LSTAR Framework: Lightweight Framework for Standardizing Tests for Adversarial Robustness

Kenneth Tran

A thesis

submitted in partial fulfillment of the

requirements for the degree of

Master of Science in Computer Science & Software Engineering

University of Washington

2024

Committee:

Brent Lagesse

Muhammad Aurangzeb Ahmad

Dong Si

Program Authorized to Offer Degree:

Computing and Software Systems

University of Washington

Abstract

LSTAR Framework: Lightweight Framework for Standardizing Tests for Adversarial Robustness

Kenneth Tran

Chair of the Supervisory Committee:

Brent Lagesse

Computing and Software Systems

The role of neural networks in various tasks has exploded in recent years, becoming prevalent in many safety-critical applications. However, improving neural network robustness has become a challenge due to the existence of adversarial examples—imperceptible perturbations to the inputs of machine learning models that mislead classifiers into producing incorrect outputs. While there have been numerous advancements in crafting adversarial attacks and defenses, research on the basis of adversarial examples has notably lagged behind, largely due to the computational difficulty of analyzing high-dimensional spaces. This inherent difficulty has led researchers to construct models for understanding adversarial examples divergent from conventional paradigms, with some relying on commonly used frameworks while others utilize their own tailored frameworks to meet their unique needs. Consequently, replicating and building upon research in this field presents a significant challenge.

In this paper, we present a modular, lightweight framework to assist researchers in addressing these challenges by providing a comprehensive approach to evaluating machine learning models through a standardized experimentation platform. We present several potential hypotheses regarding the basis of adversarial examples and utilize our framework to verify them more robustly under complex attacks and datasets through controlled experiments. Our experimental results indicate that geometric causes directly affect the robustness of machine learning models, while statistical factors amplify the effects of adversarial attacks. This framework provides a baseline for further studies to better understand the phenomenon of adversarial examples, allowing researchers to design more robust machine learning models.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

ADVERSARIAL ATTACK: The deliberate manipulation of input data, typically imperceptible to humans, with the goal of causing a machine learning model to make incorrect predictions or classifications.

ADVERSARIAL DEFENSE: Encompasses techniques employed to mitigate the impact of adversarial attacks on machine learning models.

ADVERSARIAL EXAMPLE: A sample of input data crafted to cause a machine learning model to misclassify or produce an incorrect output with high confidence.

CURSE OF DIMENSIONALITY: The phenomenon where the computational and statistical challenges of working in high-dimensional spaces increase dramatically as the number of dimensions increases. See Appendix A.1.

HYPERPARAMETER: A configuration parameter external to a model that influences its performance during training.

MODEL: A structure that is trained on data to make predictions, classify input, or understand patterns in data.

MULTI-LAYER PERCEPTRON: A type of artificial neural network composed of multiple layers of nodes. Each node in one layer is connected to every node in the subsequent layer.

NEURAL NETWORK: A computational model inspired by the structure of the human brain, capable of learning complex patterns from data through interconnected layers of neurons.

PERTURBATION: Refers to a small, intentional modification applied to input data, often imperceptible to humans, aimed at influencing the behavior of a machine learning model, particularly in the context of adversarial attacks and defenses.

ROBUSTNESS: The ability of a model to maintain its performance when subjected to variations in the input data, such as adversarial attacks.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my committee chair, Brent Lagesse, for his incredible support throughout this thesis and throughout my academic career. I would also like to thank my committee members: Muhammad Aurangzeb Ahmad and Dong Si, for their insights and suggestions that helped to refine this work. I am very grateful to Yusuf Pisan and Robert Dimpsey for their support when I was considering applying to graduate programs. My sincere appreciation goes to Phohanh Tran, Abhiram Somisetty, and Ibrahim Nasry for their support in creating the tests for this framework. I would also like to extend my thanks to T-Mobile for their partial funding of this research. Lastly, I would like to thank my family for their support and encouragement throughout this journey. Thank you all for your invaluable contributions to this thesis.

With sincere gratitude,

Kenneth Tran

Chapter 1

INTRODUCTION

Recent years have shown a sharp increase in the use of machine learning and artificial intelligence across a variety of domains. For example, as the role of epidemiologists in healthcare has expanded, the amount of complex health data that can be analyzed by machine learning models has likewise increased. Machine learning has been utilized in that respect to speed up the process of drug discovery and reduce false positives and false negatives in automated disease diagnosis [47]. Machine learning algorithms power recommendation engines behind much of the e-commerce sector [14]. They also can analyze text data from various online sources through natural language processing to extract insights and power virtual assistants [3]. Research into machine learning for autonomous vehicle research has shown that machine learning will have a predominant role in building the perception system supporting autonomous and semi-autonomous vehicles [37]. With machine learning being continually applied to new domains and industries, it is imperative that these machine learning models are secure.

Adversarial examples are inputs specifically crafted with imperceptible perturbations to an original input such that a classifier is misled to making an incorrect output. Since the emergence of [19], researchers have continued a cat and mouse game between adversarial attacks and adversarial defenses meant to increase the robustness of machine learning models to adversarial attacks. Although there is a great amount of work being put into the creation of new attacks and defenses, few studies exist that focus on why adversarial examples have such an effect on machine learning models [9].

There are several difficulties in defining such a study on the effect of adversarial examples. For a single such hypothesis, all dimensions of the input space for a sample must be examined [10]. However, there currently exists no such resource for researchers to visualize the geometry of a high dimensional image space. Known as the "curse of dimensionality," the sheer number of variables that can be collected from a sample can be problematic [30]. In [55], the authors test the impact of different numbers of output features on classification accuracy using ResNet50 [22] on the FaceScrub dataset [34] and similarly find a marked decrease in classification performance after exceeding a certain number of features. As the number of dimensions increases, any given neighborhood of that space becomes more likely to contain no data and thus be sparse, causing difficulty in the collection of data samples that are a sufficient representation of the population, and causing difficulty for machine learning models to detect accurate patterns in the data. Therefore, studies are often constrained by computational resource limitations. There exist several proposed strategies to conduct such studies to elucidate the reasons for the existence of adversarial examples [27]. With these studies as a basis, in this paper, we design a framework to allow researchers to design their own controlled experiments to verify the basis of adversarial examples.

This paper does not contain any state-of-the-art adversarial attacks or defenses. This framework and the analysis of the experiments instead provide a useful starting point for the future analysis of the basis of adversarial examples and more adversarially robust machine learning models.

1.1 Problem definition

As the prevalence of machine learning in safety-critical applications increases, it becomes increasingly more important that these systems are properly secured against adversarial attacks. Machine learning has been successfully utilized in a variety of applications [37, 3, 47, 14]. However, research has shown that models remain vulnerable to adversarial examples. For example, a malicious user can create a physical adversarial example by placing stickers on a stop sign to cause an autonomous vehicle to misclassify a stop sign as a speed limit sign [15]. Such instances underscore the importance of fortifying machine learning systems against adversarial manipulation. While efforts to develop robust defenses against adversarial attacks are crucial, understanding the fundamental mechanisms behind why machine learning models are vulnerable to adversarial examples is equally important for the development of effective solutions.

Adversarial attacks have often been considered an auxiliary task to the standard machine learning pipeline, often requiring the integration of additional libraries such as the Adversarial Robustness Toolbox [35]. This approach introduces dependencies and possible compatibility issues across different versions of software packages. Recent papers have instead focused on creating their own unique framework, causing difficulty for others to replicate and modify their work [31]. By designing a modular framework with adversarial attacks and defenses built into the modules, researchers may tailor the architecture to fit their specific use cases.

1.2 Goals

In this paper, we develop a modular framework based on strategies used to conduct studies into the basis of adversarial examples and answer the following questions:

- Do adversarial perturbations get magnified by linear coefficients in the model? (Section 3.2.3, 4.1)

- Do classifiers tend to be less robust to adversarial examples because output probabilities must add up to 1? (Section 3.2.4, 4.2)

- Do adversarial examples occur more easily in classifiers trained on a greater number of target categories due to a greater number of classes creating more boundaries between classes that can be shifted? (Section 3.2.5, 4.3)

1.3 Contributions

In this work, we expand the scope of existing studies on adversarial examples by integrating the strategies and designs developed to conduct empirical studies into the basis of adversarial examples to create a lightweight modular framework tailored to investigate adversarial examples. This framework is then employed to more rigorously examine a series of hypotheses concerning the basis of adversarial examples through the use of a broader range of datasets and adversarial attack methods. To illustrate the efficacy of our approach, we establish a performance benchmark against TensorFlow, a widely used machine learning framework. Furthermore, we ensure the practicality and user-friendliness of our framework by conducting a comprehensive usability test.

1.4 Outline

Chapter 2 contains a brief introduction to machine learning, neural networks, and hyperparameters. Then, summaries of related work and current research into adversarial attacks and defenses are presented. Chapter 3 details the methodology for framework development and hypothesis verification. Chapter 4 presents the results of the conducted experiments. This is followed by chapter 5 discussing the results, a comparison to TensorFlow, and limitations of the framework. We conclude in chapter 6 with practical implications and future directions of this work.

Chapter 2
BACKGROUND

In this chapter, we briefly introduce machine learning, the concept of neural networks, single and multi-layer perceptrons, hyperparameters, adversarial attacks and defenses, adversarial robustness, and related work.

2.1 Machine learning

Modern machine learning can be categorized into four types:

- Supervised learning
- Semi-supervised learning
- Unsupervised learning
- Reinforcement learning

Supervised learning: A method in which given input data is used to reach given output data. There are two types of supervised learning: classification and regression.

- Classification: Distribution of data into disparate categories defined on the data set according to their specific features. A mapping function f takes input variables X and maps them to a discrete output label y.

- Regression: Prediction of other features of the data based on currently available features. A mapping function f takes input variables X and maps them to a continuous output variable y.

Semi-supervised learning: A method used when obtaining sufficient labeled data is prohibitively difficult or expensive, but unlabeled data is easy to acquire. In such scenarios, neither supervised nor unsupervised learning methods will provide adequate solutions.

Unsupervised learning: In contrast to supervised learning, the output data is not given to the model. Unsupervised learning models are given unlabeled data and discover patterns and insights within the data without any explicit guidance. These are often useful for categorizing data. One technique in unsupervised learning is dimensionality reduction, where the goal is to reduce the number of features in a dataset while preserving its essential characteristics. This involves techniques such as principal component analysis or t-distributed stochastic neighbor embedding [42].

Reinforcement learning: A method that mimics the trial-and-error process to achieve a goal. They use a reward and punishment paradigm to process data. The model builds a set of if-then rules as it trains, helping it decide what action to take next for optimal cumulative "reward." The exploration process is continuously validated and improved to increase the probability of reaching the end goal of training the model [45].

2.2 Neural networks

Research into neural networks has grown explosively over the last several years. The strategy for this research has been to develop mathematical models defining brain-like systems, then utilize these models to understand how computational problems can be solved through such methods. This field has garnered the attention from a multitude of fields, including neuroscientists, physicists, computer engineers, computer scientists, psychologists, philosophers, and many others.

The inspiration for neural networks comes originally from studies on the mechanisms of information processing in biological nervous systems [40]. In more practical applications, a neural network can be regarded as a nonlinear mathematical function that takes a set of input variables and outputs a set of output variables. This transformation is governed by weights in the network that are determined through another process called model training. Alongside high processing speed, neural networks have the capability of learning general solutions to problems in a domain given a set of specific example inputs.

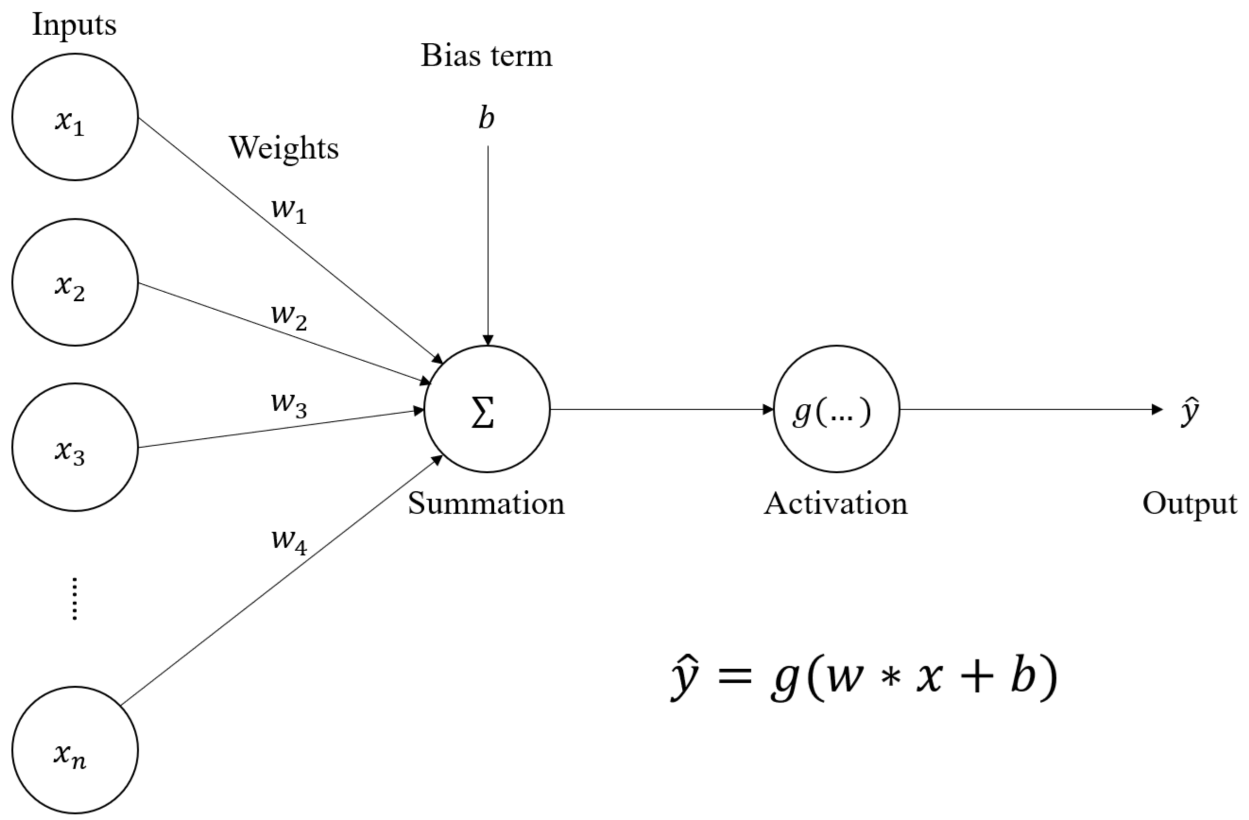


Figure 2.1: A depiction of an artificial neuron, the most basic unit of neural networks.

2.3 Multi-layer perceptron and back-propagation

Multi-layer perceptrons are one of the most widely used kinds of neural networks. They are formed by units that take an input, add a constant term, summed, then passed through a nonlinear activation function. These are most often connected in a feed-forward manner. For some kinds of applications using non-feed-forward neural networks, interconnections instead form loops. Training of such a network involves a backward propagation through the network being trained.

The classic multi-layer perceptron with back-propagation consists of five parts:

- A linear function that aggregates input values
- A sigmoid function, also called the activation function
- A threshold function for classification or an identity function for regression
- A loss function that computes the overall error of the network, also called the cost function
- A learning procedure to adjust the weights of the network, also called back-propagation

The non-linearity introduced in the activation function can be any differentiable function. The kind of non-linearity that is most used takes the form of a sigmoid function, with two horizontal asymptotes and is monotonically increasing. Some of the most common mathematical expressions for this function are:

$$S(s) = \frac{1}{1+e^{-s}} = \frac{1+\tanh(s/2)}{2} \tag{2.1}$$

$$S(s) = \tanh(s) \tag{2.2}$$

$$S(s) = \arctan(s) \tag{2.3}$$

Several different methods exist for minimizing an error function. Among those, one of the simplest is the gradient method. Iteratively take steps in weight space proportional to the negative gradient of the function to be minimized. In other words, iteratively update weights according to:

$$w_{n+1} = w_n - \eta \nabla E, \tag{2.4}$$

where w_n is the old weight, η is the learning rate, ∇E is the gradient of the error vector, and w_{n+1} is the new weight.

Back-propagation then takes the error between the predicted output and the actual target output via a loss function and propagates it backwards through the network to update the weights of connections between neurons. The gradient of the loss function with respect to each weight in the network is calculated through the chain rule, indicating the direction and magnitude of the change in weights necessary to minimize the error.

2.4 Hyperparameters

Machine learning algorithms for regression and classification involve several parameters that must be set before running them, known as hyperparameters. These tuning parameters often must be carefully optimized to achieve optimal performance [6]. To select an appropriate hyperparameter configuration for a specific task and model, users can resort to default values specified by software packages or manually configure them based on literature recommendations, experience, or trial and error. Alternatively, there exist hyperparameter tuning strategies that are optimization processes that minimize the generalization error of the machine learning model over the hyperparameter search space of candidate configurations, often utilizing grid or random search [1].

2.5 Adversarial attacks

Adversarial attacks are inputs intentionally designed to mislead machine learning models [7]. Given a model F and an input image $x \in X$ with the ground truth label $y \in Y$, an adversarial example x_{adv} satisfies

$$f(x_{adv}) \neq y \quad s.t. \quad ||x - x_{adv}|| \leq \epsilon, \tag{2.5}$$

where $||x - x_{adv}||$ is a distance metric measured by the $\mathcal{L}_p$-norm $p \in (1, 2, \infty)$ and ϵ is the limit of the magnitude of the adversarial perturbation.

Adversarial attacks can be divided into three classes: white-box attacks in which adversaries have complete knowledge and access to the target model, black-box attacks where adversaries have limited knowledge and access to the target model, and grey-box attacks where the attacker possesses partial knowledge about the target model, such as certain aspects of the model architecture, its training data, or its output responses. By adding well-designed perturbations to clean inputs that are imperceptible to humans, attackers can manipulate the predictions of a model. Such instances are called adversarial samples. Approaches to crafting adversarial samples have been proposed in a multitude of tasks, most often on image data [19], but also seen in various other data types. A non-exhaustive list of examples include: text [4], graph data [50], object detection [53], video data [49], and fraud detection [52].

2.5.1 Fast gradient sign method

The Fast Gradient Sign Method (FGSM) is a simple method for creating adversarial example [19]. As FGSM is a white-box attack, the attacker must have knowledge of the pre-trained model. Any given adversarial sample generated will be of the form:

$$x_{adv} = x + \eta, \tag{2.6}$$

where x_{adv} is the adversarial sample, x is the original input, and η is the adversarial perturbation added to the original input. This adversarial perturbation is defined as:

$$\eta = \epsilon \times sign(\nabla_x J(\theta, x, y)), \tag{2.7}$$

where y is the target vector associated with the input vector x, J is the loss function, θ is the set of parameters, and ϵ is the magnitude of the perturbation step in the direction of the gradient [28].

While FGSM is a very fast method for generating adversarial examples, it was not intended to be an attack for evaluating the robustness of a neural network. There are many defenses that achieve close to perfect success in defending against this type of attack. However, evaluation against FGSM can be a good component of an overall robustness evaluation when used in conjunction with other attacks.

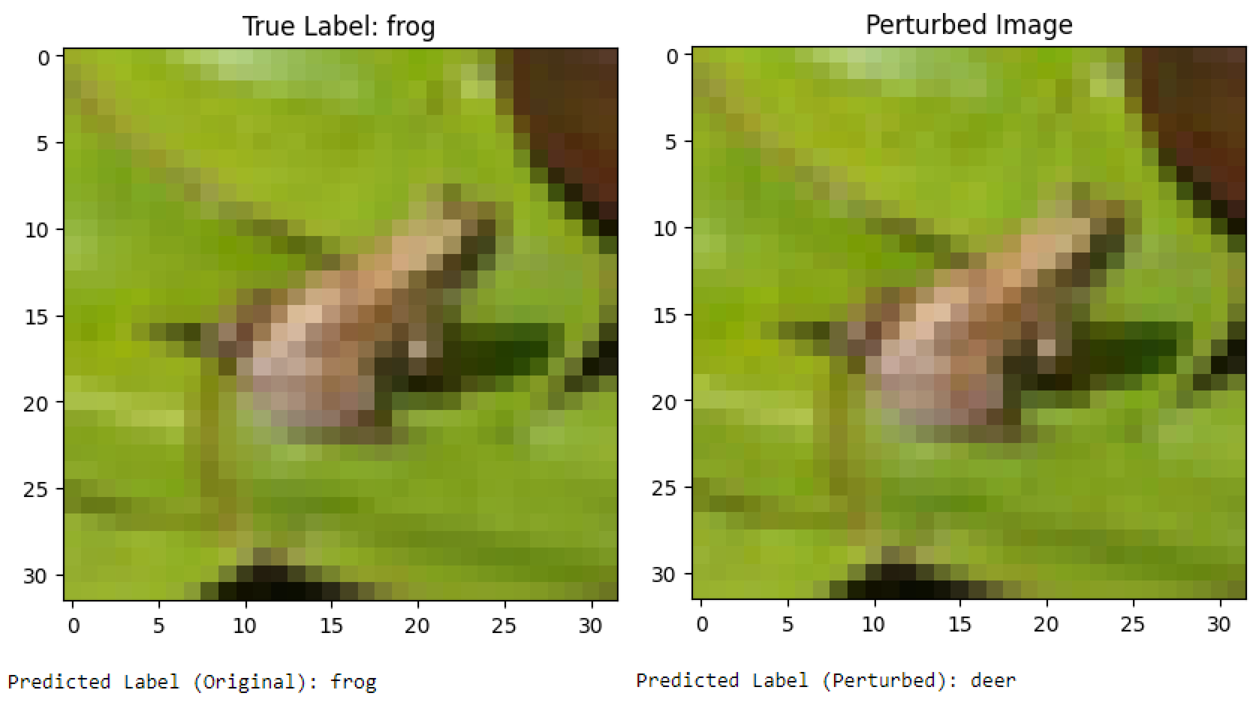

True Label: frog
Perturbed Image
0
5
10
15
20
25
30
Predicted Label (Original): frog
Predicted Label (Perturbed): deer

Figure 2.2: FGSM applied to an image in CIFAR-10 [38] using the framework proposed in this paper. The original image is correctly classified as a frog. The perturbed image looks the same to a human viewer and is incorrectly classified as a deer.

2.5.2 Projected gradient descent

The projected gradient descent (PGD) attack does not attempt to find the smallest adversarial perturbation, but instead aims for robust optimization by maximizing the loss function. The goal is to generate perturbations that cause a target model to misclassify inputs. It iteratively computes gradients of a chosen loss function with respect to the input data and updates the perturbation in the direction that maximizes this loss while constraining the perturbation within a specified epsilon bound. The foundation of perturbation applied by this attack is updated in each iteration and given by:

$$\delta \leftarrow \prod_{S}[\delta + \tau \text{sign}(\nabla_x \mathcal{L}(x + \delta, y))], \tag{2.8}$$

where δ is a randomly initialized perturbation $\delta \in S$, x is the input, y is the corresponding label for each put, and τ is a fixed step size [29]. It is inferred that strong fluctuations in the gradient help the attack escape sub-optimal local minima and maxima.

2.5.3 Momentum iterative method

The momentum iterative method (MIM) is a technique for iteratively accelerating gradient descent algorithms by accumulating a velocity vector in the gradient direction of the loss function. By memorizing previous gradients, it allows the optimizer to not get stuck on poor local minima or maxima, allowing for effective locating of global minima and maxima. A non-targeted adversarial example x_{adv} from an original input image x, satisfying the $\mathcal{L}_\infty$-norm bound, solves the constrained optimization problem given by:

$$\text{argmax } J(x_{adv}, y), \quad ||x - x_{adv}|| \leq \epsilon, \tag{2.9}$$

where J is the loss function, y is the ground truth label, and ϵ is the size of the adversarial perturbation. This can also be easily generalized to other attack settings. For example, by replacing the current gradient with the accumulated gradient of all previous steps, any iterative method can be extended to its momentum variant [13].

2.6 Adversarial defenses

There are a multitude of proposed defense strategies against adversarial attacks. The basic idea for common strategies are introduced in this subsection.

2.6.1 Input denoising

Adversarial attacks add a type of perturbation in the form of human imperceptible noise to inputs. One natural solution to this would be to filter it out or add additional random noises to offset such perturbations.

2.6.2 Model robustification

Making models more robust to prepare against potential adversarial threats is also widely applied. This is often achieved in the following ways: adversarial training [43], hyperparameter optimization [6], model distillation [18], and layer discretization [36].

2.6.3 Adversarial detection

Given an instance, the goal of adversarial detection is to identify whether the instance is adversarially perturbed. The idea is to build another binary classifier such that it successfully predicts whether an input has been adversarially perturbed. While input denoising and model robustification are proactive responses to adversarial perturbation, adversarial detection is a more reactive response to such inputs.

2.7 Robustness

Robustness metrics are a measure of the vulnerability of a classifier with respect to adversarial attacks, often used to assess the effectiveness of adversarial defenses. Existing metrics typically quantify the amount of perturbation required to cause a model to make a misclassification. In a more general sense, it measures the sensitivity of model outputs to changes in their inputs.

One issue behind the slow progress into the field of adversarial attacks and defenses is the shortage of tools to reliably evaluate the robustness of machine learning models. While research into novel attacks and defenses adds to the collective knowledge of the field, research in this field has essentially remained a cat and mouse game with researchers creating new attacks and defenses to prove that the current state-of-the-art is ineffective. For example, in [2], their machine learning models appeared robust because standard adversarial attacks could not find the true minimum adversarial perturbations. Attacks such as projected gradient descent or Carlini & Wagner may have many reasons for failing to converge on a true minimum adversarial perturbation, such as setting sub-optimal hyperparameters, having too few optimization steps, or due to gradient masking.

2.8 Related work

Investigation into how machine learning models behave within adversarial environments continues to be a popular topic of research. Several studies have delved into poisoning attacks, those in which specially crafted attack points are injected into the training data [35], evasion attacks, in which an adversary creates purposefully perturbed inputs that are misclassified by a classifier [11, 50, 19], and privacy-aware machine learning, in which researchers propose a multitude of strategies to preserve the privacy of data while performing model training [46].

Adversarial defenses have also been a large problem within the adversarial machine learning community. Researchers have made many attempts for hardening machine learning systems through various means, including a method to introduce randomness in the selection of classification boundaries [48], reducing the feature sets on classifiers under adversarial attack [54], and adversarial training, where a model is intentionally trained on both benign examples and their corresponding adversarial counterparts to enhance its overall robustness and generalization capabilities [19]. This remains a hot topic in the research community.

Several studies have created their own frameworks to further elucidate the basis of adversarial examples. In [21], the authors present a framework to systematically assess the robustness of deep learning models. They focus on comprehensive evaluation metrics to fully evaluate model robustness and provide insight into building more robust machine learning models, primarily on image classification tasks with respect to $\mathcal{L}_p$-norm bounded adversaries. They conduct the effectiveness of their evaluation framework by conducting experiments on a multitude of data sets, including CIFAR-10 [38] and SVHN [33], through which they conclude that some defenses are weak to more rigorous metrics though they show promise on simple metrics, and alongside $\mathcal{L}_\infty$-norm adversarial examples, they recommend more diverse attacks be used to conduct comprehensive evaluations.

Another such study involves the design and implementation a generic adversarial testing framework for recurrent neural networks that focuses on diversity of adversary orientations and test coverage. RNN-Test introduces the idea of evaluating the robustness of recurrent neural networks by generating adversarial inputs that expose weaknesses in model behavior by systematically perturbing input sequences to better understand model vulnerabilities that may not be apparent under normal testing conditions [20].

Due to the difficulty in uncovering vulnerabilities in systems that are deployed in complex environments, such as autonomous vehicles, the authors of [39] introduce ANTI-CARLA, a framework for assessing the adversarial robustness of autonomous vehicles. This framework employs diverse adversarial attack strategies to better identify the reasoning behind the vulnerability of these models. Similarly, another study approached this problem by proposing a framework that identifies and prioritizes tests that are statistically more likely to expose vulnerabilities in the model behavior of deep neural networks [17].

2.9 Applications

There are many real-world applications of adversarial attacks and defenses across a variety of fields. We discuss several such applications below.

2.9.1 Spam filtering

SpamBayes [32] is a content-based statistical spam filter that classifies emails using token counts. It computes a score for each token in an email based on its occurrence in spam and non-spam emails. This model was trained on a smoothed estimate of the posterior probability that an email containing the token is spam. This score is compared against two thresholds to select the label spam, not-spam, or unsure. After performing several such adversarial attacks on these models, the authors found that their adversarial attacks were highly effective even when contaminating only a small percentage of the training data.

2.9.2 Anomalous network traffic detection

In anomalous network traffic detection, researchers have shown that the injection of adversarial samples during training can poison a detector such that it cannot detect a subsequent denial of service attack. They circumvent detection by slowly poisoning the training data for the model over several weeks. Over several weeks, researchers slowly increased the magnitude of the adversarial perturbation so that the detector acclimates to the changes and fails to identify the large amount of poisoning. This caused the detector to learn a distorted set of principal components that were unable to effectively differentiate between denial-of-service attacks [25].

2.9.3 Printing adversarially perturbed images

Another researcher studied whether adversarial examples generated under FGSM (see Section 2.5.1) can be printed and remain adversarial in the physical world. This was evaluated by capturing pictures of printed adversarial examples using a cellphone camera and classified using the Inception-v3 model [24]. From their experiments, they found FGSM to be robust in the physical world but only seem to work in limited settings, and have yet to be evaluated on more diverse and extreme sets of 2D and 3D transformations.

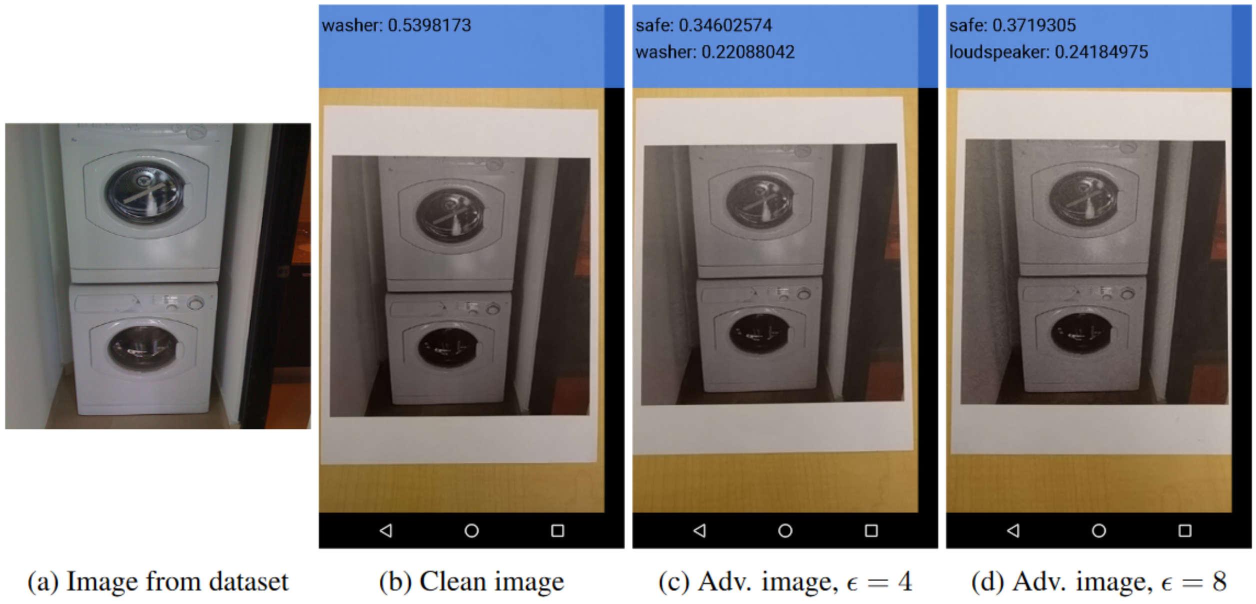


Figure 2.3: Demonstration of a black box attack on a phone app for image classification using physical adversarial examples. Taken from [24].

2.9.4 Adversarial sunglasses

It has also been demonstrated how a printable adversarial eyeglass can fool facial recognition systems. When tested on various facial recognition models, including the state-of-the-art facial recognition model at the time, they showed that the adversarially perturbed, printed eyeglasses fooled various models, including the state-of-the-art [44].

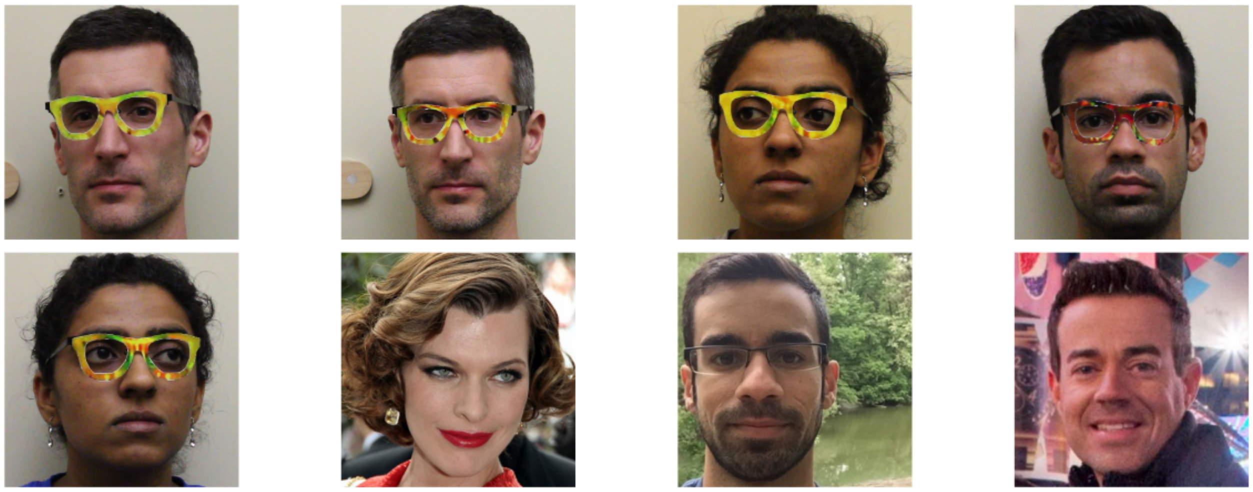


Figure 2.4: Example of the printed adversarial eyeglass attack. The top row shows input images and the bottom row shows the misclassified labels. Taken from [44].

2.9.5 Road signs

Another proposed attack is one called the Robust Physical Perturbation (RP_2), showing that adversarial samples could be crafted to be robust to changes in angle, distance, and lighting. This was specifically testing on road signs [15]. The authors collected pictures of the same road signs under various conditions and used them as part of the adversarial example generation process. RP_2 was used to generate two attacks: poster-printing and sticker attacks. Poster-printing is a scenario where an attacker prints an adversarially perturbed image of a road sign at full size and overlays it on top of the real road sign. Sticker attacks are scenarios where an attacker prints stickers containing adversarial perturbations and places them on a real road sign. They successfully show how the generated road signs can be physically realizable and fool a model in a controlled environment.

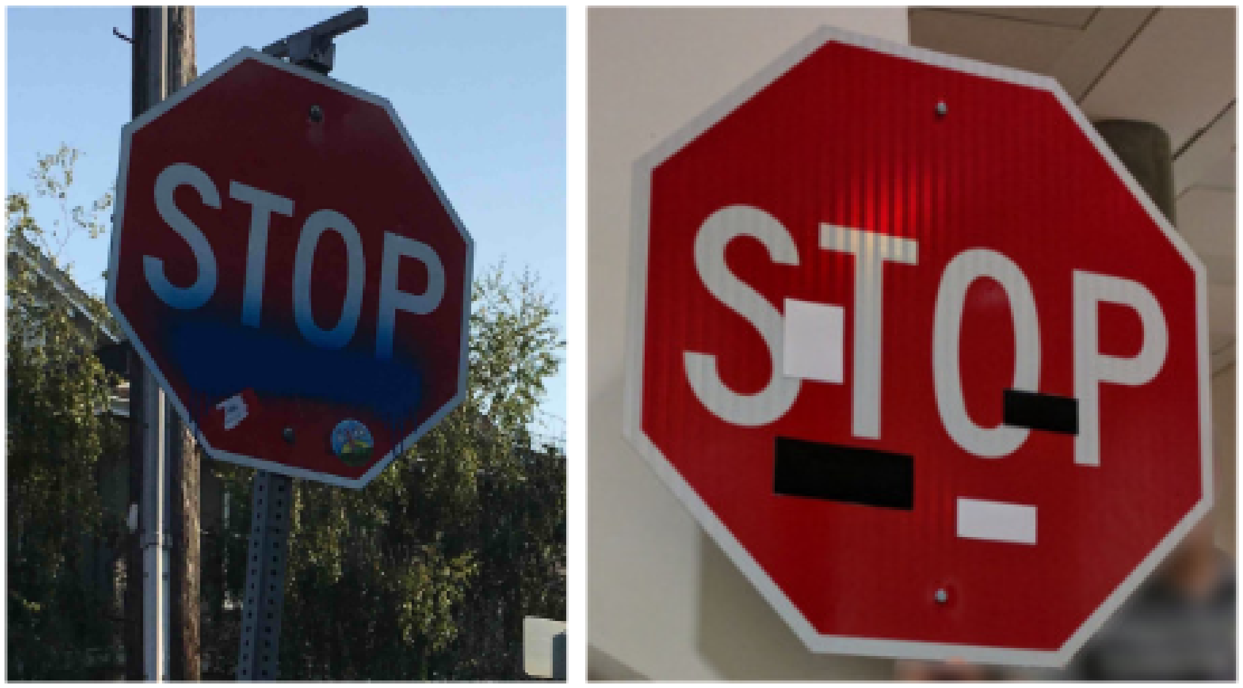


Figure 2.5: An example of a sticker attack that caused the stop sign to be misclassified as a speed limit 45 sign. Taken from [15].

2.9.6 Adversarial patch

Many attacks on image classifiers tend to focus on modifying an image such that adversarial images look like the original instance. It has been demonstrated that they are able to generate an adversarial patch that is input-agnostic that causes a classifier to misclassify an input whenever the patch is present in the image [8]. This was shown to be robust in fooling various ImageNet classifiers in both white-box and black-box settings. The following figure illustrates how the adversarial patch fools the target model into thinking that the image contains a toaster as opposed to a banana.

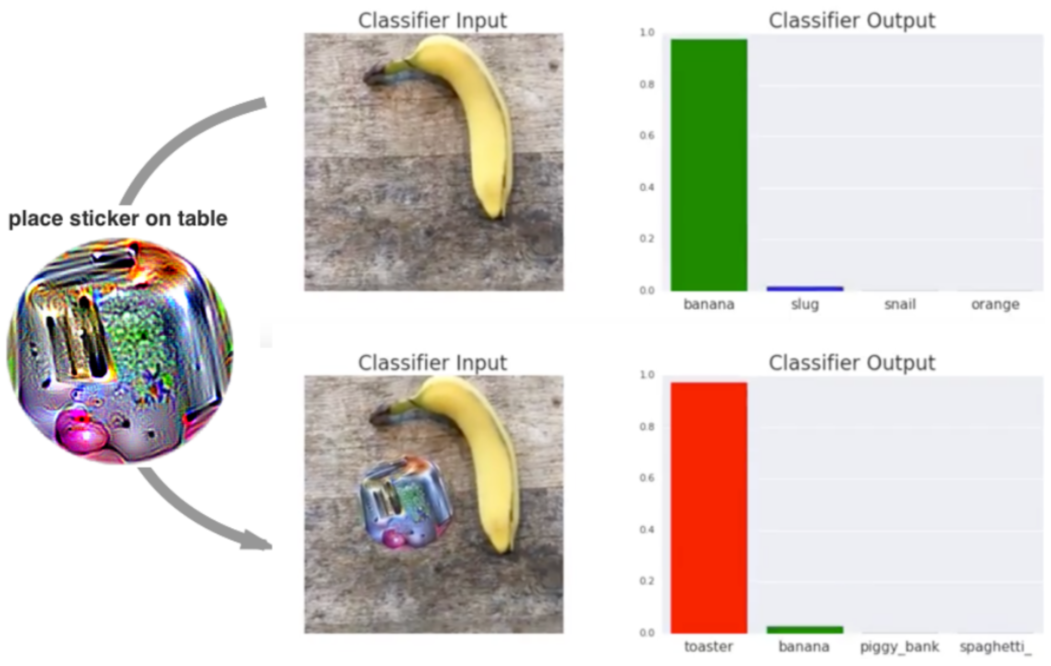


Figure 2.6: A real-world attack on VGG16 using a physical patch with adversarial perturbations. The bottom photograph is classified as a toaster with 97% confidence. Taken from [8].

Chapter 3
METHODS

In this section, we discuss the hypotheses to be tested, proposed method for evaluation, attacks, metrics, software versions and hardware used, and usability test. Additionally, reasoning for the selection of each element will be analyzed.

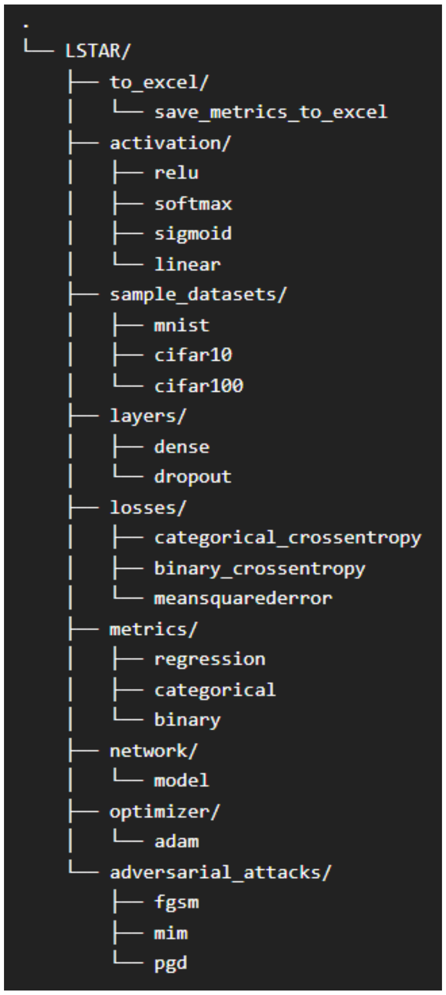
3.1 Proposed framework

The primary objective of this paper is to develop a modular testing framework to assist with standardization of the tools researchers use to analyze adversarial examples. This arises from the multifaceted nature of adversarial attacks and defenses that require comprehensive evaluation metrics to fully assess their efficacy and robustness. The modularity of such a framework allows researchers to easily integrate and build upon novel attack and defense strategies as the field continues to evolve.

Prior research on the foundations of adversarial examples has encountered some challenges in replication and modification. Moreover, the predominant focus on FGSM and MNIST in existing studies limits the exploration of alternative methodologies. To address these limitations, we propose the development of a structured modular machine learning framework tailored to facilitate the testing of hypotheses and extend our study to more complex attacks and datasets.

This framework is subdivided into eight sections: network, dataset loaders, layers, activation functions, loss functions, metrics, optimizers, and adversarial attacks. In addition, we perform a usability test to evaluate the accessibility and user-friendliness of the framework on its target audience.

The file structure of the framework is visualized in Figure 3.1 and interactions between framework components are visualized in Figure 3.2:



```
.
└── LSTAR/
    ├── to_excel/
    │   └── save_metrics_to_excel
    ├── activation/
    │   ├── relu
    │   ├── softmax
    │   ├── sigmoid
    │   └── linear
    ├── sample_datasets/
    │   ├── mnist
    │   ├── cifar10
    │   └── cifar100
    ├── layers/
    │   ├── dense
    │   └── dropout
    ├── losses/
    │   ├── categorical_crossentropy
    │   ├── binary_crossentropy
    │   └── meansquarederror
    ├── metrics/
    │   ├── regression
    │   ├── categorical
    │   └── binary
    ├── network/
    │   └── model
    ├── optimizer/
    │   └── adam
    └── adversarial_attacks/
        ├── fgsm
        ├── mim
        └── pgd
```

Figure 3.1: Directory tree of the LSTAR framework.

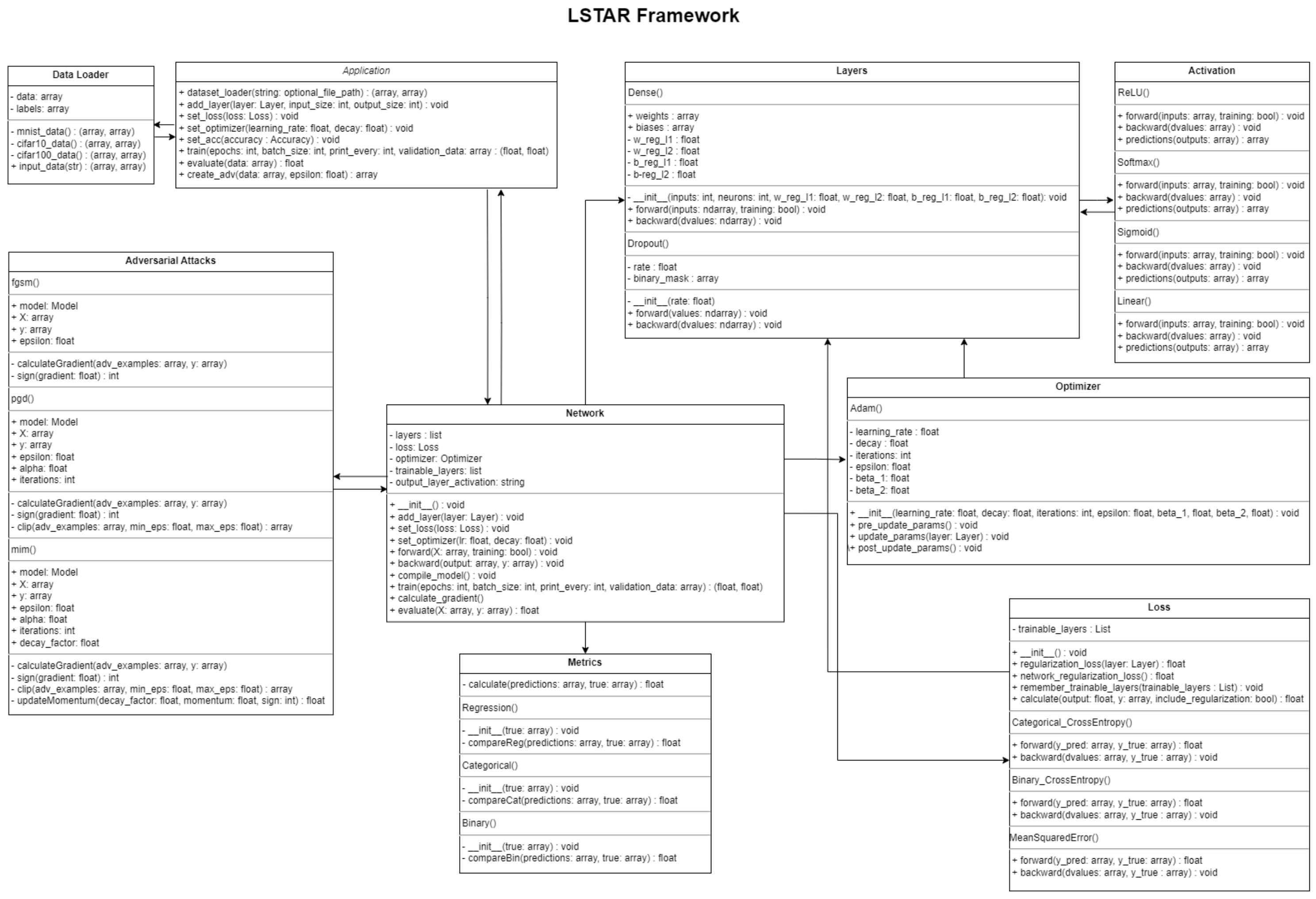

LSTAR Framework
Data Loader
- data: array
- labels: array
- mnist_data() : (array, array)
- cifar10_data() : (array, array)
- cifar100_data() : (array, array)
+ input_data(str) : (array, array)
Application
+ dataset_loader(string: optional_file_path) : (array, array)
+ add_layer(layer: Layer, input_size: int, output_size: int) : void
+ set_loss(loss: Loss) : void
+ set_optimizer(learning_rate: float, decay: float) : void
+ set_acc(accuracy : Accuracy) : void
+ train(epochs: int, batch_size: int, print_every: int, validation_data: array : (float, float)
+ evaluate(data: array) : float
+ create_adv(data: array, epsilon: float) : array
Layers
Dense()
+ weights : array
+ biases : array
- w_reg_l1 : float
- w_reg_l2 : float
- b_reg_l1 : float
- b-reg_l2 : float
- __init__(inputs: int, neurons: int, w_reg_l1: float, w_reg_l2: float, b_reg_l1: float, b_reg_l2: float): void
+ forward(inputs: ndarray, training: bool) : void
+ backward(dvalues: ndarray) : void
Dropout()
- rate : float
- binary_mask : array
- __init__(rate: float)
+ forward(values: ndarray) : void
+ backward(dvalues: ndarray) : void
Activation
ReLU()
+ forward(inputs: array, training: bool) : void
+ backward(dvalues: array) : void
+ predictions(outputs: array) : array
Softmax()
+ forward(inputs: array, training: bool) : void
+ backward(dvalues: array) : void
+ predictions(outputs: array) : array
Sigmoid()
+ forward(inputs: array, training: bool) : void
+ backward(dvalues: array) : void
+ predictions(outputs: array) : array
Linear()
+ forward(inputs: array, training: bool) : void
+ backward(dvalues: array) : void
+ predictions(outputs: array) : array
Adversarial Attacks
fgsm()
+ model: Model
+ X: array
+ y: array
+ epsilon: float
- calculateGradient(adv_examples: array, y: array)
- sign(gradient: float) : int
pgd()
+ model: Model
+ X: array
+ y: array
+ epsilon: float
+ alpha: float
+ iterations: int
- calculateGradient(adv_examples: array, y: array)
- sign(gradient: float) : int
- clip(adv_examples: array, min_eps: float, max_eps: float) : array
mim()
+ model: Model
+ X: array
+ y: array
+ epsilon: float
+ alpha: float
+ iterations: int
+ decay_factor: float
- calculateGradient(adv_examples: array, y: array)
- sign(gradient: float) : int
- clip(adv_examples: array, min_eps: float, max_eps: float) : array
- updateMomentum(decay_factor: float, momentum: float, sign: int) : float
Network
- layers : list
- loss: Loss
- optimizer: Optimizer
- trainable_layers: list
- output_layer_activation: string
+ __init__() : void
+ add_layer(layer: Layer) : void
+ set_loss(loss: Loss) : void
+ set_optimizer(lr: float, decay: float) : void
+ forward(X: array, training: bool) : void
+ backward(output: array, y: array) : void
+ compile_model() : void
+ train(epochs: int, batch_size: int, print_every: int, validation_data: array) : (float, float)
+ calculate_gradient()
+ evaluate(X: array, y: array) : float
Optimizer
Adam()
- learning_rate : float
- decay : float
- iterations: int
- epsilon: float
- beta_1: float
- beta_2: float
+ __init__(learning_rate: float, decay: float, iterations: int, epsilon: float, beta_1, float, beta_2, float) : void
+ pre_update_params() : void
+ update_params(layer: Layer) : void
+ post_update_params() : void
Metrics
- calculate(predictions: array, true: array) : float
Regression()
- __init__(true: array) : void
- compareReg(predictions: array, true: array) : float
Categorical()
- __init__(true: array) : void
- compareCat(predictions: array, true: array) : float
Binary()
- __init__(true: array) : void
- compareBin(predictions: array, true: array) : float
Loss
- trainable_layers : List
+ __init__() : void
+ regularization_loss(layer: Layer) : float
+ network_regularization_loss() : float
+ remember_trainable_layers(trainable_layers : List) : void
+ calculate(output: float, y: array, include_regularization: bool) : float
Categorical_CrossEntropy()
+ forward(y_pred: array, y_true: array) : float
+ backward(dvalues: array, y_true : array) : void
Binary_CrossEntropy()
+ forward(y_pred: array, y_true: array) : float
+ backward(dvalues: array, y_true : array) : void
MeanSquaredError()
+ forward(y_pred: array, y_true: array) : float
+ backward(dvalues: array, y_true : array) : void

Figure 3.2: UML diagram for the LSTAR framework.

A user may add custom modules to each class by following the same format they are set up in. Users may run individual modules through Google Colab requiring a CuPy import and setting the hardware accelerator to T4 GPU.

3.1.1 Datasets

We utilize four publicly available datasets for image classification as sources of example data: MNIST [26], CIFAR-10 [38], CIFAR-100 [38], and ImageNet [12]. This module is set up to return training and testing data for the relevant dataset when called. A user may import another dataset through the data loader module if desired.

Property	Datasets			
	MNIST	CIFAR-10	CIFAR-100	ImageNet
Size	70000 images	60000 images	60000 images	14197122 images
Image dimensions	28x28	32x32x3	32x32x3	Varies
Color channels	Grey-scale	RGB	RGB	RGB
Image format	IDX	PNG	PNG	JPEG & PNG
Label categories	10	10	100	21841
Label encoding	Integer	Integer	Integer	Integer

Table 3.1: Properties of utilized datasets

MNIST

The MNIST dataset is a widely used collection of 28×28 grey-scale images of handwritten digits (0-9), commonly employed for training and evaluating the image classification capability of machine learning models (see Appendix A.2.1) [26].

CIFAR-10

The CIFAR-10 dataset is a collection of 60,000 32×32 color images across 10 different classes of common objects with each class containing 6,000 images [23, 38]. This dataset is widely used as a benchmark for evaluating image classification algorithms and computer vision models (see Appendix A.2.2).

CIFAR-100

Bearing similarities to CIFAR-10, this dataset instead contains 100 classes containing 600 images each. There are 500 training images and 100 test images per class. The 100 classes within CIFAR-100 are further subdivided into 20 super-classes, with each image containing a “fine” label, the class to which it belongs, and a “coarse” label, the super-class to which it belongs [23, 38].

ImageNet

The ImageNet dataset is one of the largest publicly available datasets for image classification, consisting of over 14 million labeled images spanning thousands of categories (see Appendix

A.2.3) [12]. While used for testing hypotheses, it is not included in the dataset loader module due to its 150 GB size.

3.1.2 Network

The Network class is the core of the framework and serves as a container for organizing and managing the layers of the neural network. It instantiates an empty list to store model layers which can be added using the "add" method. The "set" method configures the loss function, optimizer, and accuracy metrics to be used. After configuration, the "compile_model" method sets up internal connections between layers, thus establishing the forward and backward propagation paths. The "train" method manages the training process by iterating over epochs and batches, performing the forward and backward passes, and updating parameters using the specified optimizer. Accuracy and loss are monitored and displayed periodically to assess model performance and convergence.

3.1.3 Layers

This subsection details the construction and functionality of the dense and dropout classes associated with the Layer module.

Dense

The Dense class is introduced as a fundamental building block of a multi-layer perceptron. It is initialized with input size, number of neurons, and regularization strengths for L_1 and

L_2 weights and biases. It also contains a backwards pass method to calculate gradients on parameters and input values. This takes an input of number of inputs, number of neurons, and optional inputs on weight and bias L_1 and L_2 regularizers. The constructor for this class defaults to a regularization strength of 0 if none is passed.

Dropout

The Dropout class is presented as a means of introducing randomness during the training process to mitigate overfitting. The forward pass generates a binary mask to randomly disable neurons while the backwards pass applies the mask to gradients on input values. This takes an input of a proportion corresponding to the number of neurons the user would like to disable.

3.1.4 Activation functions

This subsection details the construction and functionality of the classes associated with the Activation module.

Rectified linear unit (ReLU)

The forward method of the ReLU activation function computes the output by taking the element-wise maximum between 0 and the input values, thus ensuring non-negative outputs. In the backwards pass, it nullifies gradients where the output is less than or equal to zero.

Sigmoid

The forward method of the sigmoid activation function calculates the output using a sigmoid, squeezing input values to the range $[0, 1]$. In the backwards pass, it adjusts the gradients based on the derivatives of the sigmoid function.

Linear

This function maintains the input values as the output. The linear activation function caches and propagates input values and gradients respectively.

3.1.5 Loss functions

The parent class Loss contains common functionalities related to loss calculation, including regularization loss computation and memorization of trainable layers. Specific loss functions such as categorical cross entropy, binary cross entropy, and mean squared error are implemented as subclasses (see Appendix A.3).

3.1.6 Metrics

The parent class Metrics provides several tools to assess the performance of classification and regression models. It provides a generic method to compute the accuracy of predictions given both the predictions and ground truth values. There are separate methods available for regression, categorical classification, and binary classification tasks. These methods take an input of predictions and correct labels, then returns a float32 containing the proportion of correct predictions.

3.1.7 Optimizers

This subsection details the functionality of the Optimizer parent class. This class optimizes the network parameters during the training process to accelerate the convergence of the training process, provide smoother training, and contributes to the generalization capability of the trained model.

Adam

The Adam class implements the Adam optimization algorithm, an adaptive learning rate optimization algorithm combining ideas from both the momentum and RMSprop methods [51]. Key features of the Adam optimizer include adaptive learning rates for each parameter, momentum-like behavior to accelerate convergence, and RMSprop-like behavior to normalize gradients. Upon instantiating, the optimizer is initialized with default or user-defined parameters such as learning rate, epsilon, first moment exponential decay rate, and second moment exponential decay rate. During parameter updates, it computes new parameter values based on the Adam optimization algorithm, maintaining momentum estimates and caches of gradients for each parameter. Bias correction accounts for initialization bias. These updated parameters are then used to adjust the weights and biases of the model.

3.1.8 Adversarial attacks

This subsection details the functionality of the adversarial attack module. This module contains a set of adversarial example generation methods. These techniques are aimed at evaluating the robustness of machine learning models against adversarial perturbations to better understand their vulnerabilities and develop more robust defense mechanisms to mitigate potential risks. While there are a limited number of attacks provided, users may easily extend the module to cover additional attacks.

Fast gradient sign method

The FGSM function takes a model, input data, corresponding ground truth labels, and magnitude of perturbation as input parameters. The gradient of the model's loss function is calculated with respect to the input data and ground truth labels. It then computes the sign of the gradient and generates a perturbation by scaling the gradient sign with the epsilon value. The perturbation is then added to the input image and returned.

Projected gradient descent

Similarly, the PGD function takes a model, input data, corresponding ground truth labels, and magnitude of perturbation. It additionally takes step size and number of iterations as input parameters. The function iteratively applies small perturbations to the input image in the direction of the gradient sign while ensuring that the perturbed examples remain within a limit bounded by the magnitude of perturbation. The process is repeated for a specified number of iterations and returned.

Momentum iterative method

The MIM function is an extension of PGD with momentum. It takes similar input parameters to PGD, adding a decay factor to control the momentum's influence over time. In addition to perturbing the input data based on the sign of the gradient, it accumulates the gradient sign over iterations to guide the direction of the perturbation. The decay factor allows for greater convergence stability.

3.1.9 Default hyperparameter selection

The default hyperparameters chosen for each component of the framework were determined through a combination of known best practices, theoretical justifications, and practical constraints. These hyperparameters have demonstrated robust performance across many scenarios. Further, several default hyperparameters were set based on theories showing the ability of models to converge more effectively. Practical constraints, such as computational efficiency and ease of use, also guided selection of default hyperparameters.

3.2 Experimental setup

With our experimental framework, we define three hypotheses regarding the basis of adversarial examples and aim to more rigorously prove or disprove these hypotheses. This section is composed of several subsections containing the software version for the framework, hardware used, an overview of each hypothesis regarding the basis of adversarial examples, and the processes that were used to verify each hypothesis.

3.2.1 Software versions

The framework and evaluation methods proposed by this paper are built on Python 3.11.5, Nvidia CUDA Compiler driver version 12.1, CuPy version 13.0.0, TensorFlow version 2.15.0, cuTENSOR version 2.0, and cuDNN version 8.8.

3.2.2 Hardware

This framework and comparison models were evaluated on two machines. To ensure fairness, evaluation is performed using the time on the GPU as a metric.

Machine	Operating System	Processor	Graphics Processing Unit
A	Windows 10	AMD Ryzen 9 5900X	Nvidia RTX 2080 Ti
B	Ubuntu 22.04.4 LTS	Intel Core i7-6850K	Nvidia RTX 2080 Ti

Table 3.2: Machines for model evaluation

3.2.3 Classifier linearity

We ask whether adversarial perturbations are magnified by linear coefficients in the model, resulting in high misclassification probability. To weaken the linearity of model classifiers, we decrease the scale of the linear coefficients through L_2 normalization.

Four multi-layer perceptrons with back-propagation are created and trained on MNIST, CIFAR-10, and CIFAR-100, with a weight decay of 0.2 in optimizer settings. Each model consists of the standard 784 (MNIST), 3072 (CIFAR) input, or 65536 (ImageNet) input, two 200 neuron hidden layers, and 10 neuron output. This utilized categorical cross entropy as the loss function. Each classifier was trained for 10 epochs with a batch size of 64 using the Adam optimizer with a learning rate of 0.01. We then test FGSM, MIM, and PGD on each model to evaluate both the prediction accuracy on the generated adversarial test samples.

3.2.4 One-sum probability space

We ask whether classifiers are less robust to adversarial examples due to the need for output probabilities to sum to 1. Due to the need for a classifier to select a class, a classifier must select the category with highest confidence once it rules out all impossible classes. Rather than learning the features of the dataset, the model may instead be learning the posterior probability that an input belongs to one of the remaining categories.

To break the one-sum constraint, we switch the softmax activation function in the final layer for a sigmoid activation corresponding to each class. In doing so, each output probability instead belongs in the range $[0, 1]$ making the sum of probabilities instead range from $[0, n]$ where n is the number of classes. We once again train three multi-layer perceptrons with back-propagation on the three adversarial attacks and follow the constraints previously described.

3.2.5 Excessive number of classes

We ask whether classifiers trained on fewer target categories tend to be more robust than those trained on more target categories. This follows a similar idea to the curse of dimensionality in that it is more difficult for a model to learn patterns in data when there are too many dimensions to find patterns in. Due to the tendency of neural networks to classify an input space into path-connected regions, categories tend to overlap with each other as the number of categories increases. This overlap creates a large attack vector to create adversarial examples for. We train 9 classifiers using subsets of categories from MNIST and CIFAR-10.

The first classifier contains 2 classes {0, 1}, the second classifier contains 3 classes {0, 1, 2}, and so forth. The number of training samples is kept constant and the number of neurons in the final layer is kept equal to the number of categories on which the model is trained. The robustness of the classifier is then evaluated on FGSM, PGD, and MIM.

3.3 Usability test setup

The "thinking aloud" paradigm was utilized for this usability test. Due to the target audience of this framework being researchers in machine learning, the selection criteria for participants required they be a computer science student. Participants agreed to having their screen and audio recorded over a call and performed the test remotely. All materials required to run the framework were provided to the participants or available for free online. The framework source code and documentation were provided. Participants were asked to perform a series of tasks using the framework running on Google Colab utilizing Python 3.11.5 with a T4 GPU for hardware acceleration.

Prior to conducting the usability test, participants were asked a series of pre-test questions:

- How familiar are you with the concept of adversarial examples?
- What are your expectations from a modular test framework?
- What factors contribute most to your willingness to invest time into learning a new framework?

The tasks asked of the participants were as follows:

- Create a MLP with back-propagation, train it on MNIST, and evaluate the model. Leave parameters at their default values.
- Extend the previously built model by evaluating it on adversarial examples generated using FGSM with default parameters.
- Create a new MLP classifier trained on CIFAR-10 instead. Use the following parameters: 10 epochs, 32 batch size, 1e-4 learning rate, and 1e-8 decay.
- Utilizing documentation on the Basic Iterative Method adversarial attack, implement a custom adversarial attack within the framework. Evaluate on the CIFAR-10 dataset with $\epsilon = 0.1$.

After conducting the usability test, participants were asked a series of post-test questions. Scale questions were evaluated on a scale ranging from strongly positive, positive, neural, negative, and strongly negative:

- How would you rate your overall experience with the adversarial machine learning framework and corresponding documentation? Why?
- What aspects do you feel were done well?
- What aspects do you feel could use improvement?
- One alternative to this framework involves using Keras and the Adversarial Robustness Toolbox (ART) together [35]. Read through the corresponding documentation. How would you compare this framework to Keras and ART?
- For a student who may be learning the basics of machine learning for the first time, do you think this would be a good first resource to learn from? Why or why not?

Chapter 4
RESULTS

In this section, we analyze whether the impact of linear coefficients in classifiers results in low adversarial robustness using a more robust testing method utilizing multiple adversarial attacks and datasets that more closely reflect real-world data. Using the same testing methodology, we evaluate whether classifiers tendency to select adversarial samples is due to the one-sum constraint of its probabilities. Additionally, we evaluate whether models trained on a high number of target categories tend to be less robust to adversarial attacks than those trained on fewer target categories. We assess the results from our usability test on the LSTAR framework and analyze the findings to determine the effectiveness and areas of improvement for the framework.

4.1 Classifier linearity

Figure 4.1 plots the average accuracy for multi-layer perceptron classifiers with back-propagation trained on the MNIST dataset on the framework. We also perform the same test on the TensorFlow framework in Figure 4.2. Figure 4.3 shows this test performed under MNIST, CIFAR-10, CIFAR-100, and ImageNet. As the adversarial perturbation increases, we note that the error bars indicating minimum and maximum range tend to increase in size, indicating varying degrees of susceptibility to adversarial perturbation. When applying regularization to break classifier linearity, the difference in the accuracy curves between the baseline and the regularized models indicates that the decrease in accuracy is as a result of the linearity of the classifier.

We note an inflection point occurring around $\epsilon = 0.10$. This inflection point reflects the shift of the model decreasing its confidence in the correct class below a threshold and an increase in confidence of incorrect classes. Prior to reaching this threshold, the model displays an acceptable level of performance in correctly identifying the input data. As the attack strength surpasses this range, the curvature becomes concave up, indicating a threshold at which the model has been significantly compromised.

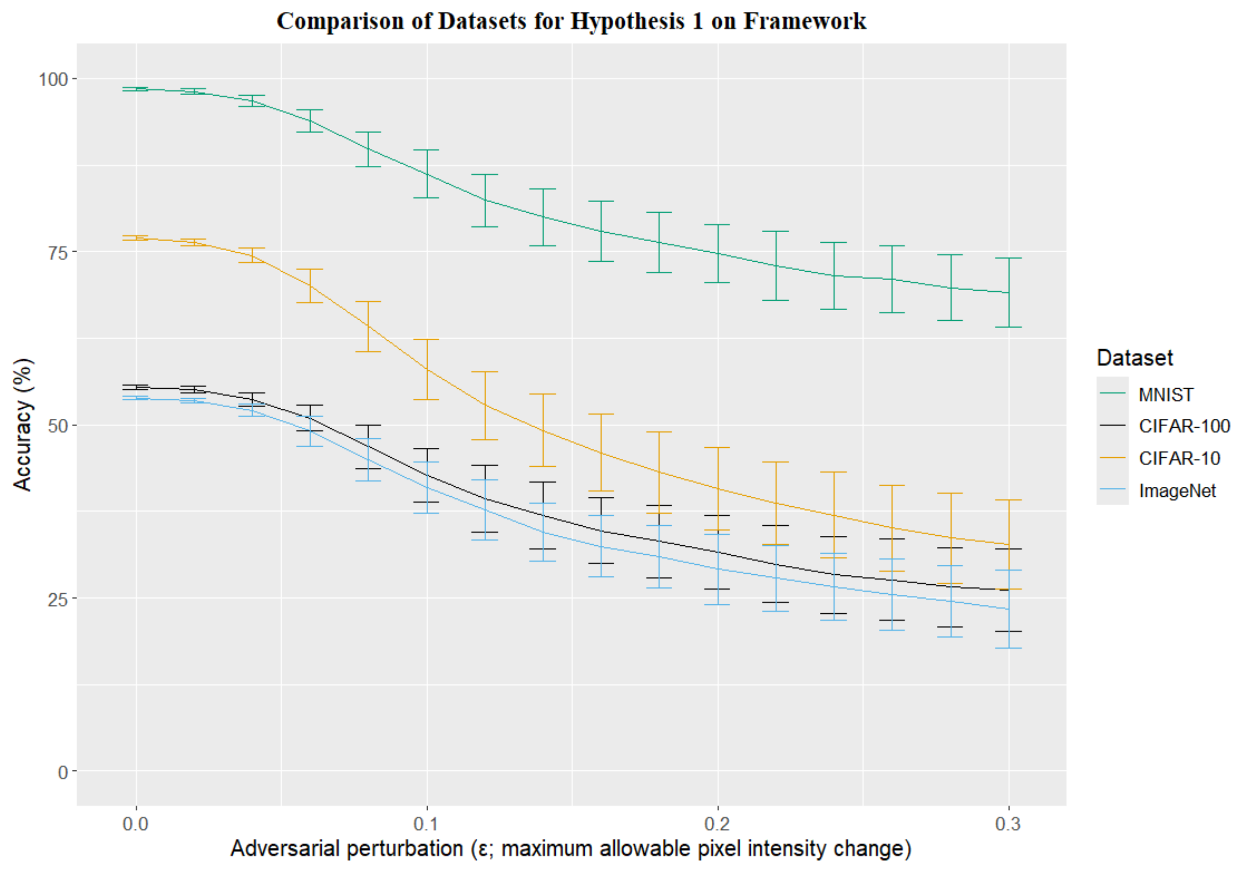


Figure 4.1: Graph depicting the relationship between classifier accuracy and adversarial strength across different adversarial attacks on LSTAR. Baseline without forcing linear coefficients.

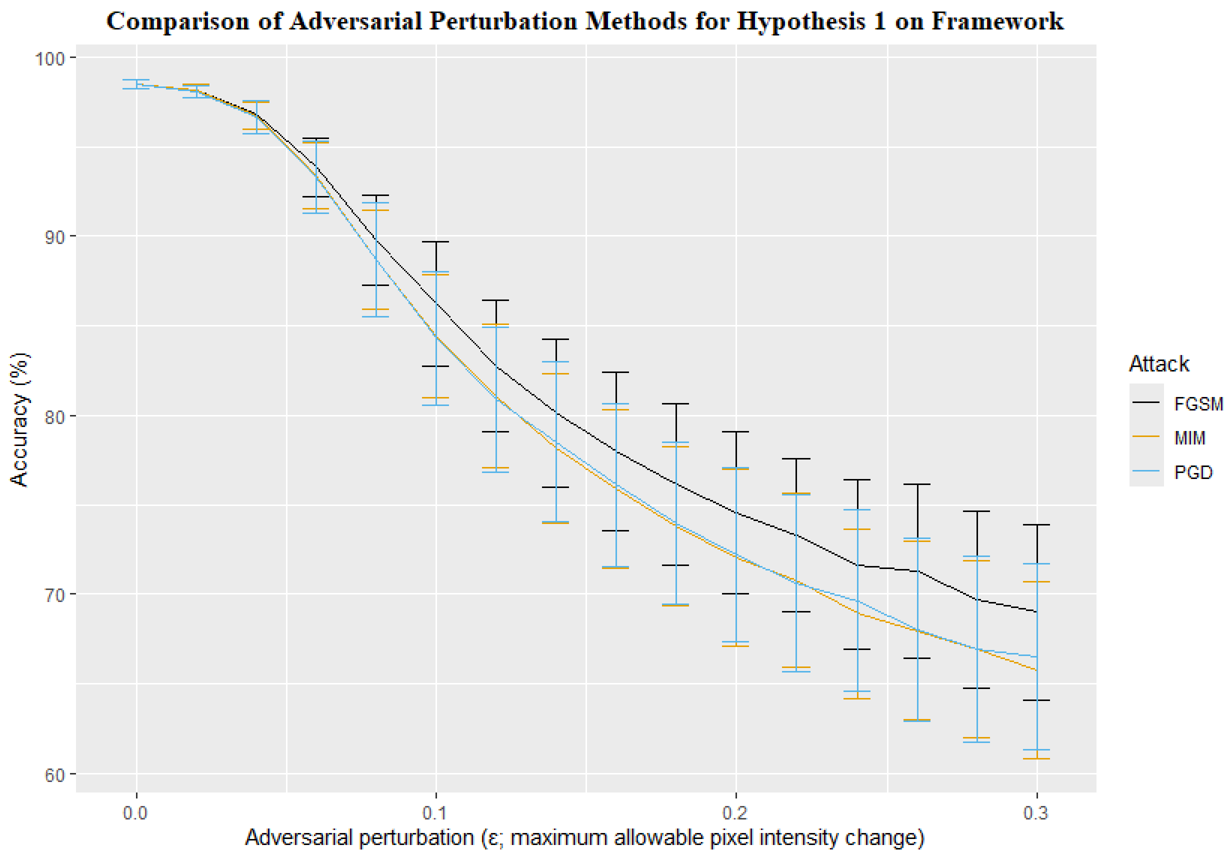


Figure 4.2: Graph depicting the relationship between classifier accuracy and adversarial perturbation strength across different adversarial attacks on LSTAR. Forced linear coefficients.

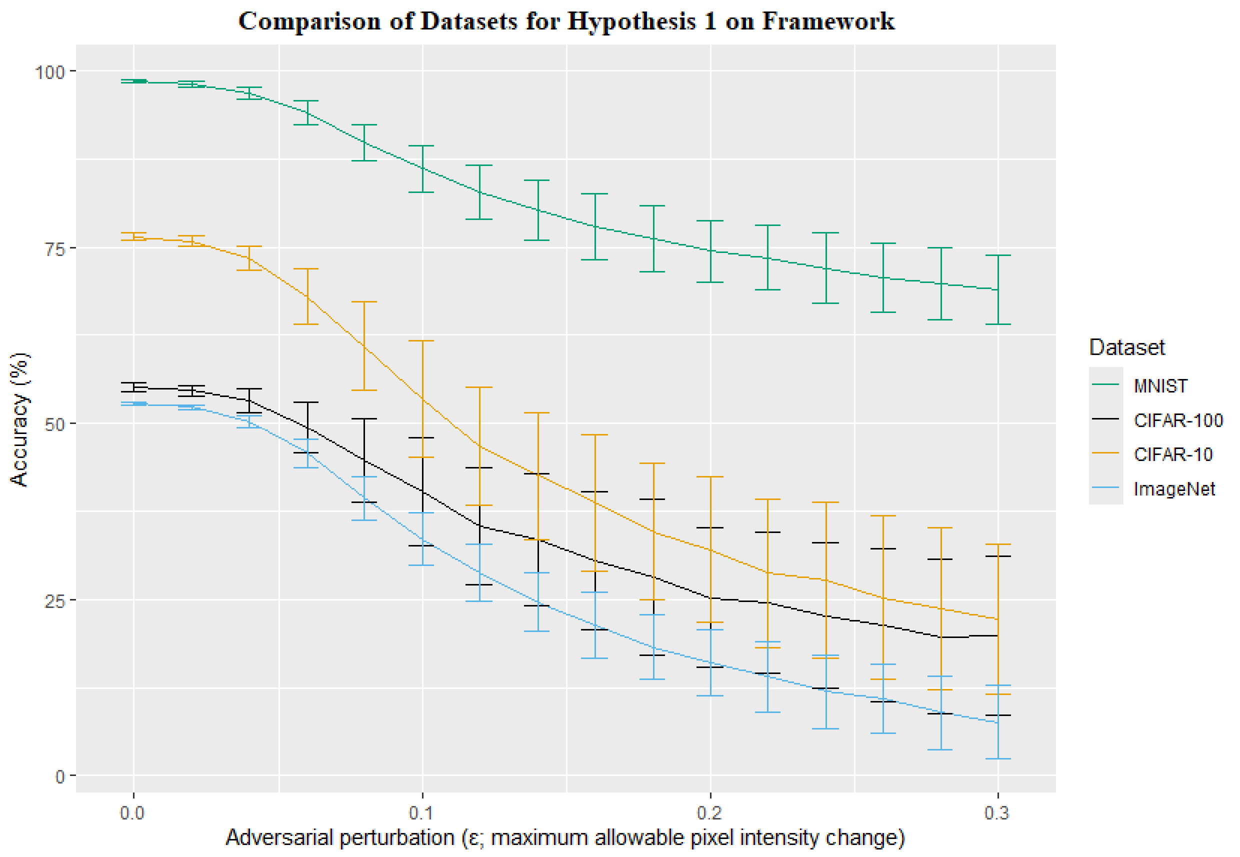


Figure 4.3: Graph depicting the relationship between classifier accuracy and adversarial perturbation strength across different datasets on LSTAR.

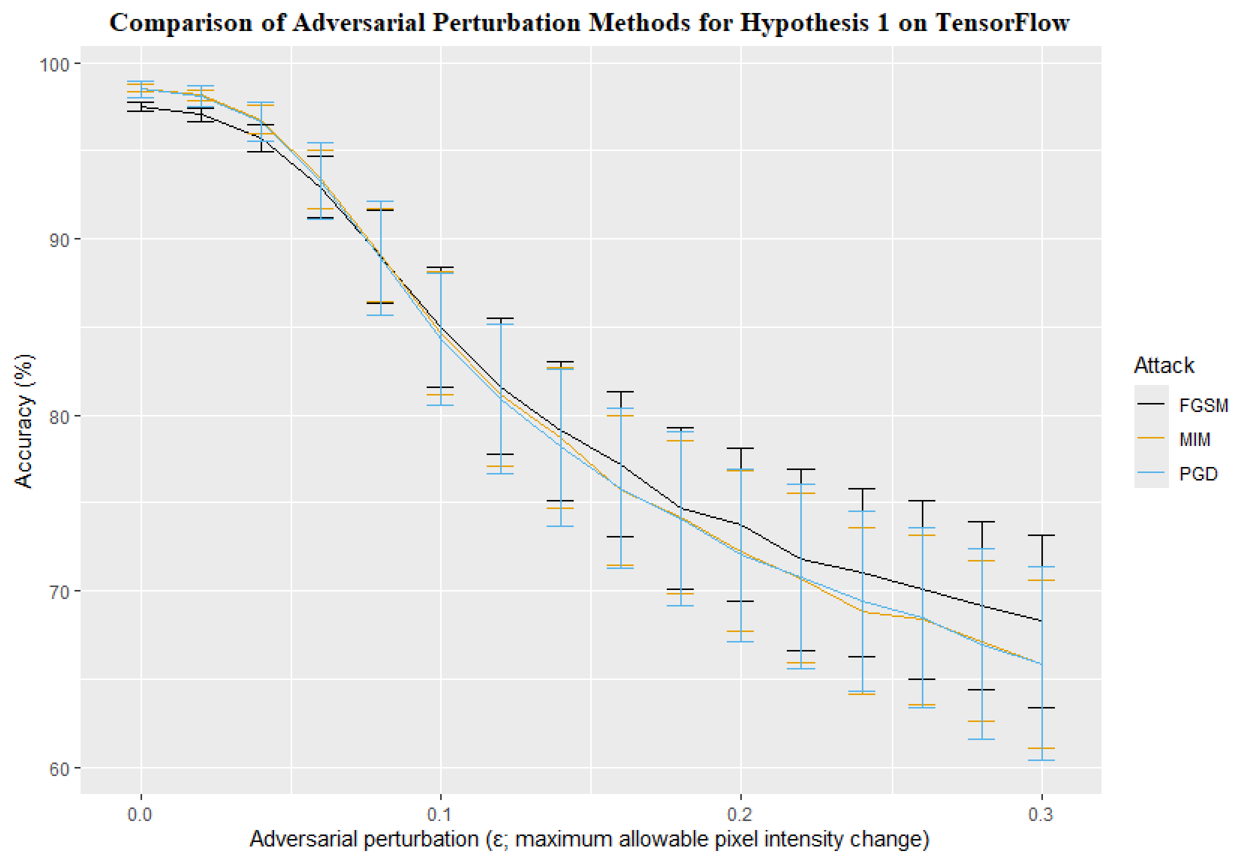


Figure 4.4: Graph depicting the relationship between classifier accuracy and adversarial perturbation strength across different adversarial attacks on TensorFlow and the MNIST dataset.

4.2 One-sum probability space

With no pooling class for situations where models are unable to achieve high confidence in any one target category, the classifier must pick the category that it has the highest confidence in, even if that confidence is comparatively low due to adversarial attacks. We notice a similar situation to the classifier linearity hypothesis. We note an inflection point around $\epsilon = 0.12$. Under attacks with low perturbation strength, as adversarial perturbation increases, the attack tends to decrease the probability in the correct category until the output classification is shifted. Under the one-sum constraint, this forces an increase in probability in another category.

Adversarial attacks can be assumed to have two generally distinct phases. In the first phase, attacks primarily decrease the probability of selecting the correct output class until the output is shifted. In the second phase, the prediction confidence on some incorrect class is shifted up. Thus, the one-sum constraint inherent in softmax activation and cross entropy loss accelerates the decreasing of confidence in the correct target class. As the attack strength increases, the classifier is required to push predictions into incorrect class categories that sum to one in phase 2. However, under binary cross entropy and sigmoid activation on each neuron, the prediction confidence for each class can be shifted without consideration for other classes. This in turn causes models to be more robust in that the effects of adversarial attacks only affect a small number of classes rather than affecting multiple classes simultaneously.

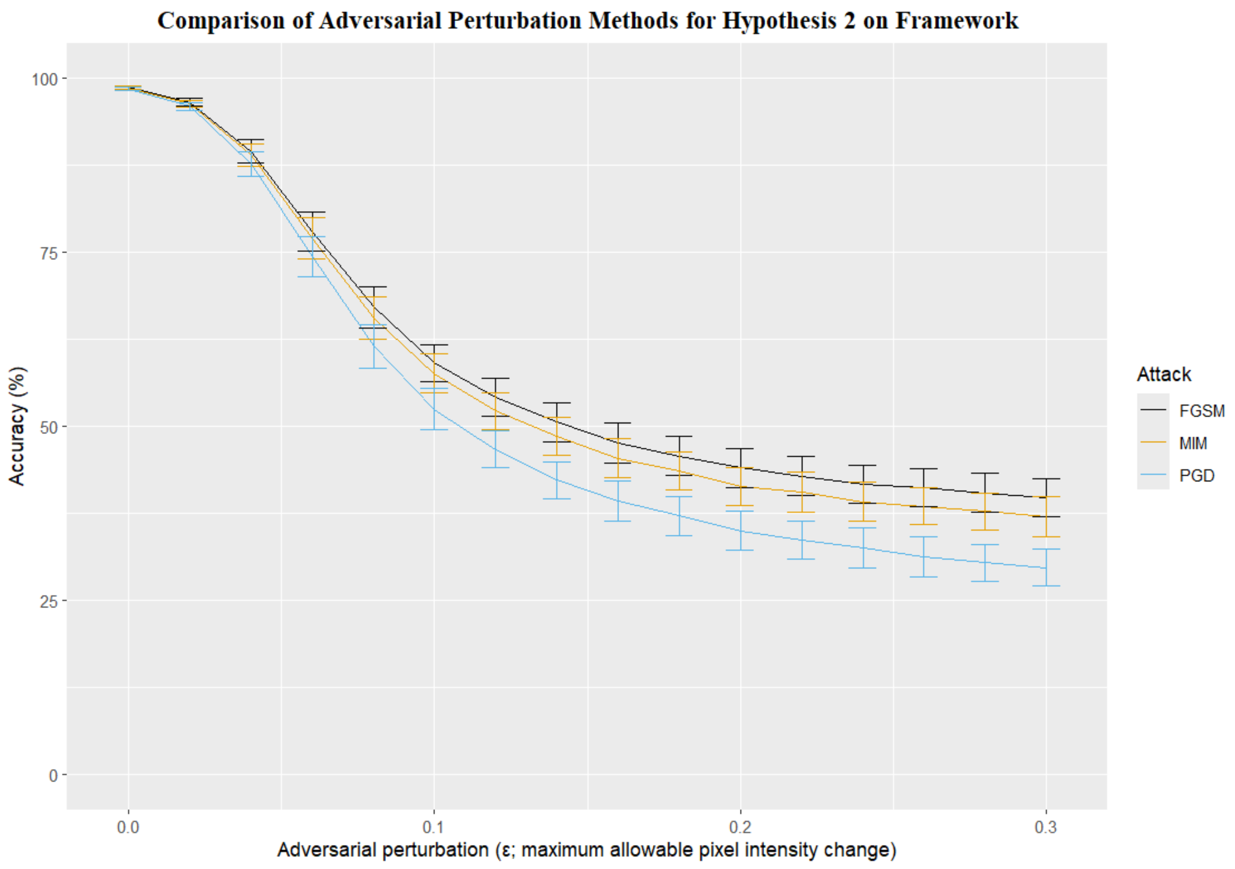


Figure 4.5: Graph depicting one-sum constrained models on LSTAR. Baseline for comparison.

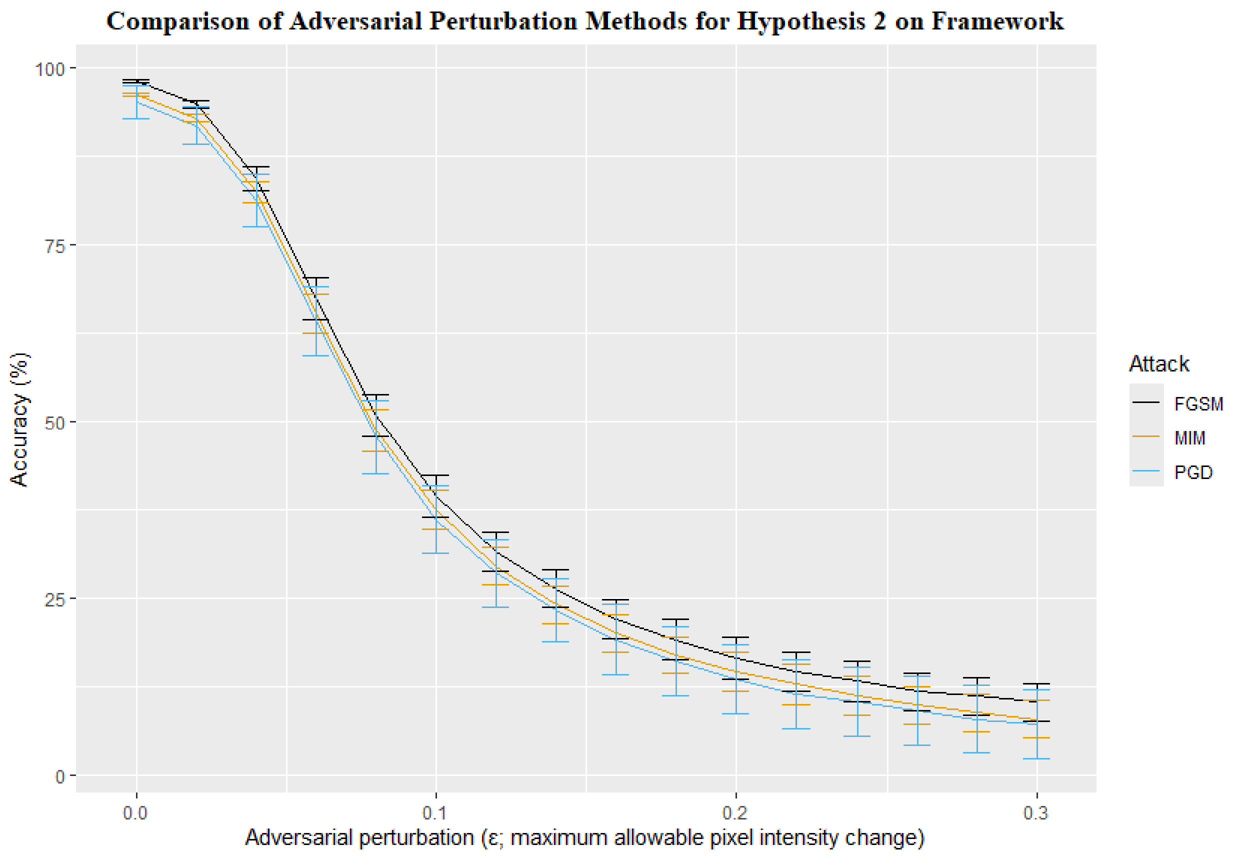


Figure 4.6: Graph depicting the relationship between one-sum constrained models and those without constraints on adversarial robustness using LSTAR.

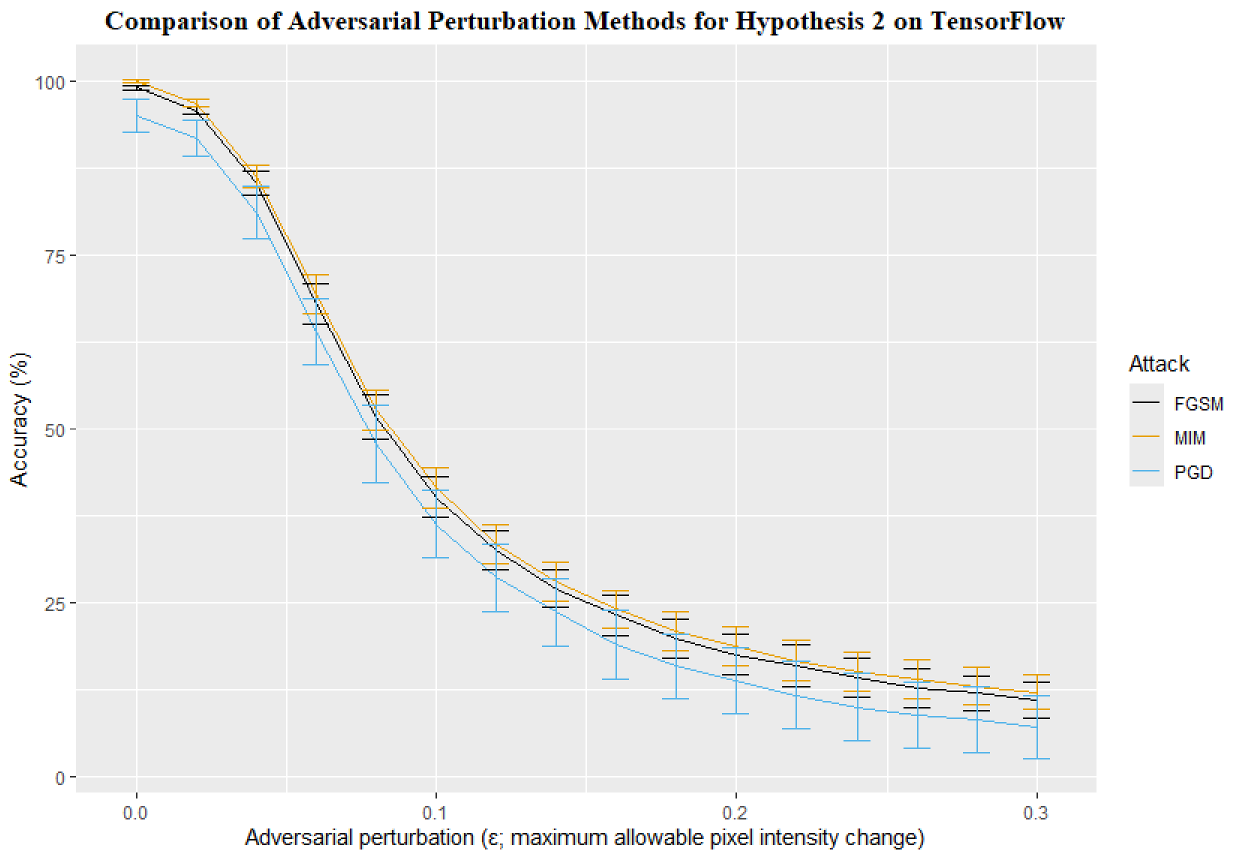


Figure 4.7: Graph depicting the relationship between one-sum constrained models and those without constraints on adversarial robustness using TensorFlow.

4.3 Excessive number of classes

We train a total of 9 classifiers on subsets of categories on the MNIST and CIFAR-10 datasets by removing all but the first two classes and adding categories sequentially. The total number of training samples per class is kept constant. The number of neurons in the last layer is changed to be equal to the number of classes being trained on.

With more classes to differentiate between, the complexity provides adversaries with more surface area upon which they may exploit model vulnerabilities. Further, due to the dimensionality of the input space increasing with the addition of more classes, it creates further challenges for the model to generalize well to unseen data due to the sparsity of data within the high dimensional input space. This also creates a higher likelihood of ambiguity between similar classes which may be exploited by a subtle shift of the decision boundary between closely related classes. Figure 4.8 indicates a significant loss in robustness as the number of classes increases.

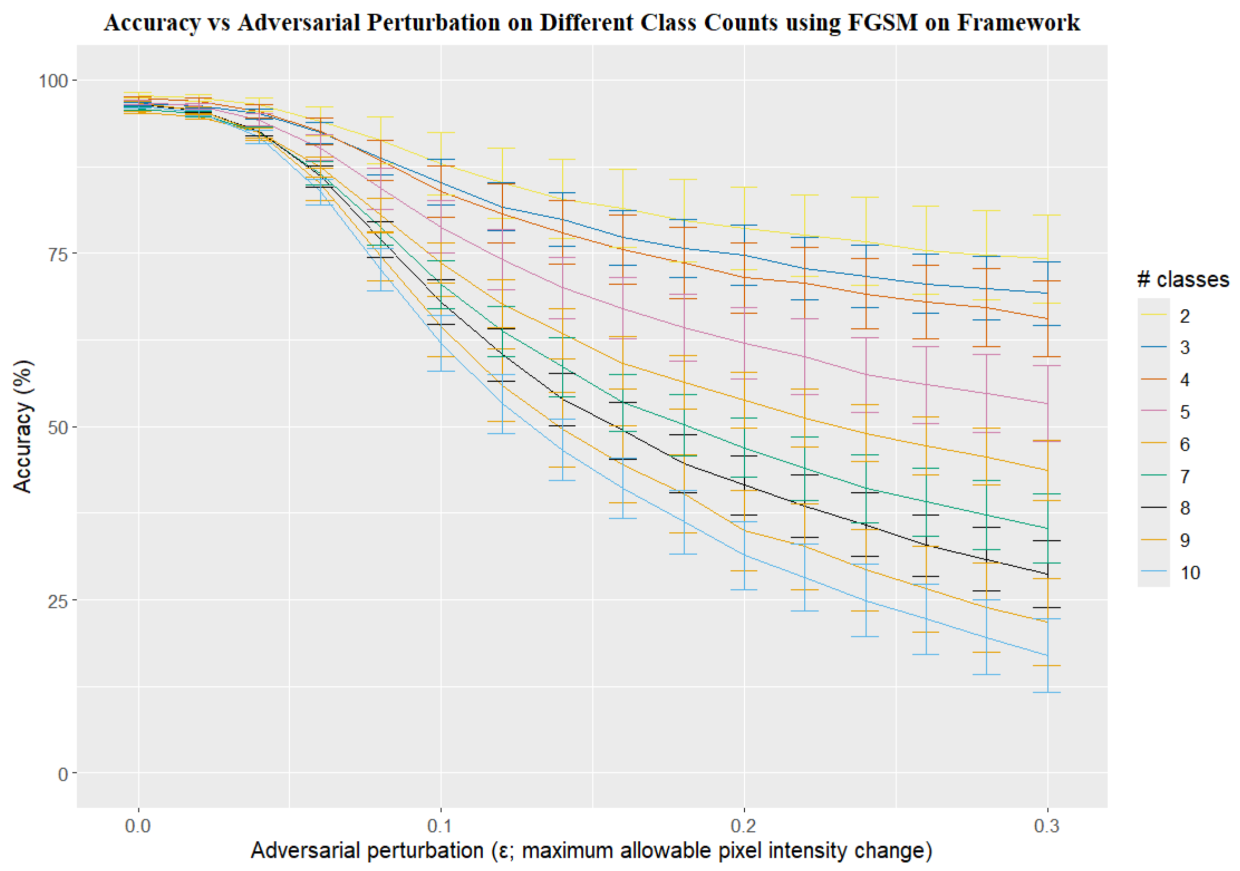


Figure 4.8: Graph depicting the relationship between accuracy vs adversarial perturbation on varying number of classes on the CIFAR-10 dataset under FGSM on LSTAR.

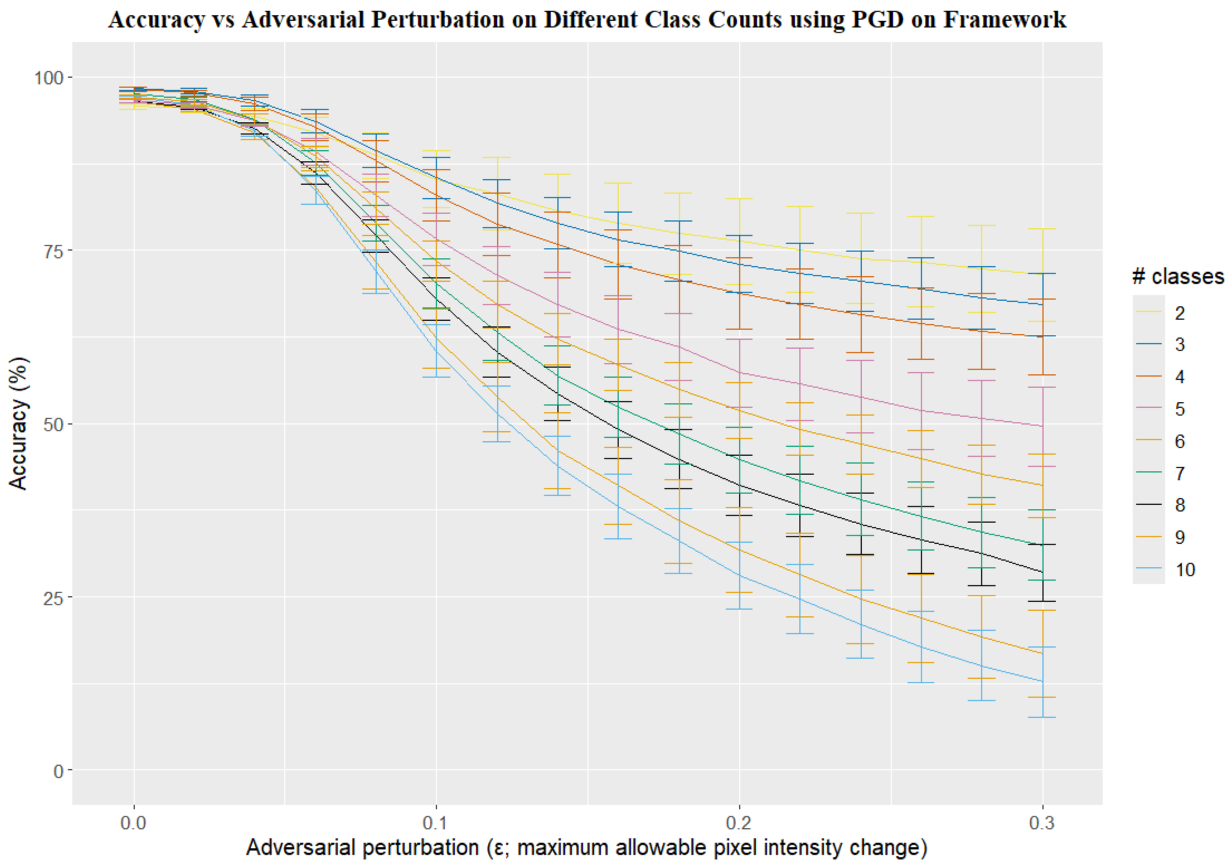


Figure 4.9: Graph depicting the relationship between accuracy vs adversarial perturbation on varying number of classes on the CIFAR-10 dataset under PGD on LSTAR.

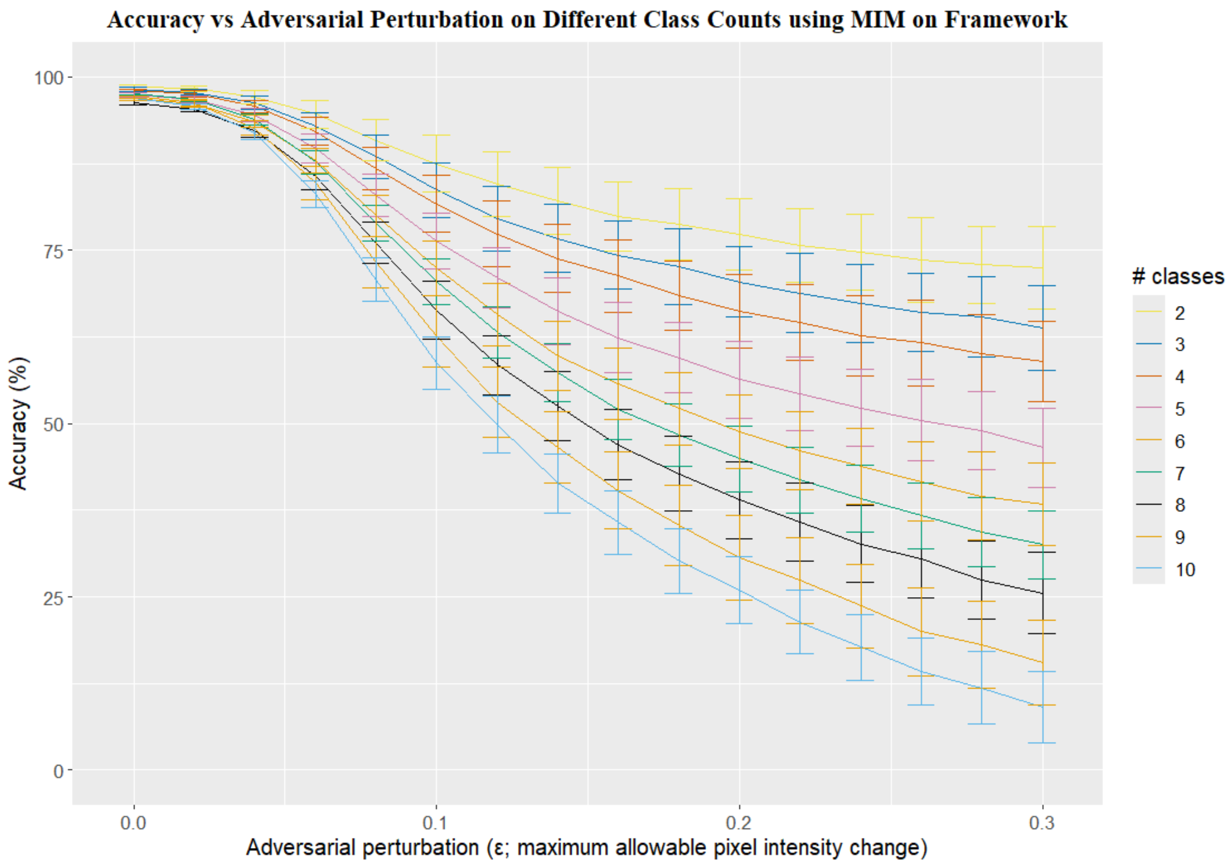


Figure 4.10: Graph depicting the relationship between accuracy vs adversarial perturbation on varying number of classes on the CIFAR-10 dataset under MIM on LSTAR.

4.4 Usability test

4.4.1 Description of participants

12 computer science students agreed to take part in this usability study. All participants participated in all parts of the study.

From the pre-test questionnaire, observations are as follows:

- Responses indicated a range of familiarity with adversarial examples. Some participants indicated a strong understanding, while others had limited to no exposure to the topic.
- A variety of expectations were expressed, including flexibility and customization support.
- Major factors influencing willingness to learn a new framework include: ease of use, intuitive design, and availability of learning resources.

4.4.2 Presentation of findings

- Create a MLP with back-propagation, train it on MNIST, and evaluate the model. Leave parameters at their default values.
- Extend the previously built model by evaluating it on adversarial examples generated using FGSM with default parameters.

- Create a new MLP classifier trained on CIFAR-10 instead. Use the following parameters: 10 epochs, 32 batch size, 1e-4 learning rate, and 1e-8 decay.

- Utilizing documentation on the Basic Iterative Method (BIM) adversarial attack, implement a custom adversarial attack within the framework. Evaluate on the CIFAR-10 dataset with $\epsilon = 0.1$.

Task completion time (mins)	Average	Minimum	Maximum
Task 1	16.86	6.90	31.65
Task 2	7.41	3.80	21.30
Task 3	5.66	4.20	11.80
Task 4	38.08	21.90	78.15

Table 4.1: Average, minimum, and maximum completion time for each task across all participants in the usability test.

4.4.3 Observations

Based on the usability test, participants demonstrated varying levels of efficiency and proficiency across the four tasks. In this section, we analyze observations from each task. All participants were able to complete all sections of each task.

Task 1

- Most participants utilized the example section from the documentation to perform this task.
- Few participants relied solely on the documentation. Those that did took longer to complete task 1, but completed tasks 2, 3, and 4 more quickly comparatively.
- About half of the participants expressed surprise that metrics output does not have visuals such as a graphs or charts.
- Several participants expressed confusion regarding what values to put for the number of neurons and inputs for each layer.

Task 2

- Participants easily found the FGSM section within the documentation and were able to implement it quickly.
- Participants with little to no exposure to adversarial examples tended to ask for more context regarding what the FGSM, MIM, and PGD methods do to image inputs.

Task 3

- Participants utilized documentation to perform this task with relative ease.

- Several participants expressed a desire for further explanations regarding what each hyperparameter changes within the optimizer function.

- Participants encountered difficulties in selecting what value to use for each hyperparameter.

Task 4

- Participants experienced varying levels of difficulty in implementing an additional adversarial attack.

- Participants were able to use existing adversarial attack implementations as a guidepost for how to implement the BIM attack.

- There were many participants who struggled with the format of image inputs and getting CuPy to successfully read the images into GPU memory.

- The long average completion time reflects the difficulty required to understand and implement an adversarial attack. This highlights the need for additional support for users attempting these tasks within the framework.

4.4.4 Comparison to hypotheses and expectations

It was expected that participants would predominantly rely on the example section of the documentation to perform tasks as its code snippets were very easy to understand and modify as necessary. The observations align with the expectations as most participants relied on this section to perform various tasks on the framework. This was particularly evident on tasks 2 and 3, where participants were able to complete the tasks relatively quickly. However, participants not utilizing the examples section performing better on subsequent tasks was unexpected, signaling a need to focus more on documentation readability. The long completion time on task 4 was expected due to the relative difficulty of the task. The largest problem was ensuring that CuPy would properly send input data to GPU memory. Due to the precise nature of inputs required by CuPy, participants found it relatively difficult to implement BIM, even with FGSM, MIM, and PGD examples. This possibly signals a need to create a method to move away from using CuPy or have a helper method to automatically convert inputs into those accepted by CuPy.

4.4.5 Post-test questionnaire

Using +2 for strongly positive, +1 for positive, 0 for neutral, -1 for negative, and -2 for strongly negative, the average user experience was averaged to 0.4167 indicating slightly positive sentiment overall. This suggests positive aspects of the framework and several areas of improvement.

All participants found the examples to be very helpful. However, they wanted more thorough explanations on the functionality of each module. Improvement of documentation and supplementary material was suggested. Participants found the parts of this framework that had equivalencies to Keras and ART to be relatively equal, indicating the framework offers comparable functionality to existing tools and libraries in a much more lightweight fashion.

Responses averaged 0.6 towards using this framework as a first resource to learn from, suggesting that while there may be areas for improvement, the framework has the potential to serve as a valuable first resource for beginners in the field.

Chapter 5
DISCUSSION

Our work presents two major contributions. We first robustly evaluate three hypotheses regarding the basis of adversarial examples through using datasets that more closely resemble real-world data and by using more advanced adversarial attacks. Further, we implement these evaluations through a novel modular framework that allows standardization of tests for adversarial robustness. With research into the basis of adversarial examples being underdeveloped compared to research into adversarial attacks and defenses, our framework serves as a valuable starting point for advancing this area of study.

From our tests in hypothesis 1, we evaluated whether classifier linearity results in greater misclassification. In doing so, we train a multi-layer perceptron with back-propagation on several datasets and plot the average accuracy for each classifier at various adversarial perturbation strengths. Based on the accuracy curves, we see a correlation between a loss of accuracy on adversarial examples and misclassification, but based on this alone, we are unable to draw a conclusion regarding whether it is a direct cause.

Hypothesis 2 asked whether classifiers tended to default to adversarial examples due to the need for output probabilities to add up to 1. Rather than learning the general features of the dataset, the classifier may instead be learning a probability that an input must belong to one of the target categories. We train a new multi-layer perceptron with back-propagation on several datasets and switch the loss to binary cross-entropy which forces each output class to output between [0, 1]. The results of this test revealed a much slower accuracy loss under weaker adversarial perturbation. However, when the attack becomes stronger, there is an inflection point in which the accuracy begins dropping considerably more quickly. Thus, we can treat these adversarial attacks as having two phases. The first phase tends to decrease the probability of selecting the correct category until the output label is changed. Under the one-sum constraint, this must increase the probability of selecting other categories. However, when this constraint is broken, we see a slower phase 1 in the new model. The second phase then occurs when model prediction confidence for incorrect classes has become higher than that for the correct class.

The third hypothesis is essentially a question of the curse of dimensionality: does a high number of target categories cause a model to become less robust? As expected, as a network classifies the input space into regions, their boundaries tend to be close to each other. As such, more categories create more boundaries between different classes, and this in turn creates more surface area upon which adversarial attacks can occur, as evident in Figure 4.6.

We performed a usability test as part of our study with the goal of assessing the practicality and user experience of our framework. With the help of 12 computer science students, we had them perform various tasks using the framework to glean insights into expectations, usability issues, and directions of improvement. Feedback from the participants showed the importance of comprehensive documentation, with a strong need for improved user guidance regarding module functionality and hyperparameter settings. Despite these challenges, participants generally found the framework to be helpful and indicated a positive overall experience.

The modular nature of the framework opens possibilities for its use in various other applications beyond adversarial testing. For example, the architecture could be leveraged for tasks such as distributed computing, where modules could be parallelized across multiple computing nodes to improve efficiency.

5.1 Comparison to TensorFlow

We performed the same tests on our framework as we did on TensorFlow to have a fair comparison between the two frameworks. We found that both frameworks achieved similar levels of accuracy and performance across the various datasets. They exhibited comparable levels of robustness against adversarial attacks. TensorFlow performed marginally worse when scaling up to the size of the ImageNet dataset, though the difference in execution time over multiple trials was negligible. Our framework allowed greater flexibility in understanding what exactly is occurring within the code. It also allows users to easily share modules. However, due to the framework's reliance on CuPy, it is more difficult for users to extend upon our framework in comparison to TensorFlow.

Due to high variance at higher adversarial perturbation levels for both the framework and TensorFlow, we removed the upper third of the adversarial perturbation data for this analysis. Based on a MANOVA analysis conducted at a confidence level of 0.1, we found there to be no statistically significant differences between the performance metrics of models trained on our framework compared to TensorFlow. The absence of statistically significant differences suggests that both options are viable choices for conducting machine learning research.

5.2 Limitations

While our framework provides a good starting point for building upon, it has several weaknesses that must be addressed before considering its adoption. As of now, the framework only supports multi-layer perceptrons. Incorporating other architectures such as convolutional neural networks and others would provide more options to users. The framework also only implements a few adversarial attacks. Including a much broader range of attack methods would help provide a more thorough environment for users to test in. There is a lack of any implemented adversarial defense mechanisms in the framework which would allow users to evaluate their effectiveness against adversarial attacks. While the framework provides metrics output, there is room for improvement in visualization. Methods to display confusion matrices and ROC/AUC curves would allow users to better understand the behavior of their models. While the reliance on CUDA is expected within the field, the framework relies on CuPy which necessitates a Nvidia GPU due to the lack of a proper equivalent on AMD or Intel platforms. Providing alternative GPU-accelerated libraries could improve accessibility across different hardware configurations. Further, this framework operates on a single-node architecture unless the modules are operated on separately, which limits scalability.

Chapter 6
CONCLUSION

We propose a modular framework for adversarial testing. The proposed framework combines the advantages of being easy to add to with being easy to share between researchers. We verify or partially verify several possible causes of adversarial example vulnerability: model linearity, one-sum constraints, and excessive target classes. Experimental results on benchmark tests demonstrate that the proposed approach achieves equivalent performance to widely used frameworks. Our framework provides a streamlined and user-friendly approach to image classification that stands out for its simplicity and modularity. We also perform a usability test to evaluate the usability of the framework for researchers with varying levels of expertise, indicating that novice researchers can adapt to and utilize the framework effectively.

6.1 Practical implications

This framework offers a versatile tool for exploring and evaluating various research questions related to adversarial machine learning. By providing a modular platform, our framework enables researchers to focus their efforts on experimental design and analysis rather than getting slowed by technical implementations inherent in particularly unique models. It facilitates easily reproducible research by providing standardized modules and workflows for conducting experiments, promoting transparency in research practices. Additionally, the modularity of the framework allows researchers to easily customize and extend its functionality to suit their specific research needs. It can be leveraged as a flexible tool for conducting experiments into the state-of-the-art in adversarial machine learning. Our usability test indicated good effectiveness across users with varying levels of expertise, suggesting it can serve as an accessible and practical tool for both novice and experienced researchers alike.

6.2 Future work

Beyond the limitations in section 5.2, several possible directions exist for further development of our modular framework. One method involves extending the framework to support a broader range of network architectures beyond multi-layer perceptrons to address a wider range of research questions. Modules could be extended to accommodate diverse types of data, such as text, audio, or time series data, to allow researchers to study adversarial impacts on various kinds of data. Efforts to enhance the usability of the framework through improvements to the user interfaces and graphical metrics outputs would make the framework more accessible to researchers.

This framework serves as a starting point to provide researchers with a modular framework to focus on experimental design and analysis. By offering an adaptable platform, we hope that it will inspire researchers to build upon this framework to explore a wider range of research questions on the basis of adversarial examples and empower researchers with varying levels of expertise to engage on adversarial machine learning research.

BIBLIOGRAPHY

[1] Răzvan Andonie. Hyperparameter optimization in learning systems. *Journal of Membrane Computing*, 1(4):279–291, December 2019.

[2] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing Robust Adversarial Examples. *International conference on machine learning*, 2018.

[3] Muhammad Farrukh Bashir, Hamza Arshad, Abdul Rehman Javed, Natalia Kryvinska, and Shahab S. Band. Subjective Answers Evaluation Using Machine Learning and Natural Language Processing. *IEEE Access*, 9:158972–158983, 2021.

[4] Melika Behjati, Seyed-Mohsen Moosavi-Dezfooli, Mahdieh Soleymani Baghshah, and Pascal Frossard. Universal Adversarial Attacks on Text Classifiers. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7345–7349, Brighton, United Kingdom, May 2019. IEEE.

[5] Richard Bellman and Robert Kalaba. A MATHEMATICAL THEORY OF ADAPTIVE CONTROL PROCESSES. *Proceedings of the National Academy of Sciences*, 45(8):1288–1290, August 1959.

[6] Bernd Bischl, Martin Binder, Michel Lang, Tobias Pielok, Jakob Richter, Stefan Coors, Janek Thomas, Theresa Ullmann, Marc Becker, Anne-Laure Boulesteix, Difan Deng, and Marius Lindauer. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *WIREs Data Mining and Knowledge Discovery*, 13(2):e1484, March 2023.

[7] Wieland Brendel, Jonas Rauber, Matthias Kümmerer, Ivan Ustyuzhaninov, and Matthias Bethge. Accurate, reliable and fast robustness evaluation. In *Advances in neural information processing systems*, December 2019.

[8] Tom B. Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial Patch, May 2018. arXiv:1712.09665 [cs].

[9] Nicholas Carlini. A Complete List of All Adversarial Example Papers, June 2019.

[10] Joshua Chuah, Uwe Kruger, Ge Wang, Pingkun Yan, and Juergen Hahn. Framework for Testing Robustness of Machine Learning-Based Classifiers. *Journal of Personalized Medicine*, 12(8):1314, August 2022.

[11] Islam Debicha, Richard Bauwens, Thibault Debatty, Jean-Michel Dricot, Tayeb Kenaza, and Wim Mees. TAD: Transfer Learning-based Multi-Adversarial Detection of Evasion Attacks against Network Intrusion Detection Systems. *Future Generation Computer Systems*, 138:185–197, January 2023. arXiv:2210.15700 [cs].

[12] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. *IEEE conference on computer vision and pattern recognition*, 2009.

[13] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting Adversarial Attacks with Momentum. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9185–9193, Salt Lake City, UT, June 2018. IEEE.

[14] Dunarea de Jos University of Galati, Romania, Adrian Micu, Marius Geru, Transilvania University of Brasov, Romania, Alexandru Capatina, Dunarea de Jos University of Galati, Romania, Avram Constantin, Dunarea de Jos University of Galati, Romania, Robert Rusu, Dunarea de Jos University of Galati, Romania, Andrei Alexandru Panait, and Transilvania University of Brasov, Romania. Leveraging e-Commerce Performance through Machine Learning Algorithms. *Annals of Dunarea de Jos University of Galati. Fascicle I. Economics and Applied Informatics*, 25(2):162–171, July 2019.

[15] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust Physical-World Attacks on Deep Learning Models, April 2018. arXiv:1707.08945 [cs].

[16] Tiantian Feng, Rajat Hebbar, Nicholas Mehlman, Xuan Shi, Aditya Kommineni, and Shrikanth Narayanan. A Review of Speech-centric Trustworthy Machine Learning: Privacy, Safety, and Fairness. *APSIPA Transactions on Signal and Information Processing*, 12(3), 2023.

[17] Yang Feng, Qingkai Shi, Xinyu Gao, Jun Wan, Chunrong Fang, and Zhenyu Chen. DeepGini: Prioritizing Massive Tests to Enhance the Robustness of Deep Neural Networks, June 2020. arXiv:1903.00661 [cs].

[18] Micah Goldblum, Liam Fowl, Soheil Feizi, and Tom Goldstein. Adversarially Robust Distillation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):3996–4003, April 2020.

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

[20] Jianmin Guo, Yue Zhao, Xueying Han, Yu Jiang, and Jiaguang Sun. RNN-Test: Adversarial Testing Framework for Recurrent Neural Network Systems. *IEEE Transactions on Software Engineering*, 2021.

[21] Jun Guo, Wei Bao, Jiakai Wang, Yuqing Ma, Xinghai Gao, Gang Xiao, Aishan Liu, Jian Dong, Xianglong Liu, and Wenjun Wu. A Comprehensive Evaluation Framework for Deep Model Robustness, November 2022. arXiv:2101.09617 [cs].

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, Las Vegas, NV, USA, June 2016. IEEE.

[23] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. PhD thesis, University of Toronto, 2009.

[24] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world, February 2017. arXiv:1607.02533 [cs, stat].

[25] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing Network-Wide Traffic Anomalies. *ACM SIGCOMM computer communication review*, 2004.

[26] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[27] Honglin Li, Yifei Fan, Frieder Ganz, Anthony Yezzi, and Payam Barnaghi. Verifying the Causes of Adversarial Examples, October 2020. arXiv:2010.09633 [cs].

[28] Yujie Liu, Shuai Mao, Xiang Mei, Tao Yang, and Xuran Zhao. Sensitivity of Adversarial Perturbation in Fast Gradient Sign Method. In *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 433–436, Xiamen, China, December 2019. IEEE.

[29] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. *stat*, 2018.

[30] Vivien Marx. Laurent Cognet. *Nature Methods*, 15(6):397–397, June 2018.

[31] John Morris, Eli Lifland, Jin Yong Yoo, Jake Grigsby, Di Jin, and Yanjun Qi. TextAttack: A Framework for Adversarial Attacks, Data Augmentation, and Adversarial Training in NLP. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 119–126, Online, 2020. Association for Computational Linguistics.

[32] Saadat Nazirova. Survey on Spam Filtering Techniques. *Communications and Network*, 03(03):153–160, 2011.

[33] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading Digits in Natural Images with Unsupervised Feature Learning. *NIPS workshop on deep learning and unsupervised feature learning*, 2011.

[34] Hong-Wei Ng and Stefan Winkler. A data-driven approach to cleaning large face datasets. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 343–347, Paris, France, October 2014. IEEE.

[35] Maria-Irina Nicolae, Mathieu Sinn, Minh Ngoc Tran, Beat Buesser, Ambrish Rawat, Martin Wistuba, Valentina Zantedeschi, Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, Ian M. Molloy, and Ben Edwards. Adversarial Robustness Toolbox v1.0.0, November 2019. arXiv:1807.01069 [cs, stat].

[36] Priyadarshini Panda, Indranil Chakraborty, and Kaushik Roy. Discretization Based Solutions for Secure Machine Learning Against Adversarial Attacks. *IEEE Access*, 7:70157–70168, 2019.

[37] Adnan Qayyum, Muhammad Usama, Junaid Qadir, and Ala Al-Fuqaha. Securing Connected & Autonomous Vehicles: Challenges Posed by Adversarial Machine Learning and The Way Forward. *IEEE Communications Surveys & Tutorials*, 22(2):998–1026, 2020. arXiv:1905.12762 [cs, stat].

[38] Murad Al Qurishee, Weidong Wu, Babatunde Atolagbe, Joseph Owino, Ignatius Fomunung, and Mbakisya Onyango. Creating a Dataset to Boost Civil Engineering Deep Learning Research and Application. *Engineering*, 12(03):151–165, 2020.

[39] Shreyas Ramakrishna, Baiting Luo, Christopher Kuhn, Gabor Karsai, and Abhishek Dubey. ANTI-CARLA: An Adversarial Testing Framework for Autonomous Vehicles in CARLA, July 2022. arXiv:2208.06309 [cs].

[40] David E. Rumelhart, Bernard Widrow, and Michael A. Lehr. The basic ideas in neural networks. *Communications of the ACM*, 37(3):87–92, March 1994.

[41] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge, January 2015. arXiv:1409.0575 [cs].

[42] Brian García Sarmina, Guo-Hua Sun, and Shi-Hai Dong. Principal Component Analysis and t-Distributed Stochastic Neighbor Embedding Analysis in the Study of Quantum Approximate Optimization Algorithm Entangled and Non-Entangled Mixing Operators. *Entropy*, 25(11):1499, October 2023.

[43] Ali Shafahi, Mahyar Najibi, Zheng Xu, John Dickerson, Larry S. Davis, and Tom Goldstein. Universal Adversarial Training. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):5636–5643, April 2020.

[44] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1528–1540, Vienna Austria, October 2016. ACM.

[45] Richard Sutton. Introduction: The Challenge of Reinforcement Learning. *Machine Learning*, 8:225–227, May 1992.

[46] Aleksei Triastcyn. Data-Aware Privacy-Preserving Machine Learning. *EPFL*, 2020.

[47] Jessica Vamathevan, Dominic Clark, Paul Czodrowski, Ian Dunham, Edgardo Ferran, George Lee, Bin Li, Anant Madabhushi, Parantu Shah, Michaela Spitzer, and Shanrong Zhao. Applications of machine learning in drug discovery and development. *Nature Reviews Drug Discovery*, 18(6):463–477, June 2019.

[48] Jingyuan Wang, Yufan Wu, Mingxuan Li, Xin Lin, Junjie Wu, and Chao Li. Interpretability is a Kind of Safety: An Interpreter-based Ensemble for Adversary Defense. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 15–24, August 2020. arXiv:2304.06919 [cs].

[49] Zhipeng Wei, Jingjing Chen, Xingxing Wei, Linxi Jiang, Tat-Seng Chua, Fengfeng Zhou, and Yu-Gang Jiang. Heuristic Black-Box Adversarial Attacks on Video Recognition Models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(07):12338–12345, April 2020.

[50] Han Xu, Yao Ma, Hao-Chen Liu, Debayan Deb, Hui Liu, Ji-Liang Tang, and Anil K. Jain. Adversarial Attacks and Defenses in Images, Graphs and Text: A Review. *International Journal of Automation and Computing*, 17(2):151–178, April 2020.

[51] Ahmet Yilmaz and Riccardo Poli. Successfully and efficiently training deep multi-layer perceptrons with logistic activation function simply requires initializing the weights with an appropriate negative mean. *Neural Networks*, 153:87–103, September 2022.

[52] Mary Frances Zeager, Aksheetha Sridhar, Nathan Fogal, Stephen Adams, Donald E. Brown, and Peter A. Beling. Adversarial learning in credit card fraud detection. In *2017 Systems and Information Engineering Design Symposium (SIEDS)*, pages 112–116, Charlottesville, VA, USA, April 2017. IEEE.

[53] Haichao Zhang and Jianyu Wang. Towards Adversarially Robust Object Detection. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 421–430, Seoul, Korea (South), October 2019. IEEE.

[54] Jie Zhang, Bo Li, Chen Chen, Lingjuan Lyu, Shuang Wu, Shouhong Ding, and Chao Wu. Delving into the Adversarial Robustness of Federated Learning, February 2023. arXiv:2302.09479 [cs].

[55] Qiuyu Zhu and Xuewen Zu. Fully Convolutional Neural Network Structure and Its Loss Function for Image Classification. *IEEE Access*, 10:35541–35549, 2022.

Appendix A
APPENDIX

A.1 Curse of dimensionality

As the number of dimensions being studied increases, many phenomena arise. The term for these phenomena is the "curse of dimensionality" coined by Richard Bellman [5]. In high dimensions, the volume of the space being studied grows exponentially. Assume there is a graduated line in one dimension from 1 to 10. There will be 10 integers on that line. Extending that to two dimensions, it now has 100 points with integer coordinates. Considering up to 80 dimensions reaches 10^{80}, a number of points equal to the estimate of the number of atoms in the observable universe. As the dimension increases, the exponential growth of the space causes data to become increasingly sparse.

The curse of dimensionality is closely related to the principle of overfitting. Because of the exponential growth of the volume of the space being studied, it is necessary to have large datasets to adequately capture and model high-dimensional patterns. The number of samples required to overcome this limitation grows exponentially. This scenario, characterized by its many features yet relatively few data points, is prone to overfitting.

Occam's Razor suggests that simpler models are generally better than complex ones because they are less likely to overfit. This principle is particularly relevant in high-dimensional contexts as it encourages the reduction of model complexity. Application of the Occam's Razor principle can mean applying dimensionality reduction to the problem through methods like PCA, t-SNE, feature selection, and others to mitigate the effects of the curse of dimensionality.

A.2 Datasets

A.2.1 MNIST

This dataset is constructed from the National Institute of Standards and Technology (NIST)'s Special Database 1 and 3 containing handwritten digit images ranging from 0 to 9. The creators judged that the suitability of the original datasets was insufficient due to the training data originating from American Census Bureau employees and the test data originating from American high school students. Further, the normalization process introduced grey-scale levels that had to be remedied. With two sets of approximately 30000 samples each, the researchers combined the datasets to create a full set with 60000 images for training and 10000 images used as the test set, 5000 belonging to Special Database 1, and 5000 belonging to Special Database 3.

The original images were normalized to fit within a 20x20 pixel box while preserving the aspect ratio. The images used in this dataset are centered in a 28x28 image by computing the center of mass of the pixels and translating the image to position the center of mass in the middle of the 28x28 field [26].

Figure A.1: Size-normalized examples from the MNIST dataset, taken from [26].

A.2.2 CIFAR-10

The CIFAR-10 dataset is a labeled subset of the "80 million tiny images" dataset and was collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. It contains 60000 32x32 color images corresponding to 10 classes with 10000 images per class. There are 50000 training images and 10000 test images. The classes are as follows: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck [38, 23].

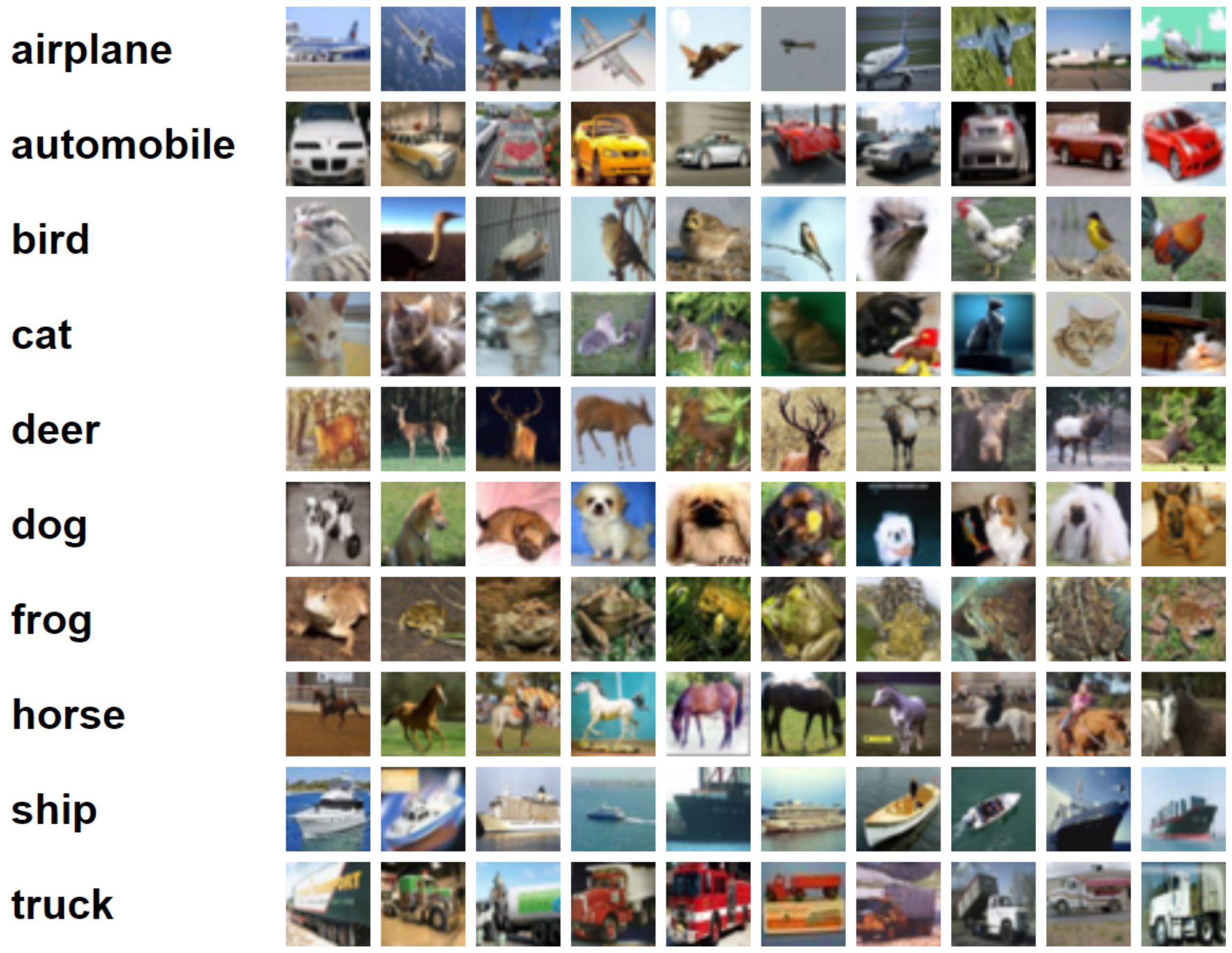


Figure A.2: Example images from the CIFAR-10 dataset, taken from [23].

A.2.3 ImageNet

ImageNet is a large visual database designed for object recognition research containing more than 15 million images. ImageNet is an image dataset organized by the WordNet hierarchy. Each concept in WordNet is described by possibly multiple phrases in WordNet called a synonym set. ImageNet aims to provide approximately 1000 images to illustrate each synonym set [12]. This dataset, while used for testing, is not entirely included with the framework. This is due to the 150 GB size of ImageNet being too large to justify bundling with the framework.

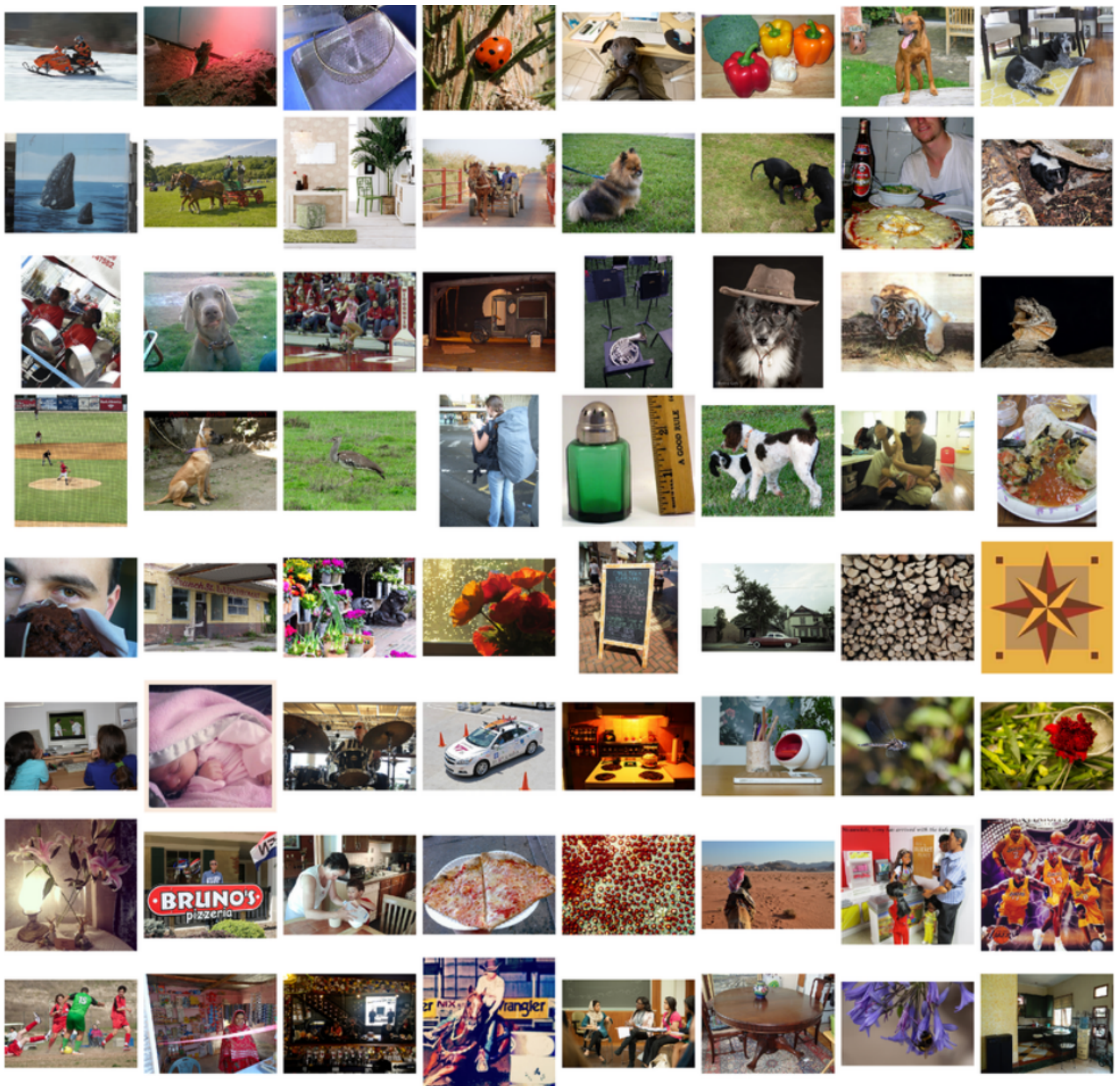


Figure A.3: Random selection of images from the ImageNet dataset [41].

A.3 Loss functions

A.3.1 Categorical cross entropy

The CategoricalCrossEntropy class computes the categorical cross entropy loss for multi-class classification tasks. The forward pass computes the categorical cross entropy loss for given predictions and true labels. It then adds a small epsilon value to avoid division by zero in the logarithmic terms. The total loss is computed as the sum of the negative log-likelihoods across all samples divided by the number of samples. The backwards method computes the gradients of the categorical cross entropy loss with respect to the input values in the final layer of the network. The gradients are divided by the number of samples to facilitate error back-propagation.

A.3.2 Binary cross entropy

Similarly, the BinaryCrossEntropy class computes the binary cross entropy loss for binary classification tasks. It computes the loss for given predictions and compares it to the true labels, then adds a small epsilon value to avoid division by zero in logarithmic terms. The mean loss across all samples is returned. In the backwards method, it computes the gradients of the binary cross entropy loss with respect to the input values in the final layer of the network, representing how much the predicted probabilities need to be adjusted to minimize the loss, and back-propagates them to update network weights.

A.3.3 Mean squared error

The MeanSquaredError class computes the mean squared error loss for regression tasks. The forward pass takes the mean of the squared differences between each predicted value and the true label across all samples in the batch and returns the average squared difference between predicted and true values across the batch. The backwards pass computes the gradients of the mean squared error loss with respect to the input values and back-propagates them. These gradients represent how much the predicted probabilities need to be adjusted to minimize the loss.