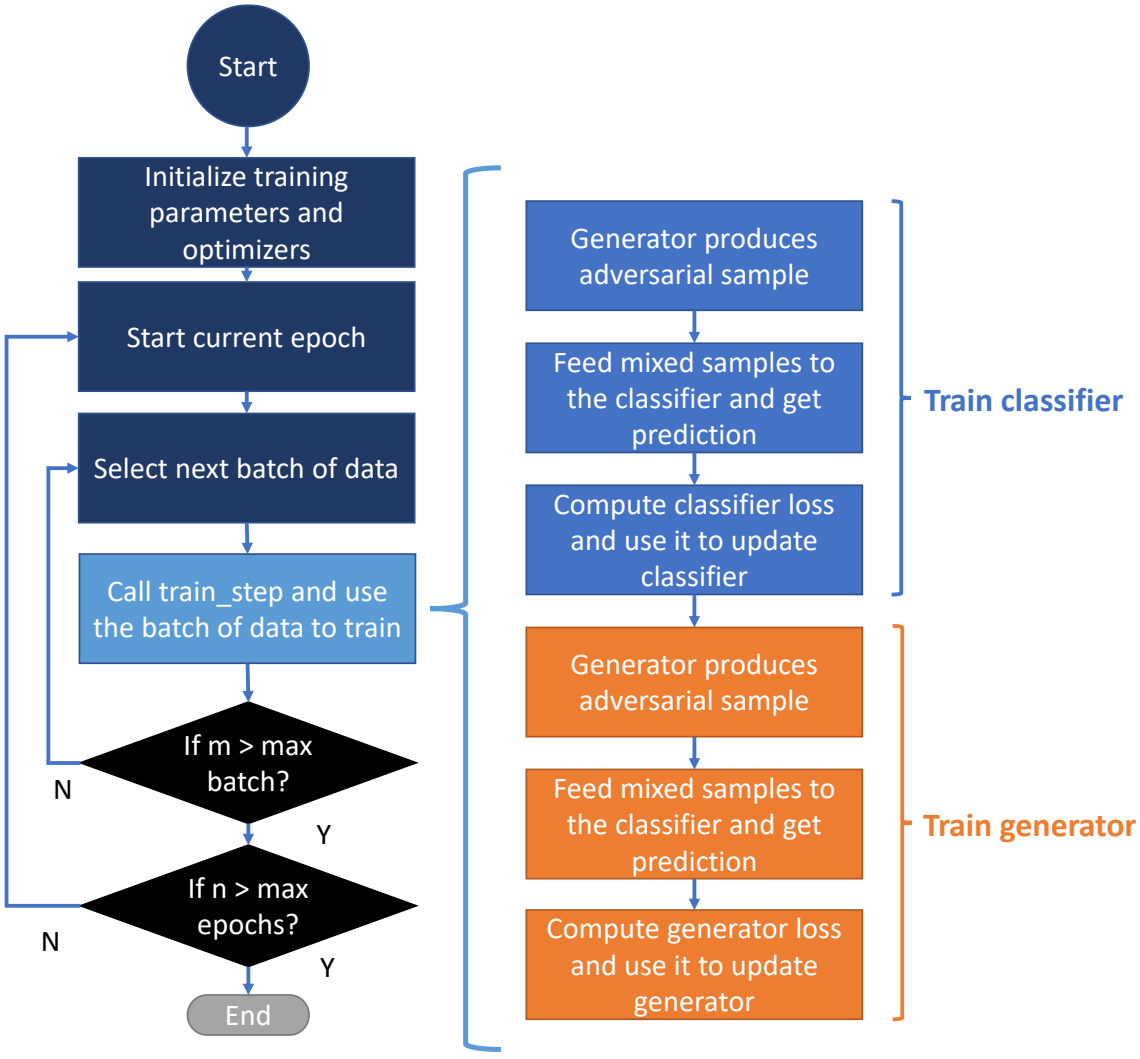

Figure 4.3: Training method of the GAN model.

Additionally, according to the proposed training procedure, the algorithm's execution complexity can be calculated using the *big-O* notation. The big-O notation can be used to represent the upper bound relationship between the input space variable numbers and the algorithm execution time. The experiment sections do not indicate the exact

execution time of the proposed model's training process. This is because the algorithm is implemented in a complex environment; hence, the runtime is heavily dependent upon the background hardware and the software environment at the time of the training. As a result, the exact training time is less convincing to represent the execution efficiency compared to the complexity analysis. In a real-life scenario, the big-O notation can also reflect the upper bound of the execution complexity and enable a comparison between the two algorithms' upper-bound runtimes within the same environment. Based on the standard training methodology, the big-O of a training function of a deep learning model can be expressed as Equation (4.1):

$$O(Epoch \times Step \times K) \tag{4.1}$$

where *Epoch* represents the overall training epochs of the model, *Step* represents the training step within one training epoch, and *K* is the backpropagation operation time within one training step. The proposed GAN does not modify the implementation of the training epoch and training step. Hence, the backpropagation time must be identified to analyze the training complexity. In the proposed GAN, three neural networks are involved during the training. Each of the networks required one backpropagation to update its parameters.

According to the input formulations of the generator, there are two proposals with the Equations (3.6), (3.7) that required a pre-calculation of the gradient. Each pre-calculation requires one-time backpropagation, and each time a generator produces an output, the pre-calculation is called. Both generators also share the input vector. Hence, the summarized number of backpropagations within one training step is five times at the most. In conclusion, this can be represented as K=5 for the proposed GAN method. In conventional adversarial training, the backpropagation time is dependent upon the

implemented attack iteration and the model number. One model is generally trained at a time. Hence, the K for conventional adversarial training is $(1+N)$, where N is the attack iteration time. In comparison, the proposed method has a lower upper-bound complexity when $N > 4$. In most cases, conventional adversarial training requires more iterations than four to generate augmented adversarial samples. Hence, most of the time, the proposed GAN has a lower worst-case execution time within the same environment.

4.2.6 Attack Algorithms

Within the scope of this thesis, the primary attacks are the gradient-based white-box attacks, which use the backpropagate gradient descent to generate an adversarial sample. The thesis selects this category of attack algorithms since they are effective and easy to implement to target any deep learning classifier with parameter access [3]. The proposed GAN training also uses gradient descent as core optimization methodology; hence the thesis can evaluate similar optimization methodologies with different formulations but act as different roles within attack and defense. The proposed GAN must be evaluated against this type of attack algorithm to determine the effectiveness of the training. The proposed GAN should be able to provide similar performance against other types of attack algorithms with similar optimization methodologies. However, the attacks with different optimization methods and constraints may results in different data distribution [3]. Hence, the proposed GAN does not provide the generalization on these attacks. GAN augmentation is a defensive strategy that does not include additional layers of the preprocessing of data or any significant modifications to the layer structure of the convolution classifier. Hence, the gradient is well-defined for all layers of the convolution classifier used in experiments. There is no requirement to implement a gradient attack with an approximate gradient to prevent

artificial gradient masking [3], which is sometimes used as a defensive mechanism. In these cases, the thesis uses the basic gradient-based attacks, namely FGSM and PGD, to evaluate all classifiers' performance. The algorithms of these attacks are provided by the Adversarial Robustness Toolbox (ART) library, introduced in Section 4.1.2. The FGSM is a simple one-step attack that is efficient in generating adversarial samples. However, the robustness evaluation using FGSM may not reflect the actual adversarial robustness of the classifier [5]. The robustness of a classifier requires a repetitive search of the vulnerability by using multi-iteration attacks, such as PGD [5]. However, it is still interesting to present the performance gap of a trained classifier between the FGSM and PGD. Hence, the FGSM is included for evaluation.

The operation of FGSM and PGD attacks is based on the assumption of the adversarial sample $x+\delta$ [3], where the implemented attack algorithms provide the solution of adversarial perturbation vector δ and the perturbation vector δ adds with test data sample x to construct the final adversarial sample. The test data sample x includes every data sample within the testing datasets. The procedure of the evaluation and attacks is as following:

1. Training the classifier used for evaluation with the proposed GAN architecture with training dataset.
2. Selecting testing dataset and the trained classifier. Using FGSM and PGD to generate the adversarial perturbation vector δ for each data sample within the testing set. During the generation, gradient descent is used as the optimization method to optimize the perturbation vectors based on the trained classifier and each testing sample.

3. Adding the generated adversarial perturbations to their corresponding testing samples to construct the adversarial samples for each testing data sample.
4. Using the trained classifier to classify the adversarial version of the testing data samples and record the accuracies of the classification. The accuracies are used for robustness evaluation.

The parameters of these gradient attacks include perturbation norm size, L_p constraint, and iteration number (PGD only). This research focuses on L_∞ and L_2 constraints, since the primary implementations of GAN are also on these two constraints. The perturbation norm sizes depend on the dataset, constraints, and the model's practical robustness. A proper value should be established around the upper limit of the robustness of the classifiers to ensure a valid evaluation. For this reason, the thesis proposes different sets of norm values for different datasets and constraints. The details of the value implementation are delineated within the dataset section and discussed under each dataset subsection. The iteration number of PGD attacks must satisfy the high effectiveness of attacking the target classifier; however, the increasing iteration number may cause the evaluation to be finished in an extensive time. For this reason, the PGD iteration is set to 100 iterations for most of the L_∞ evaluations; however, an individual section is provided in Chapter 5 to validate the best PGD 100 results, with PGD 1,000 iteration attacks for L_∞ GAN. For the L_2 evaluation, the gradient-based attacks are naturally less effective in this constraint [5]. Hence, the experimental evaluations will directly use PGD with 1,000 iterations as the default setting for this constraint. Furthermore, the L_2 constraint PGD attack cannot reflect the actual robustness of the model for the same reason. Hence, the L_2 evaluation is primarily a

comparison of the effectiveness between conventional adversarial training and GAN training to evaluate the effect of natural gradient masking.

4.3 Datasets

This section discusses the case studies conducted on two datasets used in the evaluation and verification of the implementation. The two datasets considered include the grayscale MNIST dataset and the CIFAR 10 small colorful image dataset. In the primary evaluations, CIFAR 10 has been used, as it is a dataset with higher dimensions, and it presents a more challenging task for classifiers to generalize on. The MNIST dataset with low-dimensional data has been used to verify the model's performance in low dimensions. In this case, the low dimensions refer to grayscale images with a similar image size to CIFAR 10 images. These datasets are well-studied within the adversarial machine learning research, and they are well-suited for this thesis as a standard for evaluation. During the evaluation, the proposed classifier can reach good classification results without adversarial attacks. However, it is still challenging for the deep learning model with proposed parameter numbers and the sizes to generalize on the adversarial samples of these datasets. Hence, the thesis focuses on these datasets to provide robustness improvement. More complex datasets, such as ImageNet, have not been considered in this thesis, since they are beyond the scope of the thesis, and the hardware environment is also limited to supporting a larger model implementation. The other datasets may require deep learning models and GANs with different parameter numbers to perform a successful defense. Hence, extensive study is required to evaluate different datasets.

4.3.1 CIFAR 10

The Canadian Institute for Advanced Research (CIFAR) 10 dataset is a dataset with 32×32×3-dimension images with three RGB color channels. This dataset includes 60,000 low-resolution colorful images with ten class labels (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck). CIFAR 10 is also embedded in the library of TensorFlow and is widely used to test the image classification of convolution neuron networks.

The training data of CIFAR 10 is set at 50,000 individual images by default. This thesis implements two output constraint functions to train the CIFAR 10 classifier with GAN. In the L_∞ constraint setting, the scalar is established as 16/255, and in the L_2 constraint setting, the scalar is 64/255. The reason for the larger L_2 value is that the L_2 constraint adversarial perturbation is more easily affected by gradient masking [5], and it typically represents a different adversarial attack scenario. A smaller value may not be effective in attacking the classifier. For the same reason, the L_∞ PGD attack is used for an overall comparison between the classifiers. The generator architecture used in L_∞ constraint training is the residual generator architecture, since it is flexible in adjusting the parameter settings. The U-net architecture is used for L_2 training after more data is collected.

The other 10,000 individual images are used for testing and evaluation. The same attack algorithms, namely FGSM and PGD, are implemented for a robust evaluation. The classifiers trained with different constraint settings are tested with the corresponding constraint adversarial sample. The L_∞ constraint adversarial sample uses 4/255, 8/255, and 16/255 as perturbation norm sizes, and the L_2 constraint adversarial sample uses 8/255, 16/255, 32/255, 64/255, and 128/255 as norm sizes to construct an evaluation of adversarial samples. This thesis includes four groups of experiments to test the detailed formulations

of the GAN architecture. The experiments include a comparison of the input formulations, a generator width comparison, a training epochs comparison, an L_2 robustness comparison, and an L_∞ robustness transferability evaluation for the L_2 constraint-trained classifier. The following delineates the details of these experiments and their purposes.

- The input formulation comparison experiments evaluate the formulations of the generator proposed in Chapter 3. This experiment examines the impact of the generator formulation on the adversarial training results of the classifier. Four GAN models are built with the proposed methods for this experiment, and all the GAN models are trained with 100 epochs.
- The generator width comparison involves conducting a parameter number evaluation and investigating whether the parameter number of the generator exerts a significant impact on the training results. Three GANs are built for this experiment, with width values of ×1, ×2, and ×3 and trained with 100 epochs.
- The training epochs experiment involves employing the optimal GAN settings from previous experiments. The optimal GAN is trained with 100, 200, and 300 epochs for the robustness comparison. This experiment is used to identify any positive or negative effects with an increasing amount of training.
- The classifier trained with L_2 generators is then evaluated with the adversarial sample under the same constraint. Furthermore, this version of the classifier is also tested with an L_∞ constraint adversarial sample to evaluate the transferability of different forms of constraint training.

The thesis also involves another convolution classifier with proposed architecture of Section 3.3 trained with no defenses and compares the robustness improvements in major experiments. This undefended model serves as a baseline model.

4.3.2 MNIST

The Modified National Institute of Standards and Technology database (MNIST) dataset is a well-known machine learning dataset that includes 70,000 small, hand-written digital images in grayscale. Each image vector within the dataset has 28×28×1 dimensions with one color channel. In the classification task, the classifier must classify the number within the image into 0 to 9 class labels. The dataset is easy to implement with the default inclusion of the TensorFlow database. Hence, it is an appropriate resource for evaluating the adversarial robustness of small convolution networks.

The dataset is split into the training part and the testing part with a default setup; 60,000 images are used for training, and 10,000 are used for testing. The training implementation for this dataset required a slight shift in the input dimensions of the generator and classifier. To make the input size consistent with the standard classifier, the 28×28 image size is padded with zero to scale the image up to 32×32 in size; as a result, all the neuron network models of GAN can accept the input. The scalar ϵ is set at 0.3 with the L_∞ output constraint function, which limits the maximum perturbation for a pixel within [-0.3,0.3]. Furthermore, the MNIST dataset is a rather simple dataset to generalize on; hence, the training epochs for this data set are set at ten epochs.

The attack algorithms used in the evaluation include the one-step gradient attack FGSM and the iterative gradient attack PGD. The GAN model only involves augmentation as a defensive strategy; therefore, no other method is required for these attack algorithms

to perform a successful attack, since the gradient is well-defined for all layers of the classifier model. The evaluation constraint of this dataset is mainly the L_∞ constraint, since this dataset is primarily used for validating the functionality of the model, and the L_∞ constraint typically reflects a more realistic robustness of the model. A more detailed evaluation of components is presented in the next case study. During the L_∞ constraint evaluation, three norm sizes are considered for adversarial perturbation: 0.1 (25.5/255), 0.2 (51/255), and 0.3 (76.5/255). These values define the maximum L_∞ norm size for the evaluation of adversarial samples. All accuracies under these norm sizes are evaluated. In addition to the comparison, the same structured classifier without the GAN architecture is trained using the default training method with the SGD optimizer (0.001 learning rate). This undefended classifier is tested alongside with the GAN classifier to compare the robustness improvements.

4.3.3 Data Preprocessing

This section describes the data preprocessing process before the training of the proposed GAN model. The data preprocessing prior to training any image classifier involves a standardized normalization and randomization processes. This thesis focuses on the augmentation process's contributions to the adversarial robustness of a classifier. Hence, the procedure uses a standardized dataset to perform the evaluation. As a result, no other significant feature selection and data analysis process is included in the implementation of the framework. The standardized normalization process for the image classification task converts the pixel value of the standard range [0,255] into a normalized range [0,1] using Equation (4.2):

$$u = \frac{u'}{255} \tag{4.2}$$

u' is the pre-normalized pixel value, and u is the normalized pixel value. The other preprocess includes standard augmentation techniques, such as random flips and shifts for each instance of the training data sample. These techniques can be used to reduce the overfitting effect of image classifier training [94].

4.4 Summary

This chapter elucidates the details of the implementation of the GAN. The library and tools are described and presented. The other information includes the implementation of detailed architecture and parameters of the models, training methods, attack implementations, the case studies dataset used, and data preprocessing. Chapter 5 presents the evaluations for both case studies and discusses the results.

Chapter 5. Evaluation Results

This chapter discusses and presents all of the experiments and evaluations for all the model formulations. The subsections are organized based on the model types, and within each subsection, different sets of experiments and evaluations are conducted based on the purpose of the model. The results are plotted and analyzed to indicate the accuracy against different types of attacks.

5.1 Evaluation Metrics

The evaluation involves multiclass scenarios; hence, the accuracy metric is used for evaluating the accuracy of the classifiers. The formula of the accuracy metric is described as Equation (5.1):

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{5.1}$$

where the TP, TN, FP, and FN represent true positive, true negative, false positive, and false negative predictions, respectively. The accuracy metric is applied to all of the performance evaluations of our trained classifiers. In each experiment, the suggested GAN is trained with the proposed training methodology. Subsequently, the trained classifiers are attacked by the evaluation attack algorithms, and the accuracy metric is applied to calculate the accuracy of the model on the adversarial samples as well as on clean data. For the ease of description, the accuracy that represents the performance of the classification under adversarial attacks is defined as *robust accuracy*. The accuracy that represents the performance of classification with original data sample from the testing dataset is defined after *clean accuracy*. In subsequent sections, these two words are used to describe the performance of the classification for different scenarios.

In the experiments and evaluations, the primary methodologies to generate adversarial samples are the fast gradient sign method (FGSM) and projected gradient descent (PGD), introduced in Section 4.2.6. These attack algorithms are used as threat models to evaluate the robustness of the proposed model formulations.

5.2 L_∞ GAN

This section introduces the L_∞ GAN experiments and evaluations. First, the generator input formulations are evaluated. The optimal formulation is used to compare the effects of different parameter numbers. The generator with the best formulation and parameter settings is then used to train three classifiers with different epochs. The L_2 constraint experiment is conducted based on the knowledge gleaned from the L_∞ experiments with a more complex generator setting. To make each section easy to keep track of, the basic parameter settings to train the classifiers are listed within each section before the evaluation.

5.2.1 Input Comparison

This section presents an evaluation and comparison of the robust accuracy of the training results from different generator input formulations. The accuracies are obtained by testing the trained classifiers on the 10,000 testing data from CIFAR 10. The experiment includes four different groups of classifiers adversarially-trained with different generators. The performance of the classifiers reflects the effectiveness of the training and data augmentation yielded by different generator formulations. All attacks and generators are implemented in the L_∞ setting. The followings are the parameter settings used for this experiment section. These settings are mentioned before in the Implementation chapter (Chapter 4) and this list is for reminder and ease of reading:

- GAN training parameters:
    - Generator input: x, $x+z$, $sign(\nabla)$, $x + sign(\nabla)$
    - Generator width: ×1
    - Generator output constraint function: L_∞ constraint with scalar ϵ 16/255
    - Trained epochs: 100
- Attack of evaluation:
    - Attack used: FGSM and PGD
    - Constraint: L_∞
    - Norm size: 4/255, 8/255, 16/266

Figure 5.1 presents three groups of behaviors according to the accuracies of the classifier. The first group is the baseline model. The baseline model has an optimal accuracy when it classifies original samples. However, with the lowest adversarial perturbation size, the model's accuracy decreases to around 10% against FGSM and 0% against PGD. With a greater perturbation size, the classification accuracies remain around the same value for the baseline model. This suggests that without any defense, the classifier cannot defend the smallest sizes of adversarial noises. Another observation regarding the baseline accuracies is that the FGSM accuracies remain 10% across all the perturbation sizes. This suggests that FGSM is limited in its optimization ability to find adversarial samples.

The second group includes the classifier trained with $I = x$ and $I = concat(x,z)$ input formulations. These classifiers generally exhibit similar performance and display some adversarial robustness improvement compared to the baseline. The accuracies against the strongest PGD attacks are above 10% for this group of classifiers, and the accuracies

against smaller perturbation sizes exhibit more significant improvements. Another observation is that the PGD accuracies of this group of classifiers have close values compared to the FGSM accuracies. This result indicates that after training, the classifier gained a similar robustness against the one-step gradient attack FGSM and the multi-step gradient attack PGD. A deeper explanation of this is that the perturbation directions uncovered by both FGSM and PGD become similar. This indicates two scenarios, namely that the gradient landscape of the classifier becomes more simple or more complicated. The more probable explanation is that the landscape is simplified because the PGD and FGSM accuracies remain low, which suggests that the adversarial samples are indeed effective and that the GAN training promotes this type of gradient from the classifier. However, regardless of the context, the robust accuracies of these classifiers are not ideal in the comparison and remain vulnerable to the attacks.

The third group, namely the classifiers adversarially trained with the generator of $I = sign(\nabla)$ and $I = concat(sign(\nabla),x)$, also exhibit a similar performance. The classifiers trained with these generators achieve the most robust accuracies with all attack settings. Compared to the previous group, the PGD accuracies remain significantly higher in most situations. However, the PGD accuracy starts to decay more rapidly when a 16/255 perturbation size is introduced and only results in a slight improvement under this perturbation size. In this case, the classifier is more robust against a perturbation size below or equal to 8/255. Furthermore, the difference between the FGSM accuracies and PGD accuracies are larger, especially at the 16/255 perturbation size. The dramatic increase in difference indicates that the worse gradient direction of the classifier becomes more different when the perturbation is larger. Overall, the result indicates a close relationship

between the gradient information regarding the classifier and its adversarial sample. It suggests that the generator involving the signed gradient can provide an enhanced estimate of adversarial samples and that the classifier trained against them can have more robust accuracy. However, due to the difference between the FGSM and PGD accuracies, it suggests that with a sufficiently large perturbation size, the GANs of these formulations start to lose their effectiveness in augmenting the classifier against complex attacks. The reason for this might be that the estimation of the worst-case adversarial sample becomes excessively challenging for the generators under a large perturbation size. Hence, improvements are still required to allow the generator to provide a better estimation within a complex gradient landscape.

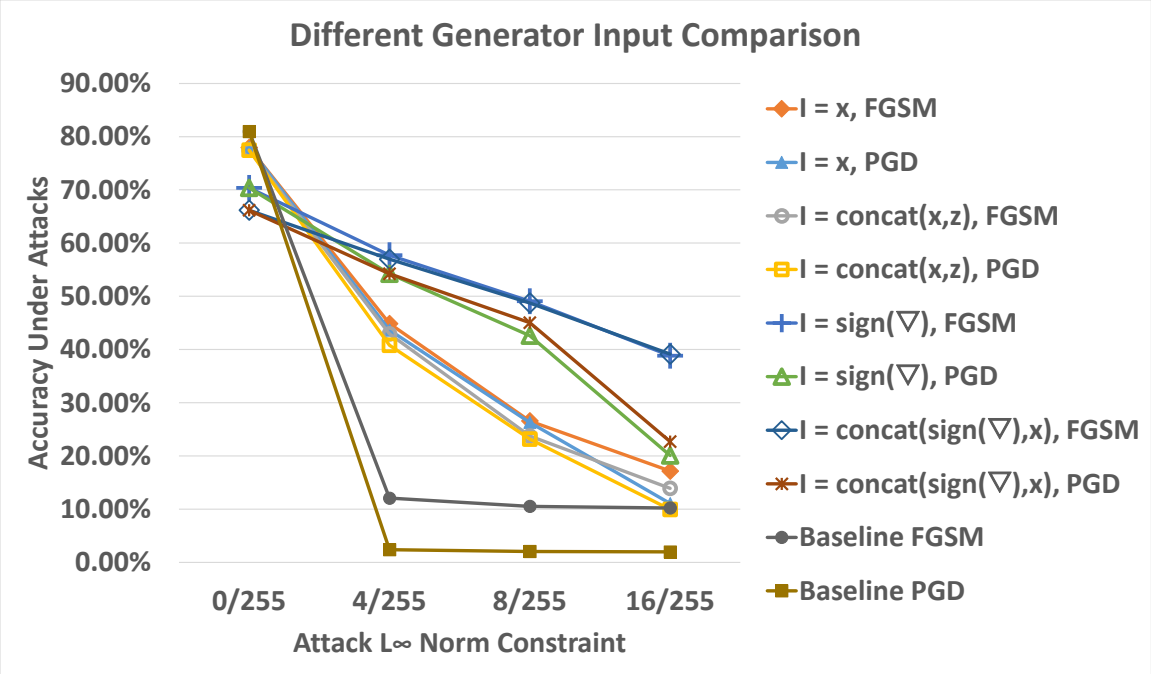

Figure 5.1: Generator input formulations comparison results.

The detailed comparison between the $I = sign(\nabla)$ and $I = concat(sign(\nabla),x)$ formulations indicates that the formulation of $I = concat(sign(\nabla),x)$ maintains more robust

accuracies. However, the $I = sign(\nabla)$ model has a lower trade-off in terms of clean accuracies (0/255), which is the performance on clean data samples. Overall, the $I = sign(\nabla)$ model involves a more simplified input formulation and high performance across all the metrics. This formulation is used for other subsequent experiments. Despite the improvement, the GAN formulation cannot achieve over 25% accuracy under 16/255 PGD attacks, indicating the potential for future improvements.

5.2.2 Width Parameter Comparison

The filter numbers for the convolution network models are important hyperparameters that can affect the performance, complexity, and capability of the model. This section presents the evaluation results regarding the number of filters within the generator model and discusses the effects of filter numbers on the augmentation performance. The following are the parameter settings used for this experiment section. The generator input formulation uses the optimal formulation found in Section 5.2.1. Other parameters are mentioned in the Implementation chapter (Chapter 4):

- GAN training parameters:
    - Generator input: $sign(\nabla)$
    - Generator width: ×1, ×2, ×3
    - Generator output constraint function: L infinity constraint with scalar ϵ 16/255
    - Trained epochs: 100
- Attack of evaluation:
    - Attack used: FGSM and PGD

- Constraint: L_∞
- Norm size: 4/255, 8/255, 16/266

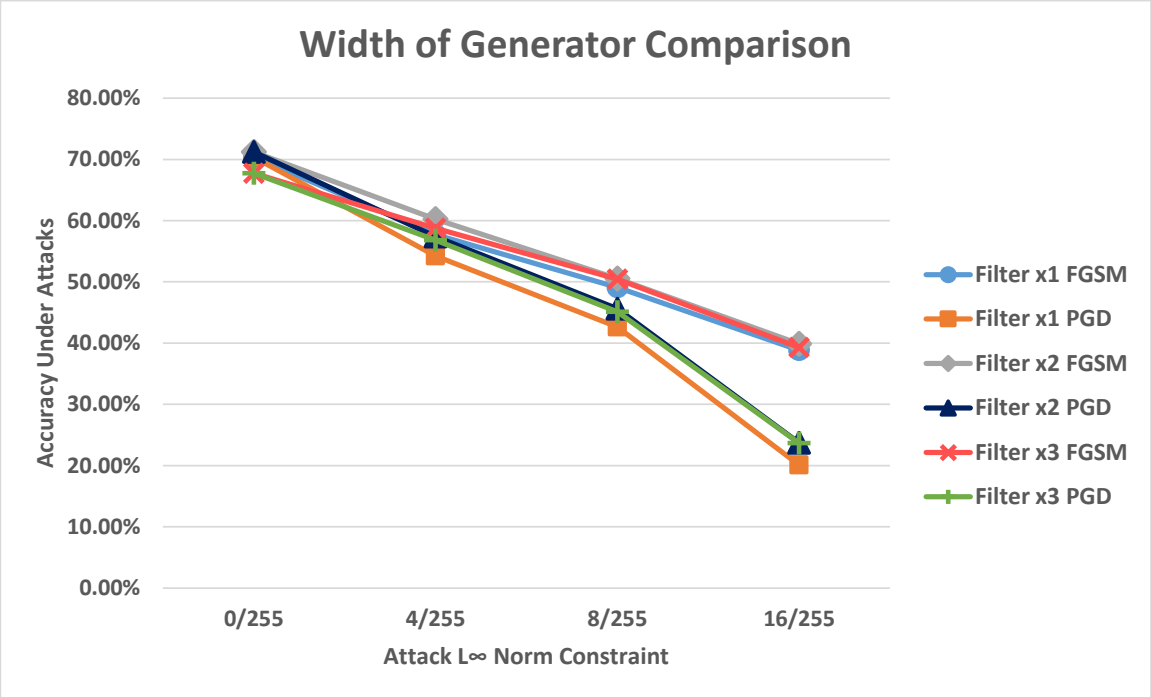

Figure 5.2: Generator width comparison results.

The filter number of the generator is controlled by a width parameter, which is defined in Chapter 4. Figure 5.2 also presents the width parameters following the "×" marks. The experiment includes three width settings, with ×1, ×2, and ×3 filters for each layer. The result indicates that the ×1 width-generator-trained classifier has the lowest robust accuracy in relation to 4/255, 8/255, and 16/255 PGD adversarial perturbation, and the ×2 and ×3 width values have similar robustness levels. The ×3-width-value-trained classifier has a lower clean accuracy. Overall, the ×2-width residual generator has the best training result in this comparison. It can be concluded that more parameters are not invariably beneficial in obtaining an overall accurate classifier. The generator with more parameters may have an increased capability of adversarial sample generation; however,

the generated sample distribution may also affect the classifier generalization on clean samples. In this case, the classifier's robustness undergoes a slight improvement after it is trained with the ×3 width GAN compared to the ×2 width; however, it overfits more into the generated sample distribution, since the clean accuracy undergoes a significant downgrade. This clean accuracy and robustness tradeoff is still present in the ×1-width GAN classifier; however, it is less significant. The experiment suggests that an adjustment of parameter numbers can be useful to control this tradeoff.

5.2.3 Training Epoch Comparison

The training epoch number is an important hyperparameter for classifier training. The classifier may underfit or overfit depending on the training epochs of the model. This section presents the results regarding three classifiers trained with the proposed GAN to compare the effects of training epochs. The training epochs for these three classifiers are 100, 200, and 300 epochs. This experiment uses the best parameters found thus far from the previous experiment, which includes the $I = sign(\nabla)$ generator formulation, and "×2" width. Other parameters are mentioned in the Implementation chapter (Chapter 4). The following are the parameter settings used for this experiment section:

- GAN training parameters:
    - Generator input: $sign(\nabla)$
    - Generator width: ×2
    - Generator output constraint function: L infinity constraint with scalar ε 16/255
    - Trained epochs: 100, 200, 300

- Attack of evaluation:
    - Attack used: FGSM and PGD
    - Constraint: L_∞
    - Norm size: 4/255, 8/255, 16/255

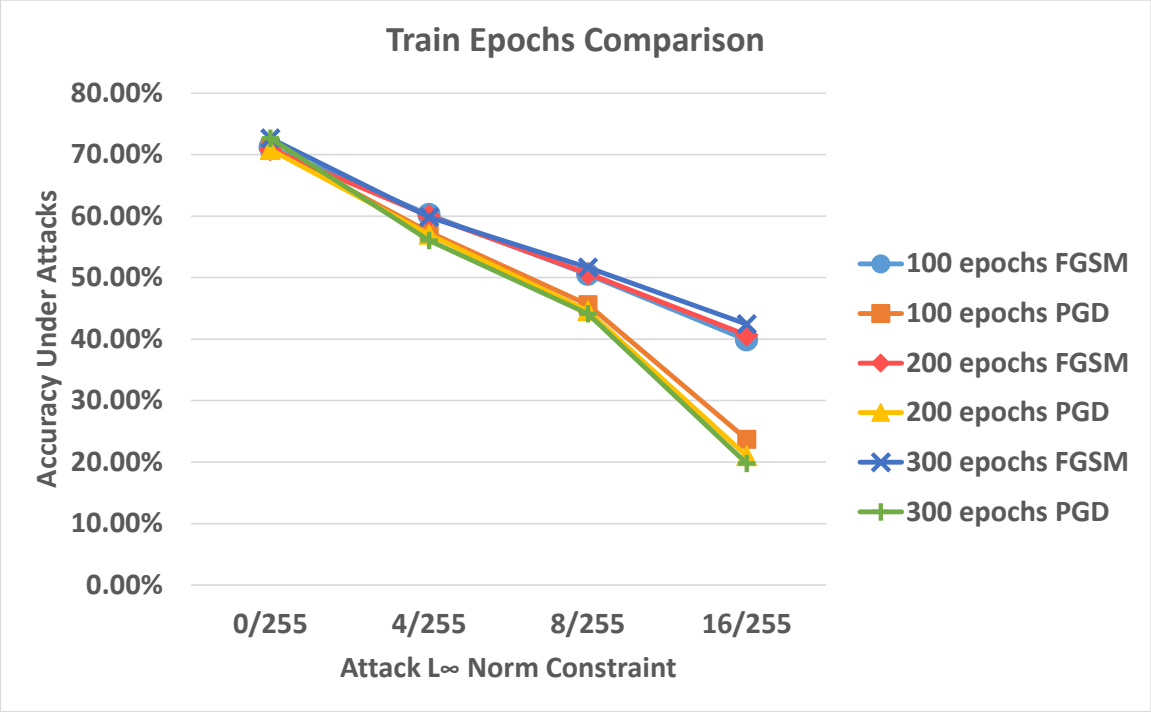

Figure 5.3: GAN training epochs comparison results.

Figure 5.3 presents the results of the comparison. Because of the FGSM's weaker attack property, the robustness accuracies against FGSM are higher than against PGD across all perturbation norms. The FGSM accuracies of the classifier indicate a slight robustness improvement with increasing training epochs. This robustness improvement against FGSM indicates that the increasing training epoch improves the generalization of the classifier on the one-step gradient perturbation. However, the PGD accuracies of the classifier start to decline as the epochs increase. This reveals an opposite effect compared to FGSM accuracies. The results suggest that GAN did not enhance the generalization of

PGD samples when the training epochs comprised more than 100 epochs. The classifier indicates a sign of overfitting that starts fitting on FGSM adversarial samples over the PGD samples. The reason for this may also relate to the overfit of the generator, where the generator starts to produce overfitted samples after a certain epoch of training. This can also explain the reason for the increase in the FGSM accuracies, which is because the generator overfits more into the simple gradient vector and produces adversarial samples similar to FGSM samples. The classifier generalizes on these flawed adversarial samples to increase only the FGSM robustness. This also suggests that the classifier's gradient landscape might become increasingly more challenging to estimate.

5.2.4 Low-Dimension Image Evaluation (MNIST)

This section presents the results of the evaluation of the MNIST test dataset. This experiment aims to validate the implementation of the L_∞ GAN with a low-dimensional dataset. The following are the parameter settings used for this experiment section. The generator width for this experiment is reduced back to "×1" and the training epoch value is set to be 10 since the data dimension is lower and more parameters and more training is not necessary to obtain an accurate result. Other parameters are the optimal parameter found from previous sections or mentioned in the Implementation chapter (Chapter 4):

- GAN training parameters:
    - Generator input: *sign(▽)*
    - Generator width: ×1
    - Generator output constraint function: L_∞ constraint with scalar ε 0.3
    - Trained epochs: 10
- Attack of evaluation:

- Attack used: FGSM and PGD
- Constraint: L_∞
- Norm size: 0.1 (25.5/255), 0.2 (51/255), 0.3 (76.5/255)

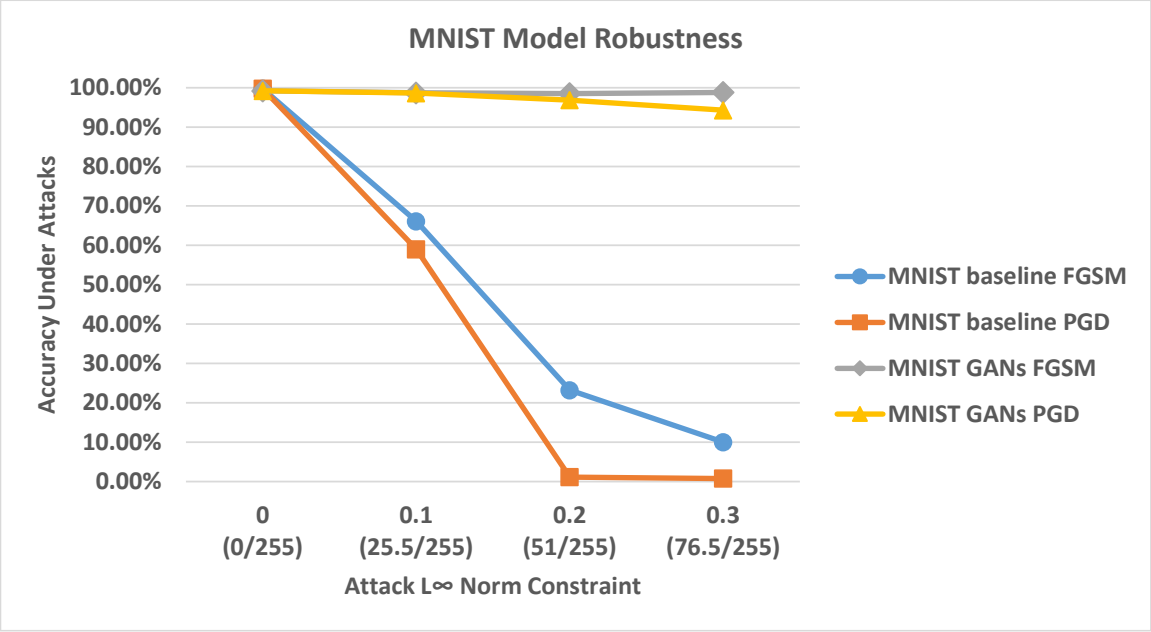

Figure 5.4: MNIST experiment results.

One classifier model is trained with the proposed GAN model, and one baseline model with identical parameters is trained without any defenses. The accuracies under different attack norms are reported under the L_∞ norm constraint. The MNIST dataset is a simple dataset that, with previously developed defensive adversarial training, can achieve high robustness results against L_∞ iterative gradient attacks. However, the proposed GAN model has the advantage of reducing the training complexity of the entire training process, which only requires several additional backward passes to update the generator parameters rather than multiple gradient descents. The classifier accuracies remain above 90% for all attack settings and exhibit significant improvements under a strong PGD attack algorithm. The results indicate that the GAN augmentation has a strong performance with small

convolution models and can achieve a more consistent performance compared to CIFAR 10 results. It also suggests that reduced dimensions can lower the difficulty of the generalization of the classifier and provide it with a much more reliable robustness increase against the attack algorithms. However, the model can be challenging to implement for a higher-dimension dataset in the future. Figure 5.4 presents the results. The x-axis is represented in two value formats. The previous research [3], [16] commonly uses 0.1, 0.2, and 0.3 as standard metrics for this dataset. Hence, the two value formats can provide consistency with previous studies and the standard format used in this thesis.

5.3 L_2 GAN

This section presents the L_2 GAN experiments and evaluations. The purpose of the L_2 GAN implementation and experiments is to evaluate whether GAN can be more effective in augmenting L_2 robustness compared to conventional adversarial training and whether L_2 GAN can also provide the robustness in the L_∞ constraint, which refers to the transferability of the robustness. To make each section easy to keep track of, the basic parameter settings to train the classifiers are listed within each section before the evaluation.

5.3.1 L_2 Robustness

This section presents the training results of the L_2 constraint generator augmentation. The generators used in this experiment are implemented using the L_2 output constraint function. This experiment aims to demonstrate the training effect with the proposed L_2 adversarial sample augmentation on adversarial robustness against L_2 gradient attacks. This section also compares the proposed GAN training method with the traditional PGD adversarial training method with the L_2 constraint. The following are the parameter settings used for this experiment section. The generator input formulation uses the optimal formulation

found in Section 5.2.1. Other parameters are mentioned in the Implementation chapter (Chapter 4):

- GAN training parameters:
    - Generator input: *sign(▽)*
    - Generator width: N/A
    - Generator output constraint function: L_2 constraint with scalar ε 64/255
    - Trained epochs: 100
- Attack of evaluation:
    - Attack used: PGD
    - Constraint: L_2
    - Norm size: 16/255, 32/255, 64/255, 128/255

When the PGD adversarial attack is applied to the L_2 norm constraint, the algorithm may face the issue of gradient masking, wherein the gradient descent can only find the local optimal perturbation vector. This phenomenon may also affect its effectiveness in adversarial training. The proposed GAN applies a fixed norm size perturbation vector in the L_2 norm space, and this solution can leverage the gradient masking issue and can enable better data augmentation compared to a traditional gradient descent. The U-net architecture can also be helpful with its long-range skip connection for easier gradient backpropagation. Figure 5.5 presents the results of the comparison. The classifier trained with L_2 PGD adversarial training is named after PGD L_2 AT. The GAN-trained classifier performs significantly better than other classifiers. The L_2 robustness accuracies remain above 60% across all attack norm sizes with PGD L_2 attacks. The parallel comparison between the GAN-trained classifier and PGD L_2 AT suggests that GAN can be more effective in this

constraint. However, the clean accuracy at the 0/255 norm size is significantly lower than the baseline, indicating that the classifier cannot provide a high-quality generalization on both normal and L_2 adversarial samples at the same time. This clean accuracy and robustness tradeoff is consistent with the L_∞ results.

In addition, the L_2 PGD attack is not effective itself to generate L_2 constrained adversarial samples. Hence this comparison cannot fully conclude the L_2 robustness of these models. The results of this experiment mostly demonstrate the comparison between the GAN L_2 model and the conventional adversarial training model and suggest that the GAN L_2 model can be more advantageous to trained under L_2 constraint.

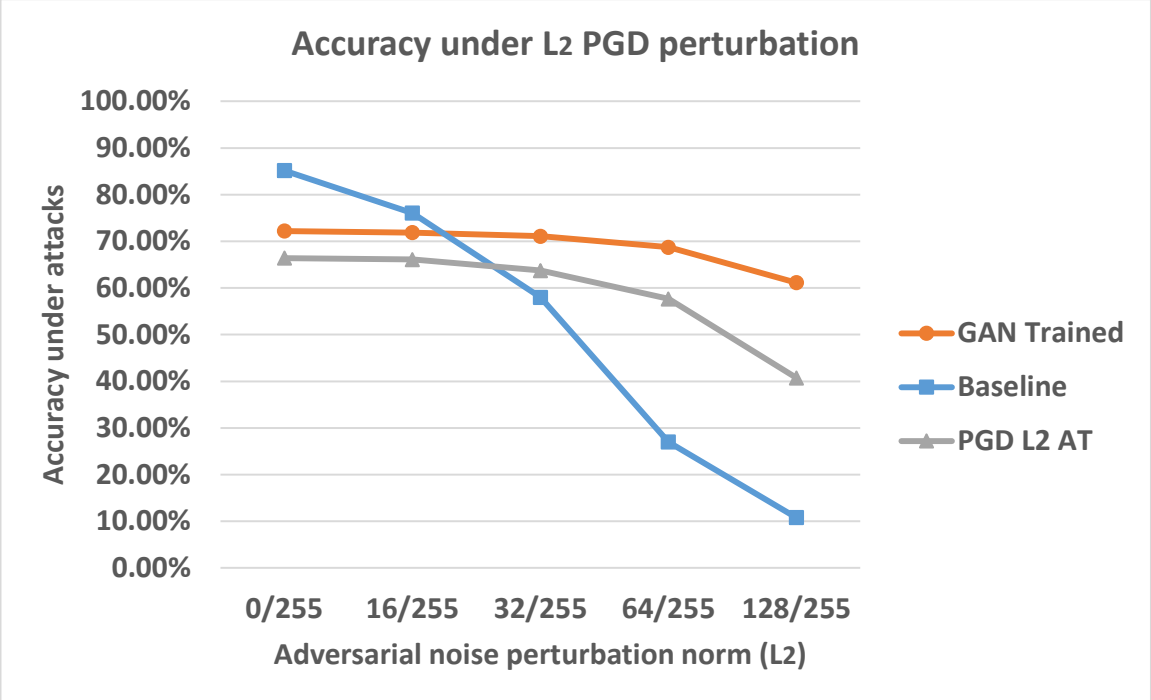

Figure 5.5: Training results under L_2 constraint against L_2 PGD attack.

5.3.2 Robustness Transferability

This section presents the results from the L_∞ robustness tests with the same classifier from the last section. The purpose of this experiment is to evaluate the L_∞ robustness of a

classifier trained with GAN under the L_2 setting. The following are the parameter settings used for this experiment section. The generator input formulation uses the optimal formulation found in Section 5.2.1. Other parameters are mentioned in the Implementation chapter (Chapter 4):

- GAN training parameters:
    - Generator input: *sign(▽)*
    - Generator width: N/A
    - Generator output constraint function: L_2 constraint with scalar ε 64/255
    - Trained epochs: 100
- Attack of evaluation:
    - Attack used: FGSM and PGD
    - Constraint: L_∞
    - Norm size: 4/255, 8/255, 16/266

Under normal circumstances, the classifier trained with a constraint or attack type of adversarial training results in limited robustness against different constraints or attack types. However, the proposed L_2 and L_∞ GAN models' most architectural designs and methodologies are consistent with each other. This experiment can demonstrate the transferability of the robustness between the L_2 and L_∞ GAN augmentation.

Figure 5.6 presents the comparison results regarding three classifiers against different norm sizes of L_∞ PGD attacks. Two of the classifiers are trained with GAN architecture and selected from the highest-performing parameters. With the comparison, the L_∞ GAN and L_2 GAN-trained classifiers exhibit a similar performance against the L_∞

PGD attack. The results indicate that there is some transferability within the robustness between the different constraints. Under some perturbation sizes, the L_2 GAN classifier appears to exhibit better robustness against the L_∞ PGD attack; however, the scalar used during the training for L_2 GAN is 64/255, which is substantially larger than L_∞ GAN's 16/255. It suggests equilibrium between these value constraints.

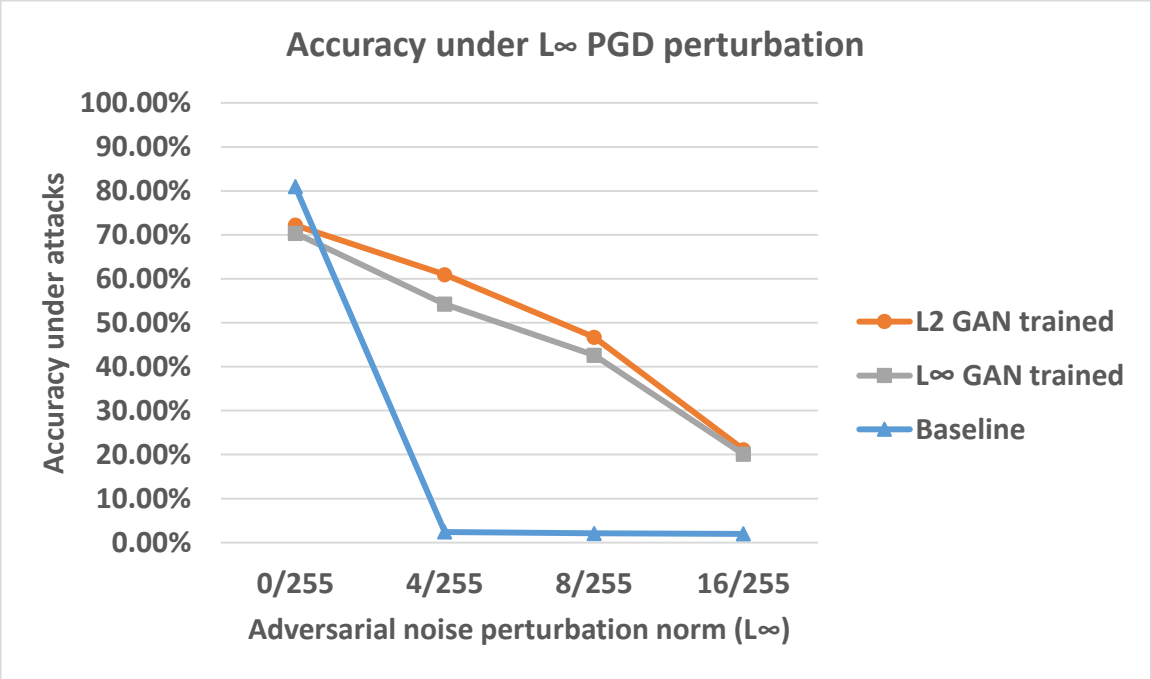

Figure 5.6: L_∞ robustness comparison between different constraint training results.

5.4 Summary of Results

This section provides the combined results across the models in terms of the L_∞ PGD attacks' accuracies. Figure 5.7 presents a parallel comparison over the maximum adversarial perturbation norm size (16/255). The PGD L_2 attack is not included in this evaluation because the PGD L_2 attack will suffer from the previously mentioned gradient issues [5]. The PGD L_∞ can demonstrate a more valid conclusion about the adversarial robustness of gradient attacks.

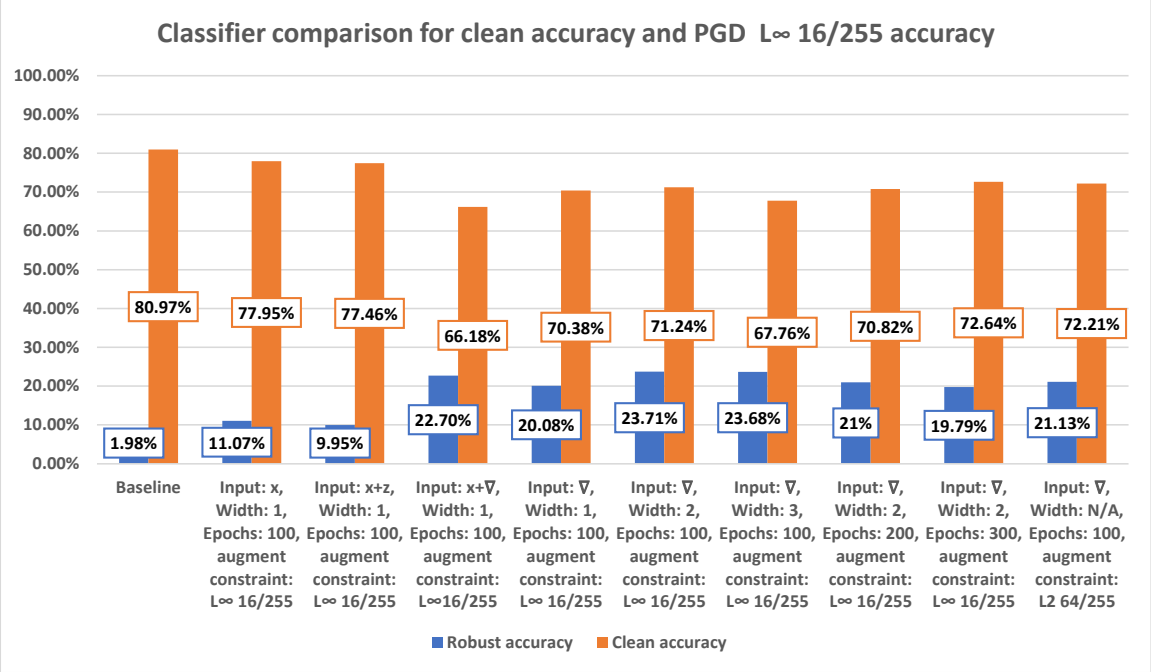

Figure 5.7: Comparison of the accuracies between all formulations.

Across the accuracies, the classifier beside the baseline exhibits lower clean accuracies, indicating the robustness and accuracy trade-offs. The generalization of the adversarial distributional shifts cannot be completely resolved by the proposed model. However, the model helps to improve the overall adversarial robustness and provides an option in the application when the robustness is required and is more important than the clean accuracy. Based on Figure 5.7, the highest-performing classifiers are associated with the GAN with the following settings:

- Input: *sign(∇)*, width 2, epochs 100, augmentation constraint: L_∞ 16/255,
- Input: *sign(∇)*, width N/A, epochs 100, augmentation constraint: L_2 64/255,

These models have a better balance between clean and robust accuracies compared to the others. The results indicate that the generator formulations, parameters, and training epochs can affect the training performance of GAN and affect both clean accuracies and

robustness. The constraint of augmentation can be flexible in providing overall robustness against gradient attacks, but the norm size needs to be considered along with the constraint.

5.5 Visualization

This section provides an additional interpretation of the adversarial samples from the generator of the GAN. The following are the parameter settings of the selected model used for this interpretation:

- GAN training parameters:
    - Generator input: *sign(▽)*
    - Generator width: ×2
    - Generator output constraint function: L_∞ constraint with scalar ϵ 16/255
    - Trained epochs: 300

This model is not the best-performing model regarding the training epochs. However, more training epochs make the comparison between classifier outputs more obvious; hence, this model has been selected for this section.

Figure 5.8 presents one of the testing samples and its adversarial perturbation, as well as its classifier outputs. Image z) is an example image from the dataset. The images a), b), c), and d) represent different perturbation vectors that are used to perturb the image. These perturbation vectors are generated by using different corresponding algorithms and by being normalized so that all of the perturbation vectors have a mean of zero and a standard distribution of one. The e), f), g), and h) are the classifier pre-SoftMax outputs regarding the different levels of perturbation. The x-axes of the e), f), g), and h) plots indicate the norm size of the perturbation. During the visualization, the value of the x-axis of the plots

is multiplied with the a), b), c), and d) vectors and added to the image z). The additional results are fed into the classifier to generate the plots e), f), g), and h), accordingly. The formula used for producing the perturbed images is given as Equation (5.2):

$$I' = I + \epsilon V \tag{5.2}$$

where I' is the result of addition, and ϵ is the value from the plots' x-axes, and the V is the selected perturbation vector.

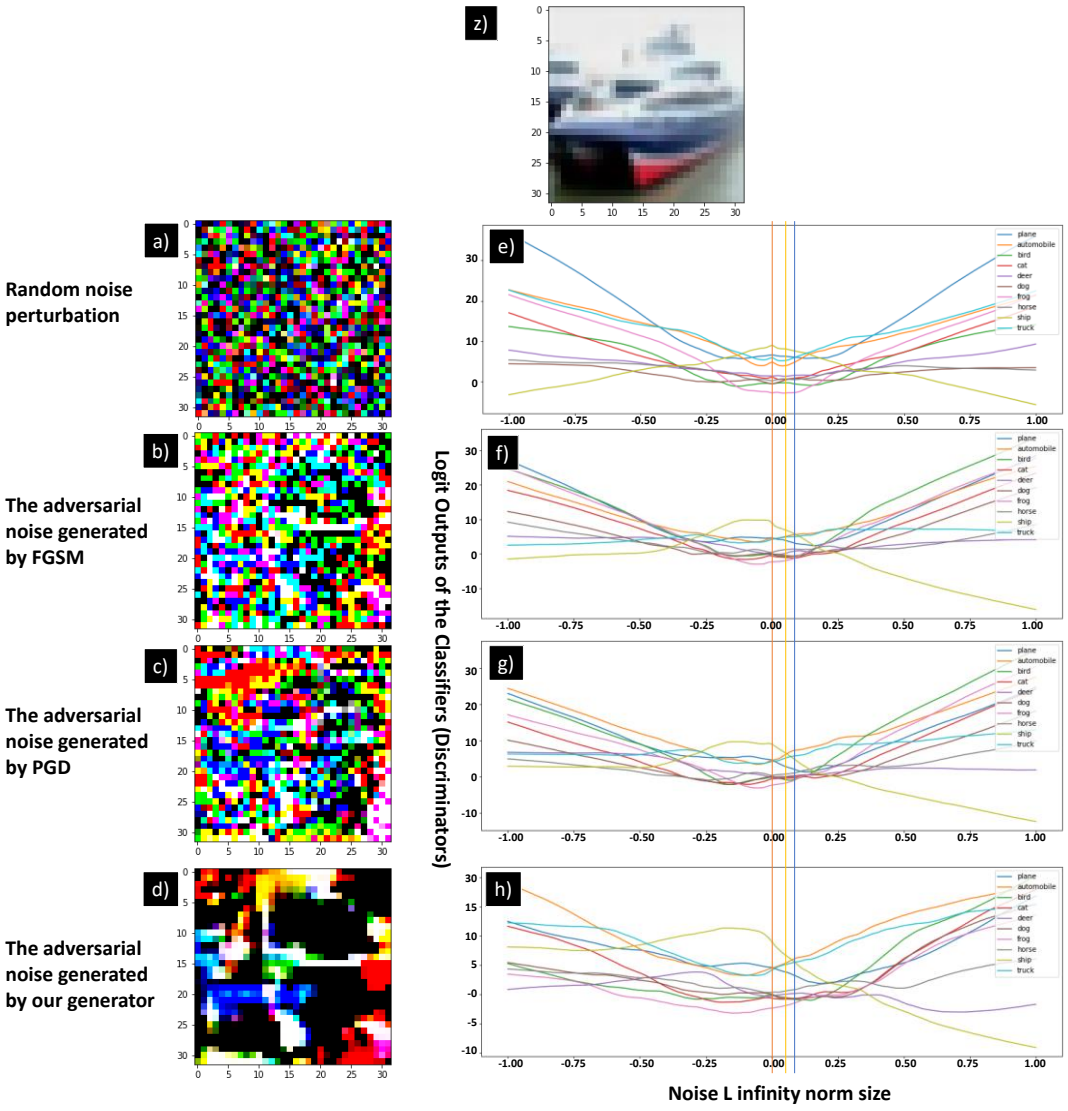

Figure 5.8: Extra visualization.

From the plots in Figure 5.8, it is obvious that the classifier exhibits more robust behavior when the image z) is perturbed by vector a). The shape of the correct class activation of the pre-SoftMax layer has a sufficiently wide margin until the wrong class activation surpasses it. However, if the image z) is perturbed by vectors b), c), and d), the shape of the activation of the correct class becomes steeper towards the positive side. The points where an erroneous class activation becomes larger than the correct class are labeled with different color lines; the orange line labels the middle where no vector is added; the blue line represents the point of surpassing caused by the FGSM vector; and the yellow line represents the point of surpassing caused by the PGD vector. From the observation, the point of surpassing caused by the generator-produced vector is between the points of the FGSM and PGD. This suggests that the generator captured the adversarial perturbation direction successfully; however, the captured direction is not as effective as the PGD algorithm.

5.6 Discussion

This thesis demonstrates a defensive training strategy against the gradient-based adversarial attack based on the GAN architecture. The experimental results indicate that the proposed GAN can improve the adversarial robustness of a convolution classifier; however, the robustness improvement is related to the GAN's input selection, parameter numbers, and the training epochs. The results suggest that the gradient information of the classifier is the most important feature for the generator to estimate the classifier's worst-case loss. The generator without this gradient information has a diminished ability to provide valid augmentation for the classifier, which results in lower adversarial robustness after the training. The width parameter also has a slight impact on the final results. A more

complex generator is not always better to train a robust and accurate model. Finally, the training epochs can also affect the model's performance after training. It is possible to over-train the model, causing the GAN to overfit on specific data distributions and resulting in lower adversarial robustness against iterative gradient-based attacks.

The L_2 constraint experiments demonstrate the effectiveness of the GAN in estimating and improving the L_2 constraint's adversarial robustness. The GAN-trained classifier shows a larger improvement in the L_2 constraint robustness against gradient attacks compared to conventional adversarial training. This suggests that the proposed GAN can mitigate the gradient masking problem of the L_2 gradient-based adversarial training. The other finding is that the classifiers trained with the L_∞ and L_2 constraint shared a similar robustness against L_∞ PGD attacks. This finding indicates that there is some transferability of robustness across the constraint types.

The limitation remains for the generalization capabilities of the proposed GAN model. In all experiments, the robustness classifiers have their clean accuracies decrease around 10% to 15%. The results indicate a clean accuracy and robustness tradeoff within all GAN-trained classifiers. The tradeoff suggests that these classifiers overfit into the generated samples from the generator and it is challenging for the classifier to generalize on both generated and clean sample distributions. By controlling the parameter numbers of the generator can reduce this overfit but the tradeoff cannot be completely mitigated. Furthermore, the generator can also overfits to produce not precise outputs with more training epochs. This generator overfitting effect leads to reduction in the augmentation performance and results in the robustness decline of the classifier with increasing training iterations. The other challenges are related to the implementation of the GAN architecture

that the current GAN cannot provide augmentation under L_0 and L_1 constraints. New constraint functions are required to realize the functionalities.

Ultimately, the visualization presents the additional information about the generator-generated adversarial sample. It suggests that the adversarial noises captured by the generator are indeed different than the ones produced using gradient-based algorithms. There is still a limitation of GAN in terms of finding the worst adversarial sample of the classifier. However, in the MNIST experiment, this limitation is not significant when the classifier remains accurate across different attack norms. Hence, the data dimension matters in the GAN optimization. The current implementation of GAN is more favorable to train a low-dimension convolution classifier.

5.7 Threats to Validity

The threat to validity includes internal and external validity threats that could challenge the implementation of the GAN models. The internal validity threats include the internal factors that have to be considered during the implementation and evaluation. The external validity threat includes the outside factors when the GAN model is applied to other datasets or real-life scenarios. The following identify the internal validity threats that affect the implementation and evaluation of the thesis:

- The parameter settings for the GAN model can depend upon the classifier architecture types and the parameters. The classifier used in this thesis is a VGG-like convolution network, which can only represent the general performance of convolution neural networks. The other architecture, such as the transformer models [97], can define a different gradient backpropagation and result in different robustness improvements. Some of the architecture may have some compatibility

issues with the current implementations, which is one of the limitations of the research. However, the proposed model is flexibly modified, and it can be implemented with different network architectures, but more experiments should be conducted to identify the optimal hyperparameters.

- The robustness improvement of the model depends upon the random initialization of the parameters of the models. The current model parameter initialization uses the traditional random initialization method. The random initialization of the parameters can result in accuracy variance during the evaluation. To address this, multiple models have been trained within one experiment, and the best one is used for the evaluation; hence, the real results can reflect the upper-bound performance of the GAN training.
- The PGD attack used in the evaluation represents a multi-iteration gradient attack. The iteration number of the PGD used in most of the evaluations is 100 to ensure a balance between an effective evaluation and the computation time. However, these attacks could be used with more iterations, which could result in lower accuracies. Hence, a PGD 1000 iteration attack is implemented and attacks on the best-performing model with the following parameter settings:
    - Input: $sign(\nabla)$
    - Width: 2
    - Output constraint: L_∞ with scalar ε 16/255

Table 5.1 presents this additional evaluation. The result suggests that the performance of the model did not differ much between the 100 and 1000 iterations.

It suggests that PGD 100 is valid; however, there can be a slight difference in performance when using different iteration values.

Table 5.1: PGD 1000 validation.

L_∞ norm size	**4/255**	**8/255**	**16/255**
Accuracy PGD 100	57.41%	45.6%	23.71%
Accuracy PGD 1000	57.33%	45.28%	23.31%

The following discusses the external validity of the study, which may affect the GAN implementation in different scenarios:

- The dataset can be a factor that affects the model's performance. This thesis mostly uses a 32×32×3 dimensionality of the image in the RGB color channel to perform the evaluation; however, more dimensionalities can affect the training results regarding the generalization and the robustness of the classifier.
- The attack type used focused by this thesis is the gradient-based algorithm using gradient descent to perform the attack. The other attack algorithms using different optimizers, such as genetic algorithms, can result in different robustness results.
- The vulnerability of the machine learning model can also include the data poisoning attack and other attack types. These attacks are not evaluated by this paper. There are other cases in which the classifier can also provide poor generalization, such as a natural distributional shift of the data. These distributional shifts may need to be addressed in future research.

5.8 Summary

The chapter has summarized the results from the evaluation of all classifiers. The two datasets are used and evaluated with two gradient-based attack algorithms on the GAN-trained classifiers. The evaluation results report the training effectiveness of GAN with different considerations of input formulations, width, training epochs, and constraint types. The experiments demonstrate how GAN can improve the adversarial robustness of the models and the limitations.

Chapter 6. Conclusion and Future Work

6.1 Conclusion

This thesis presents a method that uses GAN formulation to improve the adversarial robustness of deep learning convolution classifiers. The model uses a generative model and adversarial training techniques to generate an adversarial sample and to perform data augmentation on the generated adversarial sample. The classifier adversarially-trained with the generative model can improve the adversarial robustness against a gradient-based adversarial attack, such as FGSM and PGD attacks. The GAN represents a possible defensive strategy against such attacks.

In this thesis, several studies are presented to compare the effectiveness of different components and settings of the proposed GAN. The results indicate that the improvement of the adversarial robustness is related to the assumption of the generator formulation. The formulation includes more gradient information from the classifier, which can result in a better training result. The GAN model can provide over 20% robust accuracy improvement with the strongest attack implemented, with a 10% clean accuracy tradeoff. Furthermore, the GAN model can resist the gradient-based attack for small-image classifiers, such as the MNIST classifier. It suggests that with a lower-dimensional dataset, such as grayscale images, GAN can be useful to defend against adversarial attacks.

The primary limitation of the GAN adversarial training is the upper limit of the generalization. The generalization refers to both the generator generalization and the classifier generalization. According to the experimental results, the classifier model has a significant trade-off between the robustness and clean accuracy, and it indicates the difficulty for the classifier to generalize on both adversarial data distribution and clean data

distribution at the same time. The generator generalization is limited by its capability to discover adversarial samples. Compared to the PGD algorithm, the generator-generated adversarial samples are significantly different in visualization and are less effective than PGD adversarial samples. This represents another tradeoff between the algorithm complexity and the attack effectiveness. Another limitation is that the current formulations cannot be applied to L_0 and L_1 constraints since the generator cannot generate adversarial sample with these constraints. New constraint functions must be implemented to realize the augmentation.

This thesis proposed a GAN architecture to realize low-complexity adversarial training for defending adversarial samples from gradient-based attacks. The significance of the works includes the proposal of the different formulations and implementations of GAN to address the problems of adversarial samples from L_∞ and L_2 constraint gradient-based attacks and suggesting optimal formulation of GAN on defending these attacks. A further evaluation also reveals the relationship between the training epoch and the performance of the model and demonstrates the transferability of the robustness between constraints. Additional visualization is also provided to illustrate the differences between algorithms generated adversarial samples and the GAN's estimations. The thesis provides an extension solution of adversarial training and introduced GAN to this domain of application. Compared to other GAN solutions, the proposed architecture is implemented with more advanced formulations and under variety constraints, which as a result, provides more insight into the optimal GAN for adversarial training.

6.2 Future Work

This section discusses the possible future direction of the GAN, adversarial training and the solutions to data distributional shift problems. An extension of the dataset can be helpful for improving the generalization of the classifier. More aggressive data augmentation methods can be utilized to overcome the overfitting effect. Transfer learning can also be considered to perform the training on more data samples or more advanced datasets such as the ImageNet dataset. With a more advanced dataset, the classifier's performance can be improved compared to the current results.

More advanced model architecture can be considered, such as the Vision Transformer model [97]. The Transformer model can be used as a more advanced classifier. The Transformer model can also provide additional benefits with attention-based weights, which can provide more interpretation about the generalization.

A new cost function can be improved to make the equilibrium point of GAN easier to achieve. This thesis uses KL divergence, which has been found to have issues with gradient saturation. Recently, more advanced divergence functions have been proposed for better GAN performance. These functions can be adapted to fit the training goal of the proposed model.

Another limitation of the research is the limited evaluation of datasets type within this thesis. The image dataset only represents one of the possible scenarios of the deep learning application. The other data types, such as sequence data and text data, have their own deep learning model implementation and design. The adversarial attacks that target other types of deep learning models can be different in their optimization methodologies and operate under different constraints. More specifically, the L_2 and L_∞ are primarily

designed to target image data. The thesis can only provide an overview of convolution architectures with the data type that is attackable within L_2 and L_∞ constraints. The proposed GAN defense needs to be adapted to other constraints and model design to be able to provide effective defense for other applications.

Extensive research is also needed to evaluate the performance of other datasets with different dimensionalities. The proposed GAN only evaluated against the adversarial sample with 32×32×3 and 28×28×1 dimensions; however, there are more datasets available with higher dimensions. The study can be used to determine the relationship between the robustness performance and the dimensionalities of the data sample.

Mover over, the proposed defense only provides the evaluation of robustness against two gradient-based attack algorithms, FGSM and PGD. The other attacks include different methodologies for optimizing and generating adversarial samples. These other optimization strategies can result in different distributions within the generated adversarial samples, and the proposed defense may have different performance against adversarial samples with different optimization. Hence more research is required to evaluate the defense against other attack algorithms.

Furthermore, the generalization of distribution-shifted data has been a common issue in relation to deep learning models. Interpretation can be pivotal to find the reason for these generalization failures. A new interpretable deep learning model architecture can be used to provide more useful information regarding the model's decision-making. One of the suggested architectures is a self-interpretable model that defines both learning goals for classification and interpretation. Such a model can be used in data distributional shift

research to provide a feedback loop regarding model design. The proposed model has the following layer formulation as Equation (6.1):

$$O_t = x \times f_t(x, \theta) \tag{6.1}$$

This layer formulation includes the layer input sample *x* and the layer network f_t. The θ represents the layer-trainable parameters. The benefit of this formulation is that the layer f_t needs to output a vector that is multiplied with the input *x*. The multiplication promotes the output of f_t aligning with the input *x*. The *x* and output vector of *ft* can represent a set of dynamic weights that promotes the most important features within the sample *x*. Appendix B: Example Feature Maps of Proposed Interpretable Model presents the feature map examples of the interpretable model. Within these feature maps, more obvious features are presented that can provide a deeper explanation of the model's decision making. The model can be used to improve the research regarding the adversarial sample.

Bibliography

[1] S. H. Silva and P. Najafirad, "Opportunities and Challenges in Deep Learning Adversarial Robustness: A Survey." *arXiv*, Jul. 03, 2020. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/2007.00753

[2] C. Szegedy et al., "Intriguing properties of neural networks." *arXiv*, Feb. 19, 2014. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/1312.6199

[3] Y. Li, M. Cheng, C.-J. Hsieh, and T. C. M. Lee, "A Review of Adversarial Attack and Defense for Classification Methods," *The American Statistician*, vol. 76, no. 4, pp. 329–345, Oct. 2022, doi: 10.1080/00031305.2021.2006781.

[4] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples." *arXiv*, Mar. 20, 2015. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/1412.6572

[5] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards Deep Learning Models Resistant to Adversarial Attacks." *arXiv*, Sep. 04, 2019. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/1706.06083

[6] J. Chen, M. Su, S. Shen, H. Xiong, and H. Zheng, "POBA-GA: Perturbation optimized black-box adversarial attacks via genetic algorithm," *Computers & Security*, vol. 85, pp. 89–106, Aug. 2019, doi: 10.1016/j.cose.2019.04.014.

[7] W. Xu, D. Evans, and Y. Qi, "Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks," in *Proceedings 2018 Network and Distributed System Security Symposium*, 2018. Doi: 10.14722/ndss.2018.23198.

[8] F. Liao, M. Liang, Y. Dong, T. Pang, X. Hu, and J. Zhu, "Defense Against Adversarial Attacks Using High-Level Representation Guided Denoiser," 2018, pp. 1778–1787. Accessed: Jan. 26, 2023. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2018/html/Liao_Defense_Against_Adversarial_CVPR_2018_paper.html

[9] P. Samangouei, M. Kabkab, and R. Chellappa, "Defense-GAN: Protecting Classifiers Against Adversarial Attacks Using Generative Models." *arXiv*, May 17, 2018. Doi: 10.48550/arXiv.1805.06605.

[10] J. Wang, Z. Lyu, D. Lin, B. Dai, and H. Fu, "Guided Diffusion Model for Adversarial Purification." *arXiv*, Jun. 28, 2022. Doi: 10.48550/arXiv.2205.14969.

[11] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana, "Certified Robustness to Adversarial Examples with Differential Privacy." *arXiv*, May 29, 2019. Doi: 10.48550/arXiv.1802.03471.

[12] T. Bai, J. Luo, J. Zhao, B. Wen, and Q. Wang, "Recent Advances in Adversarial Training for Adversarial Robustness." *arXiv*, Apr. 20, 2021. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/2102.01356

[13] N. Carlini and D. Wagner, "Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, New York, NY, USA, Nov. 2017, pp. 3–14. Doi: 10.1145/3128572.3140444.

[14] C. Bowles et al., "GAN Augmentation: Augmenting Training Data using Generative Adversarial Networks." *arXiv*, Oct. 25, 2018. Doi: 10.48550/arXiv.1810.10863.

[15] C. Xiao, B. Li, J.-Y. Zhu, W. He, M. Liu, and D. Song, "Generating Adversarial Examples with Adversarial Networks." *arXiv*, Feb. 14, 2019. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/1801.02610

[16] H. Wang and C.-N. Yu, "A Direct Approach to Robust Deep Learning Using Adversarial Networks." *arXiv*, May 23, 2019. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/1905.09591

[17] W. Zhao, S. Alwidian, and Q. H. Mahmoud, "Adversarial Training Methods for Deep Learning: A Systematic Review," *Algorithms*, vol. 15, no. 8, p. 283, Aug. 2022, doi: 10.3390/a15080283.

[18] W. Zhao, S. Alwidian, and Q. H. Mahmoud, "Evaluation of GAN Architectures for Adversarial Robustness of Convolution Classifier" in *The AAAI-23 Workshop on Artificial Intelligence Safety (SafeAI 2023)*, Washington, DC, Feb 2023.

[19] W. Zhao, Q. H. Mahmoud, and S. Alwidian, "Evaluation of GAN-based Adversarial Training" *Sensors*, vol. 23, no. 5, p. 2697, Jan. 2023, doi: 10.3390/s23052697.

[20] P. P. Shinde and S. Shah, "A Review of Machine Learning and Deep Learning Applications," in *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, Aug. 2018, pp. 1–6. Doi: 10.1109/ICCUBEA.2018.8697857.

[21] I. Goodfellow et al., "Generative adversarial networks," *Commun. ACM*, vol. 63, no. 11, pp. 139–144, Oct. 2020, doi: 10.1145/3422622.

[22] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble Adversarial Training: Attacks and Defenses." *arXiv*, Apr. 26, 2020. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/1705.07204

[23] B. S. Vivek and R. Venkatesh Babu, "Single-Step Adversarial Training With Dropout Scheduling," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2020, pp. 947–956. Doi: 10.1109/CVPR42600.2020.00103.

[24] T. Huang, V. Menkovski, Y. Pei, and M. Pechenizkiy, "Bridging the Performance Gap between FGSM and PGD Adversarial Training." *arXiv*, Oct. 03, 2022. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/2011.05157

[25] G. Liu, I. Khalil, and A. Khreishah, "Using Single-Step Adversarial Training to Defend Iterative Adversarial Examples," in *Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy*, New York, NY, USA, Apr. 2021, pp. 17–27. Doi: 10.1145/3422337.3447841.

[26] E. Wong, L. Rice, and J. Z. Kolter, "Fast is better than free: Revisiting adversarial training." *arXiv*, Jan. 12, 2020. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/2001.03994

[27] M. Andriushchenko and N. Flammarion, "Understanding and Improving Fast Adversarial Training," in *Advances in Neural Information Processing Systems*, 2020, vol. 33, pp. 16048–16059. Accessed: Jan. 26, 2023. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/b8ce47761ed7b3b6f48b583350b7f9e4-Abstract.html

[28] H. Kim, W. Lee, and J. Lee, "Understanding Catastrophic Overfitting in Single-step Adversarial Training," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 9, pp. 8119–8127, May 2021, doi: 10.1609/aaai.v35i9.16989.
[29] C. Song, K. He, L. Wang, and J. E. Hopcroft, "Improving the Generalization of Adversarial Training with Domain Adaptation." *arXiv*, Mar. 15, 2019. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/1810.00740
[30] B. S. Vivek and R. V. Babu, "Regularizers for Single-step Adversarial Training." *arXiv*, Feb. 03, 2020. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/2002.00614
[31] B. Li, S. Wang, S. Jana, and L. Carin, "Towards Understanding Fast Adversarial Training." *arXiv*, Jun. 04, 2020. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/2006.03089
[32] J. Yuan and Z. He, "Adversarial Dual Network Learning With Randomized Image Transform for Restoring Attacked Images," *IEEE Access*, vol. 8, pp. 22617–22624, 2020, doi: 10.1109/ACCESS.2020.2969288.
[33] W. Wan, J. Chen, and M.-H. Yang, "Adversarial Training with Bi-directional Likelihood Regularization for Visual Classification," in *Computer Vision – ECCV 2020*, Cham, 2020, pp. 785–800. Doi: 10.1007/978-3-030-58586-0_46.
[34] Y. Qin, R. Hunt, and C. Yue, "On Improving the Effectiveness of Adversarial Training," in *Proceedings of the ACM International Workshop on Security and Privacy Analytics*, New York, NY, USA, Mar. 2019, pp. 5–13. Doi: 10.1145/3309182.3309190.
[35] A. Laugros, A. Caplier, and M. Ospici, "Addressing Neural Network Robustness with Mixup and Targeted Labeling Adversarial Training," in *Computer Vision – ECCV 2020 Workshops, Cham*, 2020, pp. 178–195. Doi: 10.1007/978-3-030-68238-5_14.
[36] W. Li, L. Wang, X. Zhang, J. Huo, Y. Gao, and J. Luo, "Defensive Few-shot Adversarial Learning." *arXiv*, Nov. 16, 2019. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/1911.06968
[37] J. Liu and Y. Jin, "Evolving Hyperparameters for Training Deep Neural Networks against Adversarial Attacks," in *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, Dec. 2019, pp. 1778–1785. Doi: 10.1109/SSCI44817.2019.9002854.
[38] Z. Ren, A. Baird, J. Han, Z. Zhang, and B. Schuller, "Generating and Protecting Against Adversarial Attacks for Deep Speech-Based Emotion Recognition Models," in *ICASSP 2020 – 2020 IEEE International Conference on Acoustics*, Speech and Signal Processing (ICASSP), May 2020, pp. 7184–7188. Doi: 10.1109/ICASSP40776.2020.9054087.
[39] C. Song et al., "MAT: A Multi-strength Adversarial Training Method to Mitigate Adversarial Attacks," in *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Jul. 2018, pp. 476–481. Doi: 10.1109/ISVLSI.2018.00092.
[40] S. K. Gupta, "Reinforcement Based Learning on Classification Task Could Yield Better Generalization and Adversarial Accuracy." *arXiv*, Dec. 08, 2020. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/2012.04353

[41] E.-C. Chen and C.-R. Lee, “Towards Fast and Robust Adversarial Training for Image Classification,” 2020. Accessed: Jan. 26, 2023. [Online]. Available: https://openaccess.thecvf.com/content/ACCV2020/html/Chen_Towards_Fast_and_Robust_Adversarial_Training_for_Image_Classification_ACCV_2020_paper.html
[42] Q.-Z. Cai, M. Du, C. Liu, and D. Song, “Curriculum Adversarial Training.” *arXiv*, May 12, 2018. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/1805.04807
[43] J. Zhang et al., “Attacks Which Do Not Kill Training Make Adversarial Learning Stronger,” in *Proceedings of the 37th International Conference on Machine Learning*, Nov. 2020, pp. 11278–11287. Accessed: Jan. 26, 2023. [Online]. Available: https://proceedings.mlr.press/v119/zhang20z.html
[44] Y. Wang, X. Ma, J. Bailey, J. Yi, B. Zhou, and Q. Gu, “On the Convergence and Robustness of Adversarial Training.” *arXiv*, Apr. 23, 2022. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/2112.08304
[45] Y. Balaji, T. Goldstein, and J. Hoffman, “Instance adaptive adversarial training: Improved accuracy tradeoffs in neural nets.” *arXiv*, Oct. 17, 2019. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/1910.08051
[46] G. W. Ding, Y. Sharma, K. Y. C. Lui, and R. Huang, “MMA Training: Direct Input Space Margin Maximization through Adversarial Training.” *arXiv*, Mar. 04, 2020. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/1812.02637
[47] M. Cheng, Q. Lei, P.-Y. Chen, I. Dhillon, and C.-J. Hsieh, “CAT: Customized Adversarial Training for Improved Robustness.” *arXiv*, Feb. 17, 2020. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/2002.06789
[48] A. Shafahi et al., “Adversarial Training for Free!” *arXiv*, Nov. 20, 2019. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/1904.12843
[49] H. Zhang, Y. Shi, B. Dong, Y. Han, Y. Li, and X. Kuang, “Free Adversarial Training with Layerwise Heuristic Learning,” in *Image and Graphics*, Cham, 2021, pp. 120–131. Doi: 10.1007/978-3-030-87358-5_10.
[50] H. Zhang, Y. Yu, J. Jiao, E. Xing, L. E. Ghaoui, and M. Jordan, “Theoretically Principled Trade-off between Robustness and Accuracy,” in *Proceedings of the 36th International Conference on Machine Learning*, May 2019, pp. 7472–7482. Accessed: Jan. 26, 2023. [Online]. Available: https://proceedings.mlr.press/v97/zhang19p.html
[51] H. Kannan, A. Kurakin, and I. Goodfellow, “Adversarial Logit Pairing.” *arXiv*, Mar. 16, 2018. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/1803.06373
[52] Y. Wang, D. Zou, J. Yi, J. Bailey, X. Ma, and Q. Gu, “Improving Adversarial Robustness Requires Revisiting Misclassified Examples,” Feb. 2022. Accessed: Jan. 26, 2023. [Online]. Available: https://openreview.net/forum?id=rklOg6EfwS
[53] C. Mao, Z. Zhong, J. Yang, C. Vondrick, and B. Ray, “Metric Learning for Adversarial Robustness.” *arXiv*, Oct. 27, 2019. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/1909.00900
[54] Y. Zhong and W. Deng, “Adversarial Learning With Margin-Based Triplet Embedding Regularization,” 2019, pp. 6549–6558. Accessed: Jan. 26, 2023. [Online].

Available: https://openaccess.thecvf.com/content_ICCV_2019/html/Zhong_Adversarial_Learning_With_Margin-Based_Triplet_Embedding_Regularization_ICCV_2019_paper.html
[55] J. Uesato, J.-B. Alayrac, P.-S. Huang, R. Stanforth, A. Fawzi, and P. Kohli, "Are Labels Required for Improving Adversarial Robustness?" *arXiv*, Dec. 05, 2019. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/1905.13725
[56] Y. Carmon, A. Raghunathan, L. Schmidt, P. Liang, and J. C. Duchi, "Unlabeled Data Improves Adversarial Robustness." *arXiv*, Jan. 13, 2022. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/1905.13736
[57] R. Zhai et al., "Adversarially Robust Generalization Just Requires More Unlabeled Data." *arXiv*, Sep. 25, 2019. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/1906.00555
[58] D. Hendrycks, M. Mazeika, S. Kadavath, and D. Song, "Using Self-Supervised Learning Can Improve Model Robustness and Uncertainty." *arXiv*, Oct. 29, 2019. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/1906.12340
[59] P. Maini, E. Wong, and Z. Kolter, "Adversarial Robustness Against the Union of Multiple Perturbation Models," in *Proceedings of the 37th International Conference on Machine Learning*, Nov. 2020, pp. 6640–6650. Accessed: Jan. 26, 2023. [Online]. Available: https://proceedings.mlr.press/v119/maini20a.html
[60] D. Stutz, M. Hein, and B. Schiele, "Confidence-Calibrated Adversarial Training: Generalizing to Unseen Attacks," in *Proceedings of the 37th International Conference on Machine Learning*, Nov. 2020, pp. 9155–9166. Accessed: Jan. 26, 2023. [Online]. Available: https://proceedings.mlr.press/v119/stutz20a.html
[61] Y. Dong, Z. Deng, T. Pang, J. Zhu, and H. Su, "Adversarial Distributional Training for Robust Deep Learning," in *Advances in Neural Information Processing Systems*, 2020, vol. 33, pp. 8270–8283. Accessed: Jan. 26, 2023. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/5de8a36008b04a6167761fa19b61aa6c-Abstract.html
[62] G. Liu, I. Khalil, and A. Khreishah, "GanDef: A GAN Based Adversarial Training Defense for Neural Network Classifier," in *ICT Systems Security and Privacy Protection*, Cham, 2019, pp. 19–32. Doi: 10.1007/978-3-030-22312-0_2.
[63] S. Rao, D. Stutz, and B. Schiele, "Adversarial Training Against Location-Optimized Adversarial Patches," in *Computer Vision – ECCV 2020 Workshops*, Cham, 2020, pp. 429–448. Doi: 10.1007/978-3-030-68238-5_32.
[64] T. Wu, L. Tong, and Y. Vorobeychik, "Defending Against Physically Realizable Attacks on Image Classification." *arXiv*, Feb. 14, 2020. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/1909.09552
[65] N. Ruiz, S. A. Bargal, and S. Sclaroff, "Disrupting Deepfakes: Adversarial Attacks Against Conditional Image Translation Networks and Facial Manipulation Systems," in *Computer Vision – ECCV 2020 Workshops*, Cham, 2020, pp. 236–251. Doi: 10.1007/978-3-030-66823-5_14.

[66] Y. Jiang, X. Ma, S. M. Erfani, and J. Bailey, "Dual Head Adversarial Training," in *2021 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2021, pp. 1–8. Doi: 10.1109/IJCNN52387.2021.9533363.

[67] L. Ma and L. Liang, "Increasing-Margin Adversarial (IMA) Training to Improve Adversarial Robustness of Neural Networks." *arXiv*, Aug. 21, 2022. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/2005.09147

[68] C. Zhang et al., "Interpreting and Improving Adversarial Robustness of Deep Neural Networks With Neuron Sensitivity," *IEEE Transactions on Image Processing*, vol. 30, pp. 1291–1304, 2021, doi: 10.1109/TIP.2020.3042083.

[69] Q. Bouniot, R. Audigier, and A. Loesch, "Optimal Transport as a Defense Against Adversarial Attacks," in *2020 25th International Conference on Pattern Recognition (ICPR)*, Jan. 2021, pp. 5044–5051. Doi: 10.1109/ICPR48806.2021.9413327.

[70] A. S. Rakin, Z. He, and D. Fan, "Parametric Noise Injection: Trainable Randomness to Improve Deep Neural Network Robustness against Adversarial Attack." *arXiv*, Nov. 22, 2018. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/1811.09310

[71] H. Xu, X. Liu, Y. Li, A. Jain, and J. Tang, "To be Robust or to be Fair: Towards Fairness in Adversarial Training," in *Proceedings of the 38th International Conference on Machine Learning*, Jul. 2021, pp. 11492–11501. Accessed: Jan. 26, 2023. [Online]. Available: https://proceedings.mlr.press/v139/xu21b.html

[72] M. Xu, T. Zhang, Z. Li, M. Liu, and D. Zhang, "Towards evaluating the robustness of deep diagnostic models by adversarial attack," *Medical Image Analysis*, vol. 69, p. 101977, Apr. 2021, doi: 10.1016/j.media.2021.101977.

[73] J. Wang and H. Zhang, "Bilateral Adversarial Training: Towards Fast Training of More Robust Models Against Adversarial Attacks," 2019, pp. 6629–6638. Accessed: Jan. 26, 2023. [Online]. Available: https://openaccess.thecvf.com/content_ICCV_2019/html/Wang_Bilateral_Adversarial_Training_Towards_Fast_Training_of_More_Robust_Models_ICCV_2019_paper.html

[74] D. Stutz, M. Hein, and B. Schiele, "Disentangling Adversarial Robustness and Generalization," 2019, pp. 6976–6987. Accessed: Jan. 26, 2023. [Online]. Available: https://openaccess.thecvf.com/content_CVPR_2019/html/Stutz_Disentangling_Adversarial_Robustness_and_Generalization_CVPR_2019_paper.html

[75] A. Sreevallabh Chivukula, X. Yang, and W. Liu, "Adversarial Deep Learning with Stackelberg Games," in *Neural Information Processing*, Cham, 2019, pp. 3–12. Doi: 10.1007/978-3-030-36808-1_1.

[76] W. Bai, C. Quan, and Z. Luo, "Alleviating adversarial attacks via convolutional autoencoder," in *2017 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence,* Networking and Parallel/Distributed Computing (SNPD), Jun. 2017, pp. 53–58. Doi: 10.1109/SNPD.2017.8022700.

[77] S. Shen, G. Jin, K. Gao, and Y. Zhang, "APE-GAN: Adversarial Perturbation Elimination with GAN." *arXiv*, Sep. 25, 2017. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/1707.05474

[78] F. Yu, L. Wang, X. Fang, and Y. Zhang, "The Defense of Adversarial Example with Conditional Generative Adversarial Networks," *Security and Communication Networks*, vol. 2020, pp. 1–12, Aug. 2020, doi: 10.1155/2020/3932584.
[79] A. ArjomandBigdeli, M. Amirmazlaghani, and M. Khalooei, "Defense against adversarial attacks using DRAGAN," in *2020 6th Iranian Conference on Signal Processing and Intelligent Systems (ICSPIS)*, Dec. 2020, pp. 1–5. Doi: 10.1109/ICSPIS51611.2020.9349536.
[80] G. K. Santhanam and P. Grnarova, "Defending Against Adversarial Attacks by Leveraging an Entire GAN." *arXiv*, May 27, 2018. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/1805.10652
[81] R. Bao, S. Liang, and Q. Wang, "Featurized Bidirectional GAN: Adversarial Defense via Adversarially Learned Semantic Inference." *arXiv*, Sep. 29, 2018. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/1805.07862
[82] Q. Liang, Q. Li, and W. Nie, "LD-GAN: Learning perturbations for adversarial defense based on GAN structure," *Signal Processing: Image Communication*, vol. 103, p. 116659, Apr. 2022, doi: 10.1016/j.image.2022.116659.
[83] D. Wang, W. Jin, Y. Wu, and A. Khan, "Improving Global Adversarial Robustness Generalization With Adversarially Trained GAN." *arXiv*, Mar. 07, 2021. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/2103.04513
[84] T. Pang, K. Xu, C. Du, N. Chen, and J. Zhu, "Improving Adversarial Robustness via Promoting Ensemble Diversity," in *Proceedings of the 36th International Conference on Machine Learning*, May 2019, pp. 4970–4979. Accessed: Jan. 26, 2023. [Online]. Available: https://proceedings.mlr.press/v97/pang19a.html
[85] S. Kariyappa and M. K. Qureshi, "Improving Adversarial Robustness of Ensembles with Diversity Training." *arXiv*, Jan. 28, 2019. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/1901.09981
[86] H. Yang et al., "DVERGE: Diversifying Vulnerabilities for Enhanced Robust Generation of Ensembles," in *Advances in Neural Information Processing Systems*, 2020, vol. 33, pp. 5505–5515. Accessed: Jan. 26, 2023. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/3ad7c2ebb96fcba7cda0cf54a2e802f5-Abstract.html
[87] J. Li, A. Madry, J. Peebles, and L. Schmidt, "On the Limitations of First-Order Approximation in GAN Dynamics," in *Proceedings of the 35th International Conference on Machine Learning*, Jul. 2018, pp. 3005–3013. Accessed: Jan. 26, 2023. [Online]. Available: https://proceedings.mlr.press/v80/li18d.html
[88] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," *arXiv*, Jan. 2017, doi: 10.48550/arXiv.1701.07875.
[89] D. J. Im, H. Ma, C. D. Kim, and G. Taylor, "Generative Adversarial Parallelization," *arXiv*, Dec. 2016, doi: 10.48550/arXiv.1612.04021.
[90] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks." *arXiv*, Aug. 24, 2020. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/1703.10593

[91] J. Ho, A. Jain, and P. Abbeel, “Denoising Diffusion Probabilistic Models,” in *Advances in Neural Information Processing Systems*, 2020, vol. 33, pp. 6840–6851. Accessed: Jan. 26, 2023. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/4c5bcfec8584af0d967f1ab10179ca4b-Abstract.html

[92] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation.” *arXiv*, May 18, 2015. Doi: 10.48550/arXiv.1505.04597.

[93] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition.” *arXiv*, Apr. 10, 2015. Accessed: Jan. 26, 2023. [Online]. Available: http://arxiv.org/abs/1409.1556

[94] J. Shijie, W. Ping, J. Peiyi, and H. Siping, “Research on data augmentation for image classification based on convolution neural networks,” in *2017 Chinese Automation Congress (CAC)*, Oct. 2017, pp. 4165–4170. Doi: 10.1109/CAC.2017.8243510.

[95] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” 2016, pp. 770–778. Accessed: Jan. 26, 2023. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html

[96] I. K. M. Jais, A. R. Ismail, and S. Q. Nisa, “Adam Optimization Algorithm for Wide and Deep Neural Network,” *Knowledge Engineering and Data Science*, vol. 2, no. 1, pp. 41–46, Jun. 2019, doi: 10.17977/um018v2i12019p41-46.

[97] A. Dosovitskiy et al., “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.” *arXiv*, Jun. 03, 2021. Doi: 10.48550/arXiv.2010.11929.

[98] TensorFlow Developers, “TensorFlow.” Zenodo, Feb. 15, 2023. Doi: 10.5281/ZENODO.4724125.

[99] M.-I. Nicolae et al., “Adversarial Robustness Toolbox v1.0.0.” *arXiv*, Nov. 15, 2019. Doi: 10.48550/arXiv.1807.01069.

Appendix A: Training Algorithms

Pseudocode for proposed GAN training with input Equation (3.4) from the Implementation chapter.

Input: Dataset images $x_{i,}$ dataset labels y_i, max number of epochs E, batch size B, total number of sample V, generator network G_1 and $G_{2,}$ generator learning rate R_g, generator scalar ε, classifier network D, classifier leaning rate $R_{d,}$ classifier loss function F_d

for N = 1,2,3…E **do**

for M= 1,2,3…⌜V/B⌝ **do**

Generating adversarial sample $x_i{}'_1$ from $\varepsilon\, G_1\,(x_i)+x_i$

Generating adversarial sample $x_i{}'_2$ from $\varepsilon\, G_2\,(x_i)+x_i$

Getting classifier output O_{i1}, O_{i2}, O_{i3} from $D(x_i{}'_1)$, $D(x_i{}'_2)$, $D(x)$

Calculating loss L_D from $F_d(O_{i1},y_i)$, $F_d(O_{i2},y_i)$, $F_d(O_{i3},y_i)$

Updating D using Adam (R_d, L_D)

Generating adversarial sample $x_i{}'_1{}'$ from $\varepsilon\, G_1\,(x_i)+x_i$

Generating adversarial sample $x_i{}'_2{}'$ from $\varepsilon\, G_2\,(x_i)+x_i$

Getting classifier output $O_{i1}{}'$, $O_{i2}{}'$ from $D(x_i{}'_1{}')$, $D(x_i{}'_2{}')$

Calculating loss L_{g1} and L_{g2} from $F_d\,(O_{i1}{}',y_i)$, $F_d\,(O_{i2}{}',y_i)$

Updating G_1 using Adam (R_g, $-L_{g1}$)

Updating G_2 using Adam (R_g, $-L_{g2}$)

Pseudocode for proposed GAN training with input Equation (3.5) from the Implementation chapter.

Input: Dataset images $x_{i,}$ dataset labels y_i, max number of epochs E, batch size B, total number of sample V, generator network G_1 and $G_{2,}$ generator learning rate R_g, generator scalar ε, classifier network D, classifier leaning rate $R_{d,}$ classifier loss function F_d

for N = 1,2,3…E **do**

for M= 1,2,3…⌜V/B⌝ **do**

Generating latent vector u_i from random gaussian distribution

Generating adversarial sample $x_i{}'_1$ from $\varepsilon\, G_1\,(x_{i,}\, u_i)+x_i$

Generating adversarial sample $x_i'_2$ from $\varepsilon G_2(x_i, u_i)+x_i$

Getting classifier output O_{i1}, O_{i2}, O_{i3} from $D(x_i'_1)$, $D(x_i'_2)$, $D(x)$

Calculating loss L_D from $F_d(O_{i1},y_i)$, $F_d(O_{i2},y_i)$, $F_d(O_{i3},y_i)$

Updating D using Adam (R_d, L_D)

Generating latent vector u_i' from random gaussian distribution

Generating adversarial sample $x_i'_1'$ from $\varepsilon G_1(x_i, u_i')+x_i$

Generating adversarial sample $x_i'_2'$ from $\varepsilon G_2(x_i, u_i')+x_i$

Getting classifier output O_{i1}', O_{i2}' from $D(x_i'_1')$, $D(x_i'_2')$

Calculating loss L_{g1} and L_{g2} from $F_d(O_{i1}',y_i)$, $F_d(O_{i2}',y_i)$

Updating G_1 using Adam (R_g, $-L_{g1}$)

Updating G_2 using Adam (R_g, $-L_{g2}$)

Pseudocode for proposed GAN training with input Equation (3.6) from the Implementation chapter.

Input: Dataset images x_i, dataset labels y_i, max number of epochs E, batch size B, total number of sample V, generator network G_1 and G_2, generator learning rate R_g, generator scalar ε, classifier network D, classifier leaning rate R_d, classifier loss function F_d

for N = 1,2,3…E **do**

for M= 1,2,3…$\lceil V/B \rceil$ **do**

Calculating sign(∇) from D using gradient descent

Generating adversarial sample $x_i'_1$ from $\varepsilon G_1(\text{sign}(\nabla))+x_i$

Generating adversarial sample $x_i'_2$ from $\varepsilon G_2(\text{sign}(\nabla))+x_i$

Getting classifier output O_{i1}, O_{i2}, O_{i3} from $D(x_i'_1)$, $D(x_i'_2)$, $D(x)$

Calculating loss L_D from $F_d(O_{i1},y_i)$, $F_d(O_{i2},y_i)$, $F_d(O_{i3},y_i)$

Updating D using Adam (R_d, L_D)

Calculating sign(∇) from D using gradient descent

Generating adversarial sample $x_i'_1'$ from $\varepsilon G_1(\text{sign}(\nabla))+x_i$

Generating adversarial sample $x_i'_2'$ from $\varepsilon G_2(\text{sign}(\nabla))+x_i$

Getting classifier output O_{i1}', O_{i2}' from $D(x_i'_1')$, $D(x_i'_2')$

Calculating loss L_{g1} and L_{g2} from F_d (O_{i1}', y_i), F_d (O_{i2}', y_i)

Updating G_1 using Adam $(R_g, -L_{g1})$

Updating G_2 using Adam $(R_g, -L_{g2})$

Pseudocode for proposed GAN training with input Equation (3.7) from the Implementation chapter.

Input: Dataset images x_i, dataset labels y_i, max number of epochs E, batch size B, total number of sample V, generator network G_1 and G_2, generator learning rate R_g, generator scalar ε, classifier network D, classifier leaning rate R_d, classifier loss function F_d

for N = 1,2,3…E **do**

for M= 1,2,3…$\lceil V/B \rceil$ **do**

Calculating sign(∇) from D using gradient descent

Generating adversarial sample $x_i'_1$ from ε G_1 $(x_i, \text{sign}(\nabla))+x_i$

Generating adversarial sample $x_i'_2$ from ε G_2 $(x_i, \text{sign}(\nabla))+x_i$

Getting classifier output O_{i1}, O_{i2}, O_{i3} from $D(x_i'_1)$, $D(x_i'_2)$, $D(x)$

Calculating loss L_D from $F_d(O_{i1}, y_i)$, $F_d(O_{i2}, y_i)$, $F_d(O_{i3}, y_i)$

Updating D using Adam (R_d, L_D)

Calculating sign(∇) from D using gradient descent

Generating adversarial sample $x_i'_1'$ from ε G_1 $(x_i, \text{sign}(\nabla))+x_i$

Generating adversarial sample $x_i'_2'$ from ε G_2 $(x_i, \text{sign}(\nabla))+x_i$

Getting classifier output O_{i1}', O_{i2}' from $D(x_i'_1')$, $D(x_i'_2')$

Calculating loss L_{g1} and L_{g2} from F_d (O_{i1}', y_i), F_d (O_{i2}', y_i)

Updating G_1 using Adam $(R_g, -L_{g1})$

Updating G_2 using Adam $(R_g, -L_{g2})$

Appendix B: Example Feature Maps of Proposed Interpretable Model

Figure B.1 and Figure B.2 show the feature maps learned by the interpretable model suggested in Section 6.2. These feature maps represent the important features and patterns learned by the proposed interpretable model before used in the task of classification. The feature maps' colors and the patterns of each class show some consistency and may represent deeper meaning within these image samples.

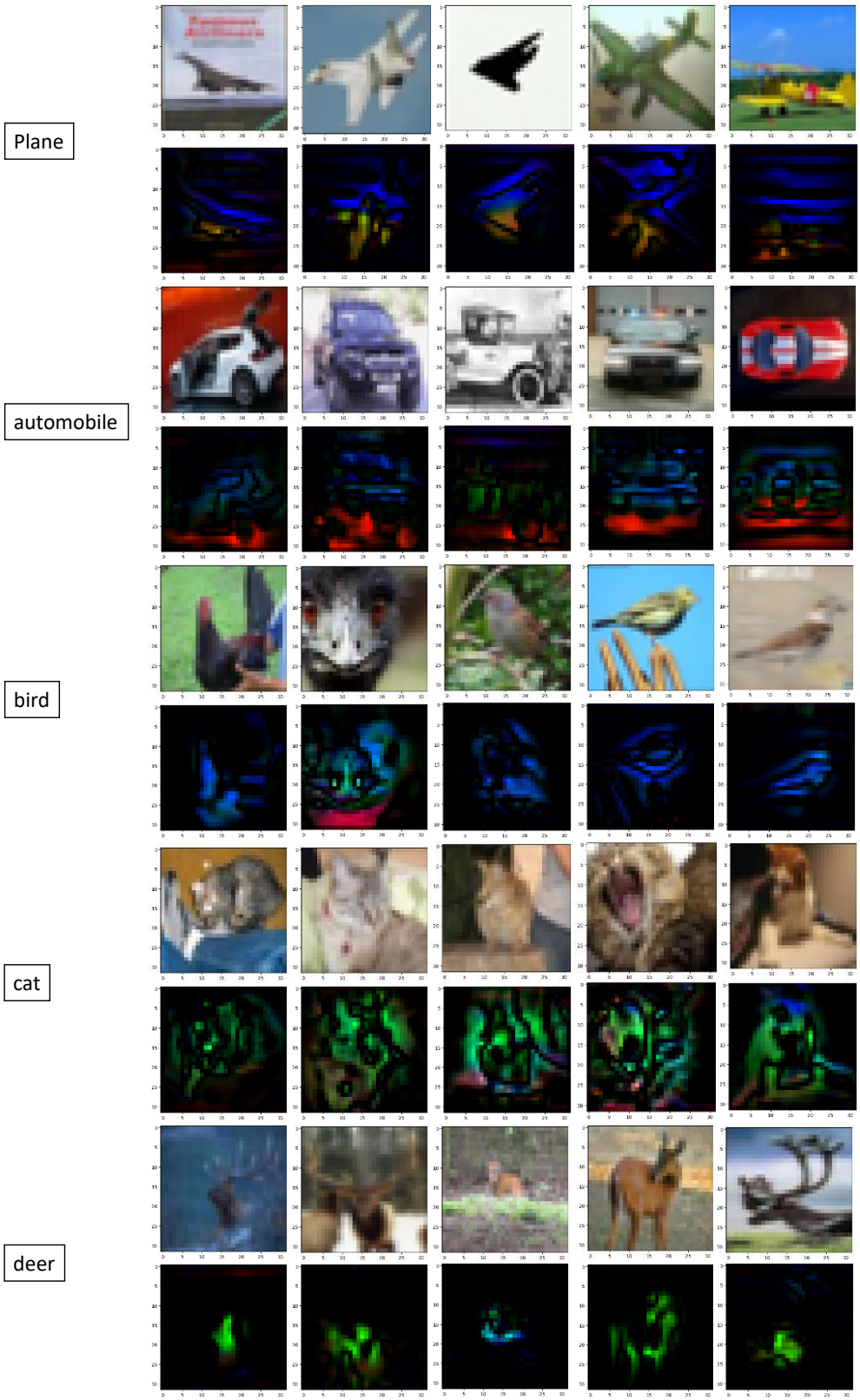

Figure B.1: Example feature maps.

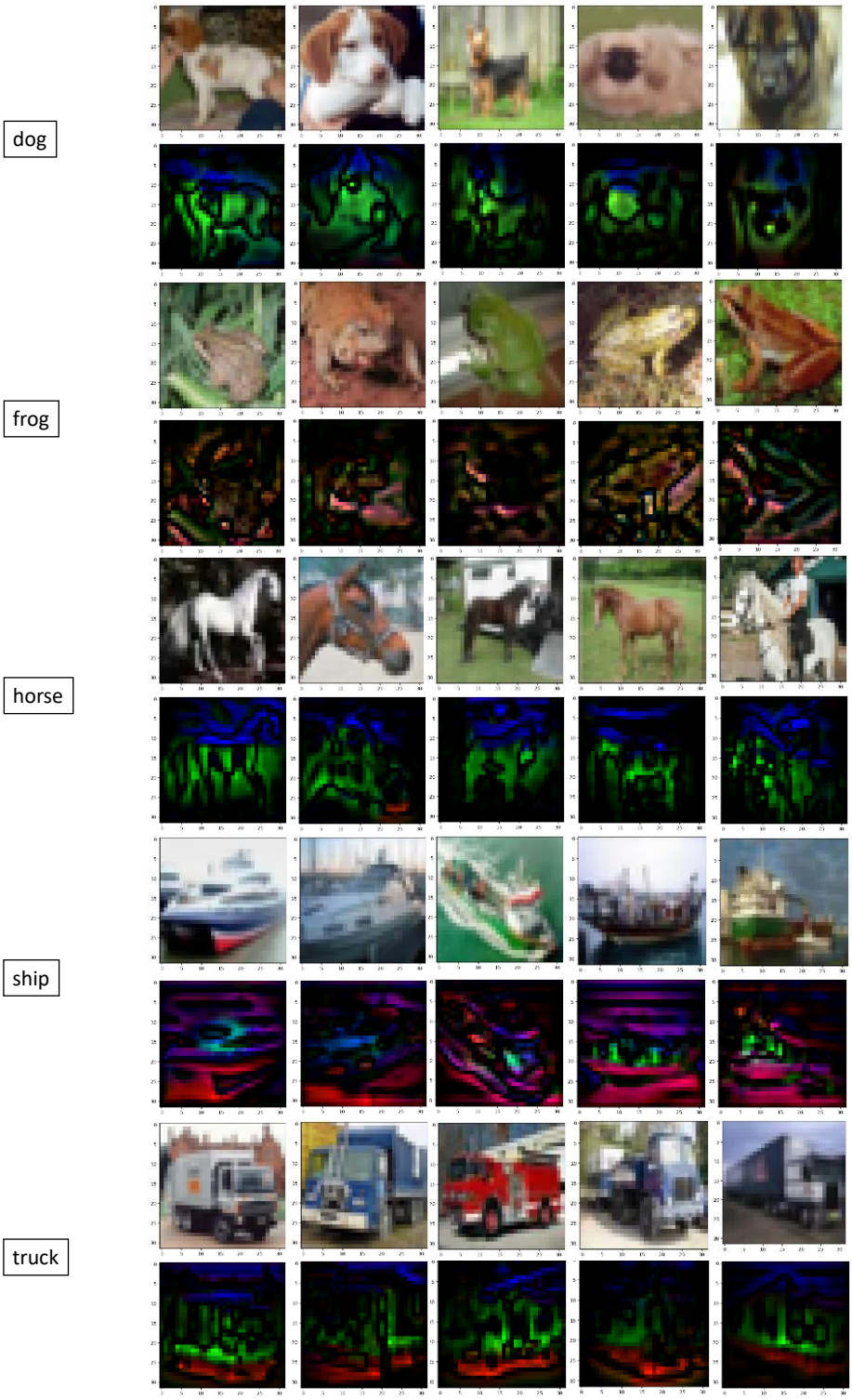

Figure B.2: Example feature maps 2.