2023-03

# EXPLORING NEURAL NETWORK DEFENSES WITH ADVERSARIAL MIXUP

Andrianopoulos, Georgios

Monterey, CA; Naval Postgraduate School

https://hdl.handle.net/10945/71971

# NAVAL POSTGRADUATE SCHOOL

**MONTEREY, CALIFORNIA**

# THESIS

**EXPLORING NEURAL NETWORK DEFENSES WITH ADVERSARIAL MIXUP**

by

Georgios Andrianopoulos

March 2023

| | |
|---|---|
| Thesis Advisor: | Armon C. Barton |
| Second Reader: | Valdis A. Berzins |

**Approved for public release. Distribution is unlimited.**

THIS PAGE INTENTIONALLY LEFT BLANK

| | |
|---|---|
| **REPORT DOCUMENTATION PAGE** | *Form Approved OMB No. 0704-0188* |

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC, 20503.

| | | |
|---|---|---|
| **1. AGENCY USE ONLY**<br>***(Leave blank)*** | **2. REPORT DATE**<br>March 2023 | **3. REPORT TYPE AND DATES COVERED**<br>Master's thesis |

| | |
|---|---|
| **4. TITLE AND SUBTITLE**<br>EXPLORING NEURAL NETWORK DEFENSES WITH ADVERSARIAL MIXUP | **5. FUNDING NUMBERS** |
| **6. AUTHOR(S)** Georgios Andrianopoulos | |
| **7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**<br>Naval Postgraduate School<br>Monterey, CA 93943-5000 | **8. PERFORMING ORGANIZATION REPORT NUMBER** |
| **9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)**<br>N/A | **10. SPONSORING / MONITORING AGENCY REPORT NUMBER** |

**11. SUPPLEMENTARY NOTES** The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

| | |
|---|---|
| **12a. DISTRIBUTION / AVAILABILITY STATEMENT**<br>Approved for public release. Distribution is unlimited. | **12b. DISTRIBUTION CODE**<br>A |

**13. ABSTRACT (maximum 200 words)**

Neural networks (NNs) are vulnerable to adversarial examples, and extensive research is aimed at detecting them. However, detecting adversarial examples is not easy, even with the construction of new loss functions in a network. In this study, we introduce the Adversarial Mixup (AdvMix) network, a neural network that adds a None of the Above (NOTA) class on top of the existing classes to isolate the space where adversarial examples exist. We investigate the effectiveness of AdvMix in improving the robustness of models trained on deep neural networks against adversarial attacks by detecting them. We experimented with various data augmentation techniques and trained nine different models. Our findings show that using an AdvMix network can significantly improve the performance of models against various attacks while achieving better accuracy on benign examples. We were able to increase the accuracy of the vanilla model from 91% to 95% and improve the model's robustness. In many cases, we were able to eliminate the vulnerability of models against some popular and efficient attacks.

| | |
|---|---|
| **14. SUBJECT TERMS**<br>AdvMix, Adversarial Mixup, PadNet, machine learning, adversarial attacks, supervised classification, CNN, NN, computer vision | **15. NUMBER OF PAGES**<br>77 |
| | **16. PRICE CODE** |

| | | | |
|---|---|---|---|
| **17. SECURITY CLASSIFICATION OF REPORT**<br>Unclassified | **18. SECURITY CLASSIFICATION OF THIS PAGE**<br>Unclassified | **19. SECURITY CLASSIFICATION OF ABSTRACT**<br>Unclassified | **20. LIMITATION OF ABSTRACT**<br>UU |

THIS PAGE INTENTIONALLY LEFT BLANK

**Approved for public release. Distribution is unlimited.**

# EXPLORING NEURAL NETWORK DEFENSES WITH ADVERSARIAL MIXUP

Georgios Andrianopoulos
Captain, Hellenic Army
B, Hellenic Army Academy, 2011

Submitted in partial fulfillment of the
requirements for the degree of

**MASTER OF SCIENCE IN COMPUTER SCIENCE**

from the

**NAVAL POSTGRADUATE SCHOOL**
**March 2023**

Approved by: Armon C. Barton
Advisor

Valdis A. Berzins
Second Reader

Gurminder Singh
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

# ABSTRACT

Neural networks (NNs) are vulnerable to adversarial examples, and extensive research is aimed at detecting them. However, detecting adversarial examples is not easy, even with the construction of new loss functions in a network. In this study, we introduce the Adversarial Mixup (AdvMix) network, a neural network that adds a None of the Above (NOTA) class on top of the existing classes to isolate the space where adversarial examples exist. We investigate the effectiveness of AdvMix in improving the robustness of models trained on deep neural networks against adversarial attacks by detecting them. We experimented with various data augmentation techniques and trained nine different models. Our findings show that using an AdvMix network can significantly improve the performance of models against various attacks while achieving better accuracy on benign examples. We were able to increase the accuracy of the vanilla model from 91% to 95% and improve the model's robustness. In many cases, we were able to eliminate the vulnerability of models against some popular and efficient attacks.

THIS PAGE INTENTIONALLY LEFT BLANK

# Table of Contents

# List of Figures

# List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

# List of Acronyms and Abbreviations

| | |
|---|---|
| **AdvMix** | Adversarial Mixup |
| **AI** | Artificial Intelligence |
| **ASR** | Attack Success Rate |
| **ART** | Adversarial Robustness Toolbox |
| **CIFAR-10** | Canadian Institute For Advanced Research-10 |
| **CNN** | Convolutional Neural Network |
| **CV** | Computer Vision |
| **DL** | Deep Learning |
| **DNN** | Deep Neural Network |
| **EAD** | Elastic-net Attacks to DNNs |
| **FGSM** | Fast Gradient Sign Method |
| **GAN** | Generative Adversarial Networks |
| **GPU** | Graphic Processing Unit |
| **ILSVRC** | ImageNet Large Scale Visual Recognition Challenge |
| **JSMA** | Jacobian Saliency Map Attack |
| **k-NN** | k-Nearest Neighbors |
| **MAP** | Maximum a Posteriori |
| **MLE** | Maximum Likelihood Estimation |
| **ML** | Machine Learning |

| | |
|---|---|
| **NN** | Neural Network |
| **NP** | Non-deterministic Polynomial time |
| **NOTA** | None Of The Above |
| **PGD** | Projected Gradient Descent |
| **ReLu** | Rectified Linear Unit |
| **ResNet** | Residual Network |
| **SGD** | Stochastic Gradient Descent |
| **SVM** | Support Vector Machine |
| **TLU** | Threshold Logic Unit |
| **TRADES** | Trade-off-inspired Defense to Adversarial Attacks |
| **WRN** | Wide Residual Network |

# Acknowledgments

I want to thank my wife for her constant support and encouragement. She has been patient and understanding throughout this process, and I am grateful for her love. I also want to thank my daughter, whose infectious smile and laughter take away any tiredness I feel and bring joy to my life. I owe a debt of gratitude to my parents, who have always been there for me and supported me in all my decisions, even the wrong ones. Their guidance and belief in me have been instrumental in making this achievement possible, and I am blessed to have such supportive parents. Without their guidance and support, I could not have done this. I feel fortunate to have such a wonderful family, and I dedicate this thesis to them.

THIS PAGE INTENTIONALLY LEFT BLANK

# CHAPTER 1:
# Introduction

Alan M. Turing was one of the pioneers who laid the foundations of Artificial Intelligence (AI). His contributions marked the beginning of humanity's exploration and definition of the various subdomains of AI. That is considered as the spring of AI [1]. The landmark today has totally changed as Machine Learning (ML), Neural Networks (NNs), and Deep Learning (DL) have become different fields of research. Figure 1.1 shows how these four domains are related. The following are focused on ML, NN and mostly in DL.

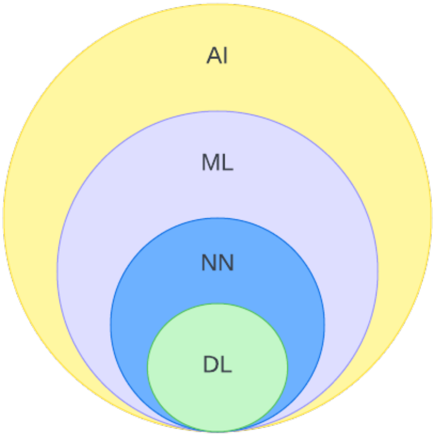


Figure 1.1. AI's domain Venn diagram. Adapted from [2].

## 1.1 Machine Learning

ML is used when we want to build a model from given data without the need for explicit programming of a machine [3]. The two broad main categories with their subcategories of ML are:

- Supervised learning
    - Classification
    - Regression
- Unsupervised learning
    - Clustering
    - Association

The scope of this research focuses on supervised classification problems, specifically in the field of Computer Vision (CV), where trained models try to correctly predict the class of an image. For general classification, many algorithms can be used to train a model to make the correct predictions. The most common algorithms according to [3] are:

- k-Nearest Neighbors (k-NN)
- Logistic Regression
- Linear Regression
- Support Vector Machine (SVM)
- Decision Trees and Random Forests
- NN

Each algorithm has its advantages and disadvantages and is preferred depending on the model's purpose and the available data for training. The use of Graphic Processing Units (GPUs) and their acceleration helped the development of NN architectures that dominate the field of classification tasks. This fact drives the wide use of NN algorithms and the development of Deep Neural Networks (DNNs) and Convolutional Neural Networks (CNNs) algorithms for which there is a brief analysis in Chapter 2. There is a significant disadvantage to those architectures. We do not fully understand their capabilities and how they are calculating the boundaries of each class. That drives the fact that DNNs are vulnerable to adversarial attacks. An adversarial attack is an algorithm that attempts to add minimal perturbation to the inputs of a model to make the model misclassify that input. Section 2.3 provides examples of state-of-the-art attacks that attackers usually use for attacking CNN models.

Evaluating the robustness of a model is not a trivial task. There is a lot of research that tries to establish the principles of the robustness of a model against adversarial examples, but by the time we establish the principles, they are outdated because the adversary has produced new attacks that explore the dark spots of a model.

Each ML algorithm listed above has its own strengths and weaknesses, and the choice of which to use depends on the specific goals of the model and the data available for training. For example, some algorithms may excel at processing large amounts of data quickly, while others may be better suited to complex, non-linear relationships between inputs and outputs [4]. In recent years, the use of GPUs and other acceleration technologies have greatly facilitated the development of NN architectures that are well-suited to a wide range of classification tasks. These NN algorithms have become increasingly popular due to their ability to learn and make predictions based on patterns in the training data [5]. The two main types of NN algorithms that have emerged in this field are DNNs and CNNs. These algorithms are analyzed in more detail in Chapter 2.

## 1.2 Security and Reliability

Despite their many strengths, these NN algorithms have a significant drawback: they can be vulnerable to adversarial attacks. An adversarial attack is an algorithm that generates inputs for the model that has a minimal perturbation, but that can cause the model to make an incorrect prediction [6]. These attacks can be especially problematic for DNNs, which may not fully understand the boundaries between different classes in the data [7] and have serious consequences such as security breaches, incorrect diagnoses, or financial losses [8]. To mitigate these risks, researchers are constantly exploring new methods for evaluating the robustness of models against adversarial examples, as described in 2.3. Adversarial attacks can be used to evaluate the robustness of a model in several ways. For example, researchers may generate adversarial examples and test the model's ability to correctly classify them [9]. By measuring the rate of misclassification, researchers can gain insights into the model's resilience against these attacks, and identify areas for improvement [10]. Additionally, researchers may evaluate the model's robustness by assessing the magnitude of the perturbations required to cause a misclassification [11]. By analyzing these results, researchers can gain a deeper understanding of the model's strengths and weaknesses, and make informed decisions about how to improve its robustness [12].

To address the aforementioned challenge, this thesis proposes a new approach for a defense against adversarial examples using Adversarial Mixup (AdvMix) which was inspired by PadNet [13], a novel algorithm that detects adversarial examples by creating a new class. Regardless of how the adversarial examples were generated, AdvMix can detect them. AdvMix works by adding a None Of The Above (NOTA) class that works as a detection mechanism into the neural network architecture. This mechanism is designed to identify the presence of adversarial examples. Chapter 3 introduce AdvMix algorithm and how it works in more detail. The algorithm is evaluated using a brute-force approach, which involves testing the algorithm against a wide range of state-of-the-art attacks. The results of the evaluation show that AdvMix is effective in detecting adversarial examples and provides a promising solution for improving the robustness of deep learning models.

## 1.3 Thesis Organization

The thesis consists of six chapters including the current one:
Chapter 1 provides an overview and a brief introduction of this research domain and how it is related to ML. The overview provides readers with a clear idea of the context and relevance of the research domain, while the introduction highlights the critical role of ML in advancing the field. Chapter 2 provides background information on DNNs and more advanced architectures that we use in our research. It also includes state-of-the-art attacks and defenses for NNs and adversarial examples. Chapter 3 is about the methodology used in the research, including the use of the Canadian Institute For Advanced Research-10 (CIFAR-10) dataset, the process for generating adversarial examples, the Adversarial Mixup defense, the training procedure, and the development of a brute force attack method. Chapter 4 describes the test design and implementation, including the experimental design and validation of Adversarial Mixup. Chapter 5 discusses the results and future work, including potential areas for further research and development. Chapter 6 concludes the study by summarizing the key findings.

# CHAPTER 2: Background and Related Work

This chapter focuses on one of the most important domains of AI, which is ML, and its recent advances and limitations. In particular, the combination of NN and DL with hardware acceleration has led to impressive progress in image classification prediction models, among others. However, this progression also drives the development of adversarial attacks aimed at causing these models to produce incorrect predictions. The subsequent sections will delve into these areas of ML and the state-of-the-art defenses in more detail.

## 2.1 NNs, DL, and CNNs

NNs, DL, and CNNs are very often confused and believed to be synonymous with one another. The reason is that DL and CNNs are different algorithms that are based on NNs. The following subsections deconstruct those algorithms, giving a better understanding of each and the relationship between them.

### NNs

The concept of artificial neural networks (NNs) can be traced back to 1943 when Warren McCulloch and Walter Pitts first presented the idea in their paper "A Logical Calculus of Ideas Immanent in Nervous Activity." Inspired by the workings of biological neurons, they constructed a simple model, later referred to as an artificial neuron, with a binary output [3]. By employing a step function at the end of the neuron, linear models capable of performing classification could be generated. The simplest of these networks is the Perceptron, which uses a single artificial neuron, as depicted in Figure 2.1. In place of the Threshold Logic Unit (TLU), various activation functions may be utilized depending on the requirements of the model. Building upon this simple network structure, more complex architectures can be created by interconnecting nodes. The primary advantage of NNs lies in their ability to learn patterns from input data through adjustments of the input weights of each neuron [4]. NNs containing more than one hidden layer are considered to be DNNs.

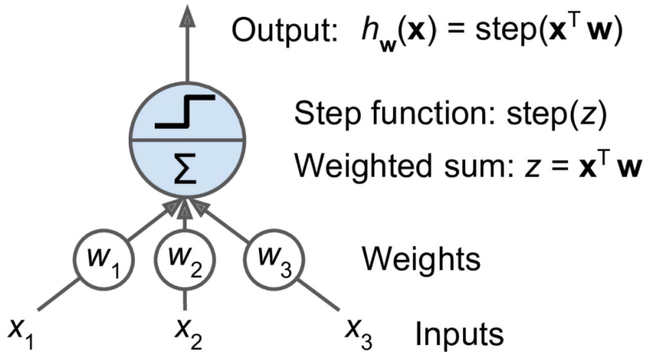

Output: $h_{\mathbf{w}}(\mathbf{x}) = \text{step}(\mathbf{x}^T \mathbf{w})$
Step function: step($z$)
Weighted sum: $z = \mathbf{x}^T \mathbf{w}$
$\Sigma$
$w_1$
$w_2$
$w_3$
Weights
$x_1$
$x_2$
$x_3$
Inputs

Figure 2.1. Perceptron network with TLU. Source: [3].

### DL and DNNs

DL algorithms have gained widespread use in the development of DNNs for image classification tasks. The success of DL can be attributed to the use of back-propagation and optimization algorithms, which enable the training of models to discover complex data structures and relationships among them [5]. The rise in computational power and the availability of large data sets have made DL a leading approach in the classification domain, leading to the creation of a variety of algorithms by both industry and academic research groups. It is noteworthy that one of the earliest DNN models to win the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012 was AlexNet [14], which outperformed all other models and demonstrated the potential of DL with its eight layers. Recent models that consist of many layers, such as Residual Network (ResNet)-152 that consists of 152 layers, won the ILSVRC 2015 contest [15]. Most of the DNNs use some convolutional layers to apply different filters and extract important features from the input data.

### CNNs

The introduction of CNNs has resulted in improved performance for image classification tasks, making them the dominant architecture in the field [16]. A key aspect of their success is their ability to learn hierarchical representations, where lower-level features are combined to form higher-level features [17]. This allows them to effectively capture both local and global information from an image, making them well-suited for image recognition tasks.

Each convolutional layer in a CNN applies filters that the model trains in order to find important features in an image [18]. The filters are convolved with the input image to generate feature maps, which are then passed through activation functions to create nonlinear algorithms that fit better to the training data [19]. The activation functions, such as the Rectified Linear Unit (ReLu) (equation 2.1 and Figure 2.3), introduce non-linearity into the model, allowing it to learn complex, non-linear relationships in the data [20]. In addition to the convolutional layers, CNNs usually consist of pooling layers, which are used for keeping only the important features and for dimensionality reduction of the input image [21]. Pooling layers perform down-sampling operations, such as max-pooling, to reduce the spatial size of the feature maps. This reduction helps the network to be invariant to small translations and distortions in the input image. Finally, at least one fully connected layer (a classic NN) is used in a CNN to make the final prediction. The fully connected layer uses the feature maps generated by the convolutional and pooling layers as input, and outputs a probability distribution over the class labels [4].

Overall, the combination of convolutional, pooling, and fully connected layers in a CNN allows it to effectively learn hierarchical representations and make accurate predictions for image recognition tasks. The algorithm that is implemented to train various models in this study is a special ResNet algorithm called Wide Residual Network (WRN) [22].

## 2.2 Wide ResNet

ResNet is a DNN that was introduced by Microsoft researchers in 2015 [21]. It is considered to be one of the most influential models in the field of CV. The key idea behind ResNet is to make use of residual connections, which allow the gradient to bypass one or more layers, helping to prevent the vanishing gradient problem that can arise in deep networks [21]. This architecture allows ResNet to be trained effectively, even with very deep architectures. Each ResNet model consists of residual blocks (Figure 2.2), where the inputs of each block are combined with its output through a shortcut connection [21]. This residual connection allows the network to remember the features learned in previous blocks and use them in later blocks, making the network more effective [21]. The ReLu activation function (equation 2.1) is used after each residual block, where the function removes all the features that are not important when applying the convolutional filters because of the function's nature as depicted in Figure 2.3 [19]. ResNet retains these features from the previous layer instead of removing them, and as explained below the retain provides additional information that does not add complexity to the algorithm or increase the number of parameters [21].

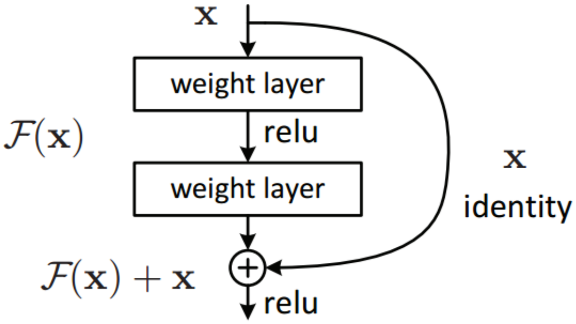

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2.2. ResNet building block. Source: [21].

The success of ResNet can be attributed to its ability to prevent the vanishing gradient problem, which allows for deep architectures to be trained effectively [21]. This is possible because the residual connections allow the gradient to flow directly from the output to the input of each block, bypassing the layers in between [21]. As a result, the gradients in deeper layers are easier to calculate, allowing for better optimization of the network parameters by using the same optimization functions such as Stochastic Gradient Descent (SGD) [21].

$$f(x) = max(0, x) = \begin{cases} x & if \quad x \geq 0 \\ 0 & otherwise \end{cases} \tag{2.1}$$

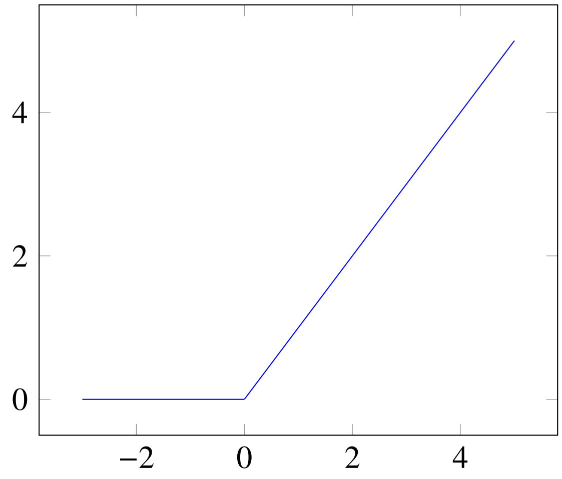


Figure 2.3. ReLu Function.

In the paper "Wide Residual Networks", Mark D. McDonnell proposes the WRN as an improvement to ResNet. The WRN model architecture includes several modifications that aim to improve its performance. Firstly, the model introduces a batch normalization layer immediately after the input layer, which has learnable parameters that can be updated during the training process. This additional layer at the very beginning of the model helps to normalize the inputs and stabilize the training process [22]. The second modification involves the use of average pooling to downsample the residual pathways, which helps to reduce the computational complexity of the network while maintaining the spatial information of the features [22]. The third modification seeks to simplify the model by making the number of channels in the first convolutional layer equal to that of the first residual block. This modification simplifies the structure of the model but increases the number of parameters that need to be learned during the training process. Compared to the total parameters of the model, this addition is small and it is not considered a problem [22]. The WRN employs a one-bit representation for each weight, which greatly reduces the memory requirements and computational cost of the network. This makes the WRN model computationally more efficient than traditional residual networks. The improvements in the architecture and efficiency of the WRN model, as proposed by McDonnell, make it a promising solution for a wide range of computer vision tasks.

## 2.3 Adversarial Attacks

There are many different kinds of attacks that can be performed on an ML model, each with its own purpose. Table 2.1 gives a summary of known attacks that are separated into three main categories. This thesis focuses on evasion attacks, which, along with poisoning attacks, are the most common in CNN architectures.

Table 2.1. Attack summary. Source [23].

| | |
|---|---|
| **Exploratory Attacks** | Model Inversion<br>Membership Inference attack<br>Model Extraction via APIs<br>Information Inference |
| **Evasion Attacks** | Adversarial Examples Generation<br>Generative Adversarial Networks (GAN)<br>GAN-based attack in collaborative learning<br>Intrusion Detection Systems<br>Adversarial Classification |
| **Poisoning Attacks** | Support Vector Machine Poisoning<br>Poisoning on collaborative filtering systems<br>Anomaly detection systems |

### 2.3.1 Evasion Attacks

Evasion attacks are a type of attack in which the attacker tries to make a model misclassify an input. A data set is used during the training phase of a model. In CV, these data sets consist of images. Adversarial examples are "inputs formed by applying small but intentionally worst-case perturbations to examples from the dataset, such that the perturbed input results in the model outputting an incorrect answer with high confidence" [7]. CNNs are known to be particularly vulnerable to adversarial examples. This vulnerability can cause unpredictable performance and hinder the use of CNNs in security-critical domains [10]. One of the main reasons for this vulnerability is the linear nature of the models, even though they are not necessarily linear models. This linearity arises from the fact that most optimizing functions, such as ReLu (eq. 2.1), are linear-based [7]. Adversarial examples are calculated using different approaches, which allows for the creation of small, targeted perturbations to the original examples that result in the model making incorrect predictions.

### White and Black Box Attacks

This thesis focuses on two main categories of attacks on ML models: white-box and black-box. Table 2.2 depicts briefly the main differences between the two attacks. In a white-box attack, the attacker has complete access and knowledge of the model used for predictions. This includes the algorithm, architecture, and parameters ($\theta$) of the fully trained model. With this information, the attacker can identify potential vulnerabilities and create highly targeted perturbations to manipulate the model's output. This type of attack is the most effective, as the attacker has a complete understanding of the model's inner workings [23].

On the other hand, a black-box attack occurs when the attacker has no knowledge of the model's architecture, algorithm, or parameters. This makes the attack more challenging, as the attacker must rely on the model's outputs to identify vulnerabilities and create perturbations, based on previous predictions. There are three subtypes of black-box attacks: Non-Adaptive, Adaptive, and strict. In a Non-Adaptive black-box attack, the attacker cannot modify the inputs based on the model's output, making it the most difficult type of black-box attack. In an Adaptive black-box attack, the attacker can modify the inputs based on the model's output, making it a more effective attack. In a strict black-box attack, the attacker has limited information about the model's outputs, making it a less effective attack compared to the other two subtypes [23].

Table 2.2. Distinction between black box and white box attacks. Source: [23]

| Description | Black box attack | White box attack |
|---|---|---|
| **Adversary Knowledge** | Restricted knowledge from being able to only observe the network's output on some probed inputs. | Detailed knowledge of the network architecture and the parameters resulting from training. |
| **Attack Strategy** | Based on a greedy local search generating an implicit approximation to the actual gradient w.r.t the current output by observing changes in input. | Based on the gradient of the network loss function w.r.t to the input. |

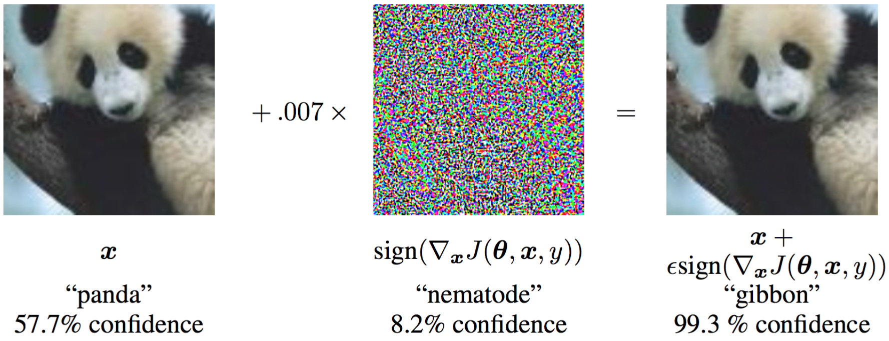
Using either of those two kinds of attacks, there are many different ways to compute efficiently the perturbation for an input. The following sections are about the state-of-the-art attacks that are used.

**Fast Gradient Sign Method**

The Fast Gradient Sign Method (FGSM) attack is a popular and effective method for generating adversarial examples, and it has been an inspiration for many other attack methods [7], [24]. The attack works by adding a small perturbation to the input $X$ that is proportional to the sign of the gradient of the loss function with respect to the input. The adversarial example can be computed by the following equation:

$$X^{adv} = X + \epsilon \cdot \text{sign}(\nabla_x \mathcal{L}(\theta, x, y)) \tag{2.2}$$

where $\epsilon$ is a small constant and $J(\theta, x, y)$ is the cost function for the model with parameters $\theta$ on input $x$ and target label $y$. One of its biggest advantages is that it is really "fast" because it requires a single gradient computation. However, it assumes full knowledge of the model's parameters to be highly successful, making it a more effective white-box attack. Figure 2.4 demonstrates how adding a perturbation to an image from the ImageNet dataset produced with the FGSM attack causes a GoogleNet model to misclassify it [7].



$+ .007 \times$
$=$
$\boldsymbol{x}$
“panda”
57.7% confidence
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
“nematode”
8.2% confidence
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
“gibbon”
99.3 % confidence

Figure 2.4. FGSM attack to GoogleNet on ImageNet. source: [7].

FGSM remains a popular and widely used attack in many studies [7], [24], and also in this one, FGSM is used as a baseline for comparison.

### Projected Gradient Descent

Projected Gradient Descent (PGD) attacks are considered one of the strongest and most effective methods for evaluating the robustness of ML models, as they are white-box attacks that use the full knowledge of the model and its parameters [9]. The goal is to find the smallest perturbation that can cause the model to misclassify an input example. It calculates the gradient of the loss function with respect to the input and takes a step in the direction that maximizes the loss [7]. This process is repeated until the model misclassifies the input or a maximum number of iterations is reached. The perturbation can be calculated by using the equation 2.3 [11].

$$x_{t+1} = \Pi_{x \in [x_{min}, x_{max}]}(x_t + \alpha \cdot sign(\nabla_{x_t} \mathcal{L}(\theta, x_t, y))) \tag{2.3}$$

where:

$x_{t+1}$ is the updated input at iteration t+1, $\Pi_{x \in [x_{min}, x_{max}]}$ is the projection operator that projects $x_{t+1}$ back into the valid space $[x_{min}, x_{max}]$, $x_t$ is the input at iteration t, $\alpha$ is the step size, $sign(\nabla_{x_t} J(\theta, x_t, y))$ is the sign of the gradient of the cost function $J(\theta, x_t, y)$ w.r.t. $x_t$, computed for the current model parameters $\theta$ and target label $y$.

The number of iterations is an important parameter in the PGD attack, as it affects the strength of the adversarial example generated. A larger number of iterations means a stronger adversarial example, as more steps are taken in the direction that maximizes the loss. However, a larger number of iterations also increases the computational cost of the attack [11].

### Auto Projected Gradient Descent

This is a variant of the PGD attack [25]. The difference from the PGD attack is the use of an optimization process to find the optimal parameters that are used in PGD. For example, the maximum iterations and the step size. Auto-PGD generates examples that are more efficient and more robust compared to PGD and can evaluate the robustness of a model with high accuracy.

### Elastic-net

The Elastic-net Attacks to DNNs (EAD) is another method for generating adversarial examples, which operates similarly to the PGD attack [26]. The main difference is the regularization term used in the loss function. EAD uses both $L_1$ and $L_2$ regularization terms and incorporates a surrogate function, which enables sparse perturbations, meaning it only changes a small number of input features. When the model misclassifies the input, the algorithm stops updating and calculating the noise. The EAD method appears to outperform many other attacks, including the PGD, and is considered a highly efficient attack that can be used in both black and white box attacks.

### DeepFool

The DeepFool attack is another method for generating adversarial examples that calculates the minimum perturbation by using many iterations to push the input example into the decision boundaries of another class [27]. It is an attack that uses the gradient of the model to determine the direction and magnitude of the perturbation. The algorithm works by first finding the minimal distance from the input to the decision boundary of each class, then takes a step in the direction of the closest decision boundary. This step is repeated until the classifier misclassifies the input label. The algorithm also uses a property of maximum iterations to stop adding the computed perturbation to the input. Unlike the PGD attack, DeepFool generates a small perturbation that lies close to the decision boundary, which makes it less perceptible to humans and difficult to visually detect. DeepFool is considered one of the most effective adversarial attacks, with high success rates against various types of models, including deep neural networks that seem to outperform other methods. One disadvantage is that it can be computationally expensive, especially for large datasets.

### Carlini and Wagner attack

The Carlini and Wagner attack is a family of attacks [10], [28]. Similar to other attacks, it tries to add a small perturbation to an input $x$ to create an adversarial example $x'$. It is considered a family of attacks because the distance between the input and the adversarial example can be measured in $L_0$, $L_2$, or $L_\infty$ norm. $L_\infty$ is considered the most efficient and effective method, and it is more difficult to detect than those generated by other attacks. The attack is formed as an optimization problem that tries to minimize the perturbation size and the aforementioned distance. This research deals with $L_2$ and $L_\infty$ norms. The perturbation calculation function for the $L_2$-norm attack is calculated by:

$$\underset{\delta}{\text{minimize}} \quad c \cdot f(x+\delta) + D(\delta) \quad \text{such that} \quad x+\delta \in [0,1]^n \tag{2.4}$$

where $\delta$ is the perturbation, $c$ is a constant that controls the trade-off between the size of the perturbation and the confidence of the adversarial example, also called confidence level, $f$ is the targeted misclassification loss, and $x$ is the original input. For $L_\infty$-norm attacks, the perturbation calculation function is:

$$\underset{\delta}{\text{minimize}} \quad |\delta|_p + c \cdot f(x+\delta) \quad \text{such that} \quad x+\delta \in [0,1]^n \tag{2.5}$$

where $\| \delta \|_\infty$ is the maximum absolute value of the elements of $\delta$.

These attacks are very strong, very difficult to detect, and outperform most of the other attacks, but they are computationally expensive, especially when the maximum number of iterations is big.

### Square attack

The square attack is considered one of the state-of-the-art black box attacks [29]. It is query-efficient, which means that it requires fewer evaluations of the targeted model compared to other black box attacks. The attack works by iteratively generating adversarial perturbations and querying the model to evaluate their effectiveness. One of the strong points of this algorithm is that it uses random samples for the evaluation, which makes it more efficient and less computationally expensive. Even though it is considered a black-box attack, it outperforms white-box methods as well.

### Boundary attack

The decision-based boundary attack is a powerful black-box attack that generates adversarial examples by querying the target model and estimating the gradients of the decision boundary, as described in the paper [30]. Unlike some other attacks, this approach does not require the attacker to calculate the gradients of the model, making it computationally efficient. The attack involves sampling and adding perturbations from a uniform distribution to the original input, and then evaluating the model to try to change the classification decision boundary in a targeted direction. By repeatedly refining the perturbations and estimating the gradients of the decision boundary, the attack can converge to an adversarial example that is difficult for the model to correctly classify. The decision-based boundary attack has been shown to be reliable and transferable, and has been used in various applications to evaluate and improve the robustness of machine learning models against adversarial attacks.

### Jacobian-based Saliency Map Attack

The Jacobian Saliency Map Attack (JSMA) is a white-box attack that aims to generate adversarial examples by modifying only a small subset of features of the input [31]. The attack uses the Jacobian matrix of the model's output with respect to the input to compute a saliency map that shows the features of the input that are most important in the model's decision. Then, the attack perturbs the input by changing the most important features in a way that maximizes the loss function and at the same time tries to keep the perturbation small. One of its main disadvantages is that it can be computationally expensive, especially for high-dimensional inputs and big datasets. Moreover, JSMA is a type of untargeted attack, which means that it aims to cause misclassification without a specific target label in mind. This limitation makes it less effective than targeted attacks, which can be used to steer the model's prediction toward a specific incorrect class. The architecture of the attack implies that the algorithm is effectively compared to the vulnerabilities of the targeted model. The more important features a model creates, the less effective the attack will be, and it will be easily detected.

## 2.4 Adversarial Defenses

Ever since the vulnerability of DNN models to adversarial attacks was discovered, researchers have been striving to develop and introduce defenses to either produce more robust models or to detect and discard adversarial examples and keep the network isolated. Another defense also can be produced when those different defenses are combined [13].

Another distinction between defense strategies is whether they are *proactive* or *reactive*. Proactive defense involves building more robust models before an attacker can produce an adversarial example, meaning that all defenses are implemented during the training process of a network. On the other hand, reactive defense involves building defenses after the model has been trained [32]. AdvMix is considered to be a proactive defense. Other proactive defenses that served as inspiration for this thesis are discussed in the following subsection.

#### Adversarial Training

Adversarial training is widely used and one of the best defense methods. The principle behind this defense is to train a DNN not only on a benign dataset, but also on adversarial examples generated from the training dataset. Adversarial training has been shown to increase a model's robustness to adversarial attacks, but only to the attack used for data augmentation [7]. However, this creates a challenge in that the dataset must be augmented for almost all known attacks, leading to increased computational inefficiency and potential overfitting, as the model may become too specialized in recognizing adversarial examples [33].

#### Data Augmentation

The idea behind using data augmentation as a defense method against adversarial attacks is to introduce small variations in the training data so that the model becomes robust to similar variations in the input. This is achieved by applying data augmentation techniques such as rotations, translations, scaling, flipping, and so on. By exposing the model to a wider range of variations in the input data, it becomes less vulnerable to adversarial examples, as the model is more robust to small perturbations in the input, reducing the likelihood of misclassification. This has been shown to be effective in improving the robustness of deep neural networks against adversarial examples [13].

Another technique, called Adversarial MixUp, combines two or more examples in a dataset to create a new training example. This method trains the network to make predictions based on the combination of training examples, making it more robust against small adversarial perturbations [34].

### TRADES

Trade-off-inspired Defense to Adversarial Attacks (TRADES) is designed to trade-off between accuracy on benign examples and robustness against adversarial examples. The main idea behind TRADES is to use a regularization term in the loss function that encourages the network to have large decision margins, meaning that the confidence in the network's predictions should be high. By doing this, the model becomes more robust to adversarial examples, as they would require larger perturbations to the input to change the network's predictions, because the decision boundaries of the classifier are far away from the input data [35].

### Gradient Regularization

The main concept of the gradient regularization defense is to regularize the input gradients of the network. By making them less noisy, the result is a less sensitive network, and so a more robust network to adversarial perturbation. The regularization is achieved by adding a term to the loss function that penalizes the magnitude of the gradients [36]. In other words, it helps to prevent the network from making large changes to the input during the forward pass. This results in a model that is less susceptible to adversarial examples, as the perturbations required to cause a misclassification are larger [36].

# CHAPTER 3: Methodology

This chapter provides an overview of the design and setup of a AdvMix algorithm. In this study, AdvMix uses a WRN as a base model, with an added component that is capable of detecting adversarial examples. To train an AdvMix model, we augment the training set with samples that are a mixture of benign samples with their adversarial example counterparts. The new AdvMix samples are labeled as the None of the Above (NOTA) class. We find that AdvMix can improve both benign accuracy due to better generalization, and robustness due to adding more separation between decision boundaries. The creation of adversarial examples is achieved through a range of methods, each with its own advantages and disadvantages, which are provided in subsection 3.2.3. Finally, the evaluation process and the metrics used to evaluate the performance of AdvMix are listed and analyzed.

Section 2.4 is about the defenses against adversarial attacks, which covers the different developed methods to increase the robustness of a model. AdvMix is under the domain of an adversarial detector that will help us to train more robust models. Before examining AdvMix's design, it is best first to have a closer look at the benign dataset used in the test case.

## 3.1 Benign Dataset: CIFAR-10

AdvMix can use any dataset as input, but for our test case, we have chosen CIFAR-10 for several reasons. CIFAR-10 is a well-known and widely used dataset in computer vision and adversarial attacking methods, making it an ideal choice for comparison with other methods. The dataset consists of 50,000 training images and 10,000 testing images, each with a size of 32x32x3 pixels as it is depicted in Table 3.1 [37]. It includes 10 classes, with 6,000 images per class, making it one of the best options for image classification tasks. The relatively small size of each image in CIFAR-10 makes it computationally efficient and feasible for experimental purposes. Furthermore, the dataset has been widely studied and has a large body of research available, providing a good benchmark for comparing the performance of various algorithms.

Table 3.1. CIFAR-10 dataset

| Name of class | Number of images |
|---|---|
| airplane | 6,000 |
| automobile | 6,000 |
| bird | 6,000 |
| cat | 6,000 |
| deer | 6,000 |
| dog | 6,000 |
| frog | 6,000 |
| horse | 6,000 |
| ship | 6,000 |
| truck | 6,000 |
| Training images | 50000 |
| Testing images | 10000 |

## 3.2 Adversarial Mixup Design and Parametrization

The inspiration is to create a model and use it as a detector for any potential attack. Figure 3.1 visualizes what we aim to achieve after the end of training, by comparing the borders for each class of a model trained using a traditional DNN algorithm and the AdvMix algorithm.

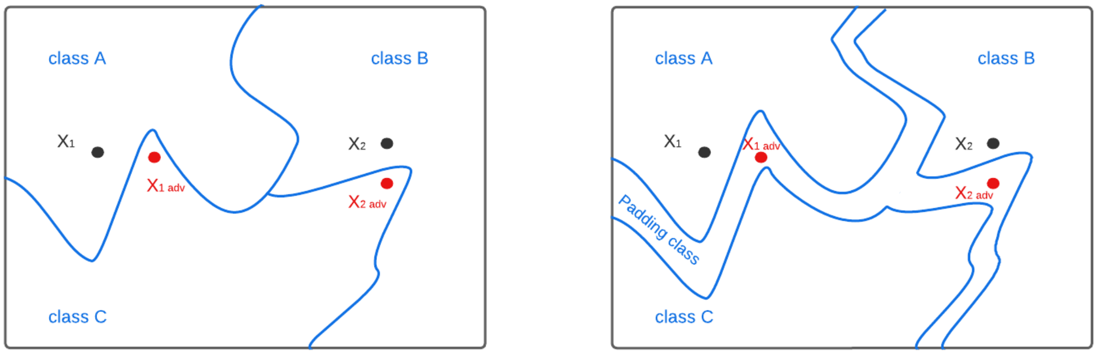


Figure 3.1. On left are the hypothetical borders of the base model. On the right, the hypothetical borders as AdvMix transform them. Adapted from [13].

One of the biggest advantages of this defense is that we do not need to know the method by which the attacker is going to generate adversarial examples to train the model based on this. Additionally, the data augmentation used is the same for each pipeline regardless of the attacking method [13].

### 3.2.1 The Base Model

The first step is to determine the base model and architecture we want to train. In the test case, we used a vanilla WRN that was analyzed in 2.2, which is known to be a highly efficient model with high accuracy in most cases. The width of the model is six and the depth is 12.

As aforementioned, instead of training a model with the 10 classes that CIFAR-10 has, we train a model with 11 classes. The last class will be the None of the Above NOTA class. The purpose is to detect adversarial examples as the adversarial class when the trained model accepts them as input. The problem is that the ground truth of the dataset only contains 10 classes.

To estimate the most probable values of model parameters we use the Maximum a Posteriori

(MAP) method. That is equivalent to adding a regularization term to a Maximum Likelihood Estimation (MLE). The loss function that we use is:

$$\mathcal{L}(\boldsymbol{\theta}) = -\log(P(\boldsymbol{\theta}|\mathcal{D})) \tag{3.1}$$

where $\boldsymbol{\theta}$ represents the parameters of the model, and $\mathcal{D}$ represents the training data. The posterior probability distribution $P(\boldsymbol{\theta}|\mathcal{D})$ is computed using Bayes' theorem:

$$P(\boldsymbol{\theta}|\mathcal{D}) = \frac{P(\mathcal{D}|\boldsymbol{\theta})P(\boldsymbol{\theta})}{P(\mathcal{D})} \tag{3.2}$$

where $P(\mathcal{D}|\boldsymbol{\theta})$ is the likelihood of the training data given the parameters, $P(\boldsymbol{\theta})$ is the prior probability distribution on the parameters, and $P(\mathcal{D})$ is the marginal likelihood of the training data. As mentioned before, in our case we have eleven classes. So the loss function can be written as follows:

$$\mathcal{L}(\boldsymbol{\theta}) = -\sum_{i=1}^{n} \log P(y_i|\mathbf{x}_i, \boldsymbol{\theta}) + L_2 \sum_{j=1}^{k} \theta_j^2 \tag{3.3}$$

where $\boldsymbol{\theta} = [\theta_1, \theta_2, \ldots, \theta_k]$ are the model parameters, $n$ is the number of training examples, $k$ is the number of parameters, $\mathbf{x}_i$ is the feature vector for the $i$-th example, $y_i$ is the corresponding label, and $\lambda$ is the regularization parameter.

In our implementation, we use the TensorFlow library. To use the MAP estimation of the model we have to regularize the kernel. We use a $L_2$ regularization coefficient of 0.00025. The update loss function is:

$$\mathcal{L}(\boldsymbol{\theta}) = -\sum_{i=1}^{n} \log P(y_i|\mathbf{x}_i, \boldsymbol{\theta}) + 0.00025 \sum_{j=1}^{k} \theta_j^2 \tag{3.4}$$

### 3.2.2 Creating the NOTA Class

We use a data augmentation technique called *adversarial mixup* to create the new class [13]. First, an adversarial example $x'$ was produced from a benign example $x$ by applying an attack method that we want.

$$x' = adversarial\_attack(x) \tag{3.5}$$

We introduce two different types of adversarial mixup samples:

- *mean Advmix*, where the new sample is given by getting the mean of the benign and the adversarial example and adding some noise [13].

  $$x_{mean\_AdvMix} = \frac{(x' + x)}{2} + n \tag{3.6}$$

  where $n \sim \mathcal{N}(0, 0.1)$

- *uniform adversarial mixup* where the new sample is given by drawing a random variable $\alpha$ from a uniform distribution of our choice, using it as a weight for the benign and the adversarial example, and then adding some gaussian noise [13].

  $$x_{uniform\_AdvMix} = a * x + (1 - a) * x' + n \tag{3.7}$$

  where $n \sim \mathcal{N}(0, 0.1)$ ($\mathcal{N}$ is a normal distribution) and $a \sim \mathcal{U}[0.2, 0.8]$ ($\mathcal{U}$ is a uniform distribution)

The new samples generated using adversarial mixup are labeled as the NOTA class and are concatenated with the benign examples before being passed to the model for training. In each case, the adversarial mix up samples are produced during the loading of a batch sample for training. In this architecture, we used both MixUp methods because it adds more randomness to the model by training the NOTA class in more adversarial examples. Algorithm 1 outlines the adversarial mixup data augmentation process and can be adjusted in any batch size. The output of this algorithm serves as the new input to the network, including the NOTA class and its associated features during the training phase.

```
Algorithm 1 Generate NOTA augmented dataset
Require: $X$ array with the original batch samples
Require: $Y$ array with the ground truth label of the samples
Require: $num\_classes$ the number of classes
Require: $n$ sample from $\mathcal{N}(0, 0.1)$
Require: $a$ sample from $\mathcal{U}[0.2, 0.8]$
Require: $adversarial\_method(x)$ function that returns the perturbed example using a
    desired adversarial attack method
Require: $append(A, x)$ function that appends $x$ to the array $A$
Ensure: $X\_train$ the new augmented training batch and $Y\_train$ the associated labels for
    $X\_train$.
X_train ← empty float array
$Y\_train \leftarrow$ empty integer array
$nota\_class \leftarrow num\_classes + 1$
for x in $X$ do
    $X\_train \leftarrow append(X\_train, x)$
    $Y\_train \leftarrow append(Y\_train, y)$
    $x' \leftarrow adversarial\_method(x)$
    $mean\_advmix \leftarrow (x' + x)/2 + n$
    $X\_train \leftarrow append(X\_train, mean\_advmix)$
    $Y\_train \leftarrow append(Y\_train, nota\_class)$
    $uniform\_advmix \leftarrow a * x + (1 - a) * x' + n$
    $X\_train \leftarrow append(X\_train, uniform\_advmix)$
    $Y\_train \leftarrow append(Y\_train, nota\_class)$
end for
```

### 3.2.3 Adversarial Datasets

We experiment with two different methods to produce adversarial examples:

- **Generate adversarial examples on-the-fly:** This technique integrates data augmentation into the model's training procedure. The advantage is that the space complexity remains the same as the benign dataset, but the training time of the model increases significantly and depends greatly on the selected adversarial method.

- **Generate adversarial examples offline:** In this technique, we generate adversarial examples in a separate pipeline and map them to the benign dataset. It requires double the space on disk, but the main advantage is that they can be reused and the tasks can be easily distributed among different computational resources. Figure 3.2 illustrates the pipeline of this technique.

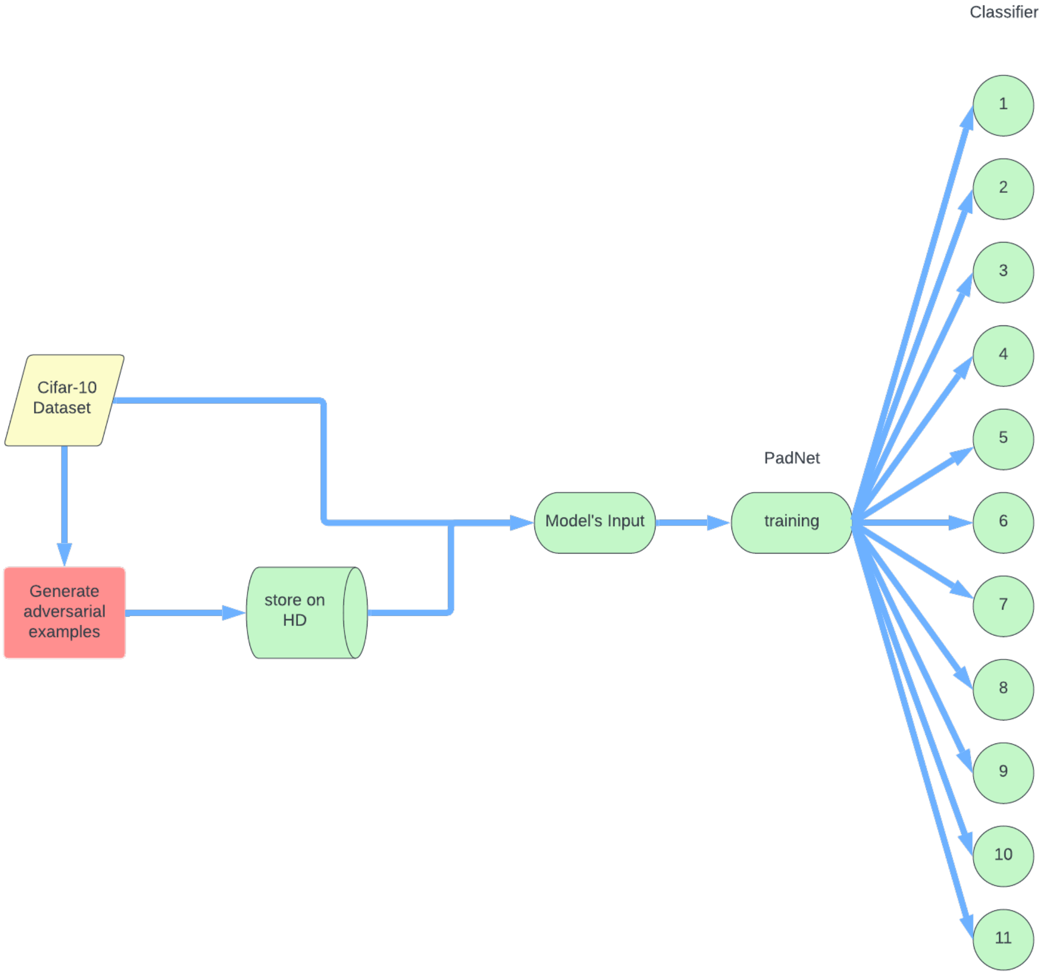


Figure 3.2. AdvMix pipeline using offline mode.

In fast adversarial attacks, such as the PGD with a small number of iterations, we experimented with the on-the-fly mode. For more time-intensive methods, we used the offline mode. The Adversarial Robustness Toolbox (ART)[1] library was used to develop and perform the attacks in all cases. For each attack performed, a new adversarial dataset was created and used to compute the uniform adversarial mixup, as mentioned in Subsection 3.2.2. The listed adversarial examples were generated to train a AdvMix model. Each one was named after the adversarial method that was used:

- Data augmentation using PGD attack with 1 iteration and a maximum perturbation of 0.031. During training, we used an adaptive loss function, which means that half of the time we maximize loss between the prediction and truth label, and the other half we maximize loss between the prediction and NOTA class. *Name:* AdvPGD_1_AdvMix.
- Data augmentation using PGD attack with 1 iteration and a maximum perturbation of 0.031 on the fly. *Name:* Mp_AdvMix.
- Data augmentation using PGD attack with 10 iterations and a maximum perturbation of 0.031. *Name:* PGD_10_AdvMix.
- Data augmentation using FGSM attack with a maximum perturbation of 0.031. *Name:* Fgsm_AdvMix.
- Data augmentation using DeepFool attack with 100 iterations and a confidence of 0. *Name:* DeepFool_AdvMix.
- Data augmentation using Square attack with 100 iterations and a maximum perturbation of 0.031. *Name:* Square_AdvMix.
- Data augmentation using Carlini $L_2$ attack with 2 iterations and a confidence of 0. *Name:* CW_2_AdvMix.
- Data augmentation using Carlini $L_2$ attack with 10 iterations and a confidence of 0. *Name:* CW_10_AdvMix.
- Data augmentation using JSMA attack with a gamma of 1. *Name:* Jsm_AdvMix.

## 3.3 Evaluation Methodology

After training, each model was evaluated using a brute-force approach. The first 100 samples of the CIFAR-10 dataset were used and different attacks were generated. The goal of the

[1] official API https://adversarial-robustness-toolbox.org/

evaluation procedure is to compare the robustness of the different models that we trained and get a better insight into their behavior against each attack on the smallest attacking perturbation. For the attacks that are based on $\epsilon$ as a magnitude for the perturbation, we provide an $\epsilon = 0.031$. The reason is that it is proved to be the most efficient value for CIFAR-10 and it is tight to $L_\infty$ and $L_2$ norms [9]. For attacks that required more than one iteration to find the most efficient perturbation, values of 10,100, and 1000 were passed as the maximum number of iterations. As we increase the number of iterations the attack tries to find the smallest perturbation to make the model misclassify. For measuring the robustness of the model, we increase the maximum interrelations periodically [13]. We would like to see the behavior of AdvMix models against unbounded and inefficient attacks. The JSMA attack was chosen for this purpose, with different $\gamma$ values. The bigger the $\gamma$ value is, the stronger the perturbation of each pixel is, which makes the attack inefficient because the perturbation is noticeable by the human eye. In Figures 3.3 and 3.4, we visualize how adversarial examples are produced using a Carlini and Wagner $L_2$ attack and a JSMA attack with a $\gamma = 1$, respectively, to compare them. In the first case with the Carlini and Wagner attack, we are not able to detect the perturbation in the adversarial example, while in JSMA attack, we can easily detect it.

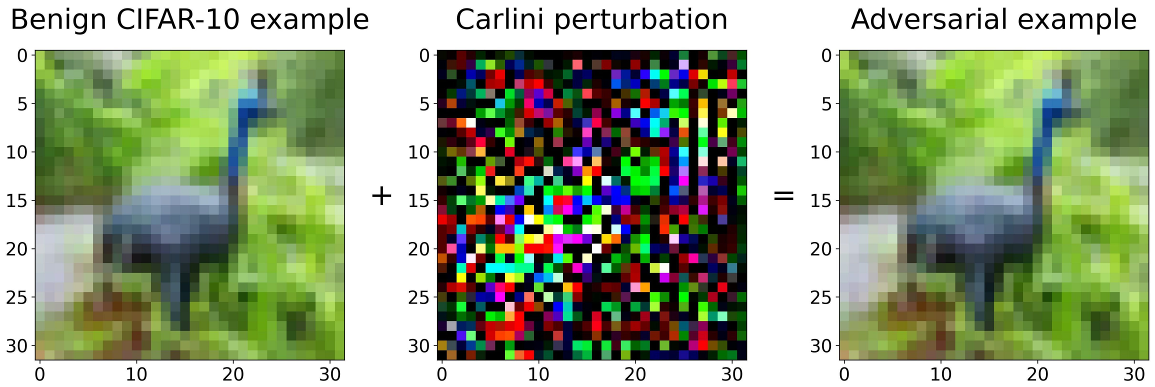


Figure 3.3. Carlini and Wanger adversarial example pipeline.

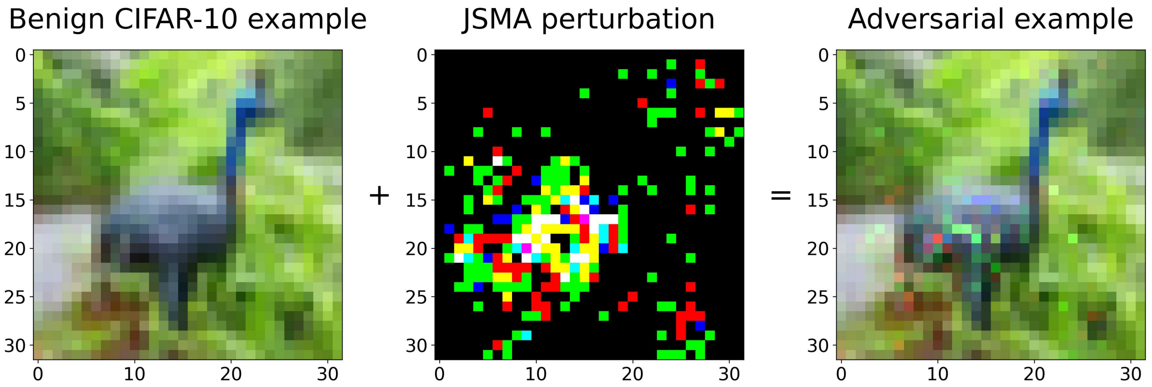


Figure 3.4. JSMA adversarial example pipeline.

A brief summary of all attacking methods and their hyperparameters is provided in Table 3.2 white-box evaluations were performed because are considered more efficient than the black-box attacks because the attacker has full knowledge of the model. The metrics used were the accuracy of the model on the benign CIFAR-10 dataset, the accuracy of the first 100 adversarial examples, and the Attack Success Rate (ASR). A model's prediction was considered accurate if it correctly predicted the ground truth label of a benign example. An attack was considered successful if the model misclassified the ground truth label and did not classify it as a NOTA class.

Table 3.2. Attacking methods in the evaluation procedure with their hyperparameters

| **Attacking method** | **Maximum iterations** | **confidence** | **magnitude** | **gamma** | **Batch size** |
|---|---|---|---|---|---|
| AutoAttack | - | - | 0.031 | - | 32 |
| PGD | 10, 100, 1000 | - | 0.031 | - | 32 |
| Square | 10, 100, 1000 | - | 0.031 | - | 32 |
| C&W $L_2$ | 10, 100, 1000 | 0 | - | - | 32 |
| C&W $L_{inf}$ | 10, 100, 1000 | 0 | - | - | 32 |
| Deepfool | 10, 100, 1000 | - | 0.031 | - | 32 |
| AutoPGD | 10, 100, 1000 | - | 0.031 | - | 32 |
| EAD | 10, 100, 1000 | 0 | - | - | 32 |
| Boundary | 10, 100, 1000 | - | 0.031 | - | 32 |
| JSMA | - | - | 0.031 | 0.1, 0.2, 0.5, 1 | 32 |

THIS PAGE INTENTIONALLY LEFT BLANK

# CHAPTER 4: Results

This chapter presents the results of training various machine learning models under the AdvMix network architecture, as well as their evaluations under our brute force evaluation algorithm discussed in Section 3.3. We first describe the models based on their training process. We then evaluate the models' performance and their detection ability under several adversarial attacks, using white-box attacks which as described in Chapter 2 are considered the most efficient. Our results show that the AdvMix network architecture successfully detects adversarial examples in almost all attacks, making them inefficient and that some models perform better than others under specific attack scenarios. These findings provide insights into the effectiveness of AdvMix as an augmentation method for detecting adversarial examples and improving model robustness.

## 4.1 Training results

We trained nine different models on a AdvMix architecture using GPUs, and each model was trained for approximately 300 epochs. For each training circle, we used an RTX6000 GPU manufactured by NVIDIA along with 64 gigabytes of memory and 10 CPUs. The first model trained was the one with its NOTA class based on the PGD attack. Figure 4.1 shows that the model was trained properly, and the testing accuracy increased until it reached a plateau, while the attack success rate decreased. This indicates that the network learned its parameters until it reached its full capacity. During the training procedure, we performed a Carlini and Wagner $L_2$ attack on a small portion of 15 adversarial examples, which produced a noisy attacking success rate. We chose this attack because it is one of the most efficient and successful attacks. After training, we kept the model with the best accuracy and the lowest attack success rate to perform a more comprehensive and accurate evaluation. It took approximately two days for the training process.

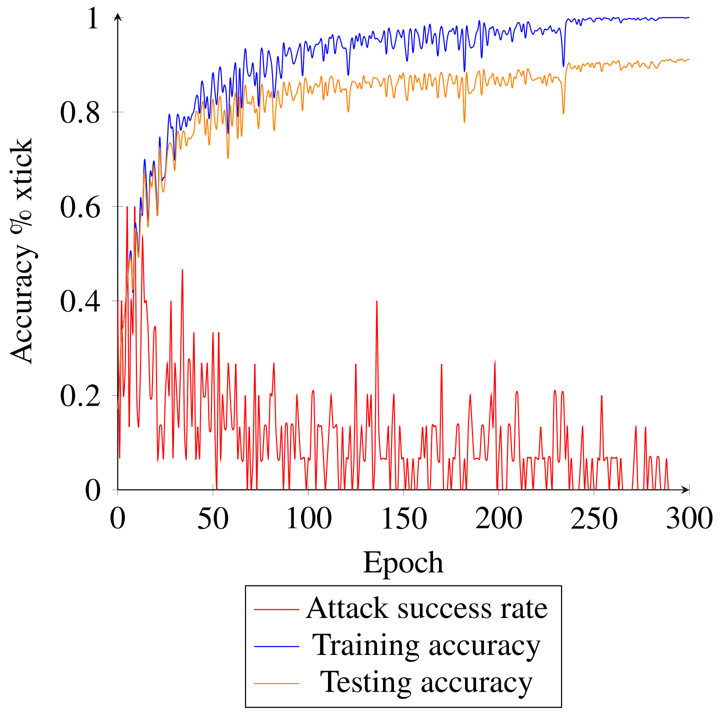


Figure 4.1. AdvMix's Training history of accuracy and attack success rate for an adversarial PGD mixup model

All nine models were trained using the same procedure as the first model. Figure 4.2 displays the accuracy of each model on the benign CIFAR-10 dataset. We observe that all the detection models, with the exception of CW_2_AdvMix and CW_10_AdvMix, demonstrate an increase in accuracy of 0 to 4 units compared to the vanilla model. The reason for this improvement is the use of the mixup data augmentation technique, which is implemented in the AdvMix network. The linear interpolation of the benign input and the perturbed input, along with their labels, helps the model generalize better and adds regularization by forcing the model to have linear behavior between benign samples and adversarial example space near benign samples. [34]. On the other hand, CW_2_AdvMix and CW_10_AdvMix exhibit a decrease in accuracy by 7 and 6 units, respectively. This phenomenon is expected since the borders of each class are compressed to create the NOTA class. The change in class borders is particularly noticeable when the Carlini and Wagner methods are used to create the NOTA class.

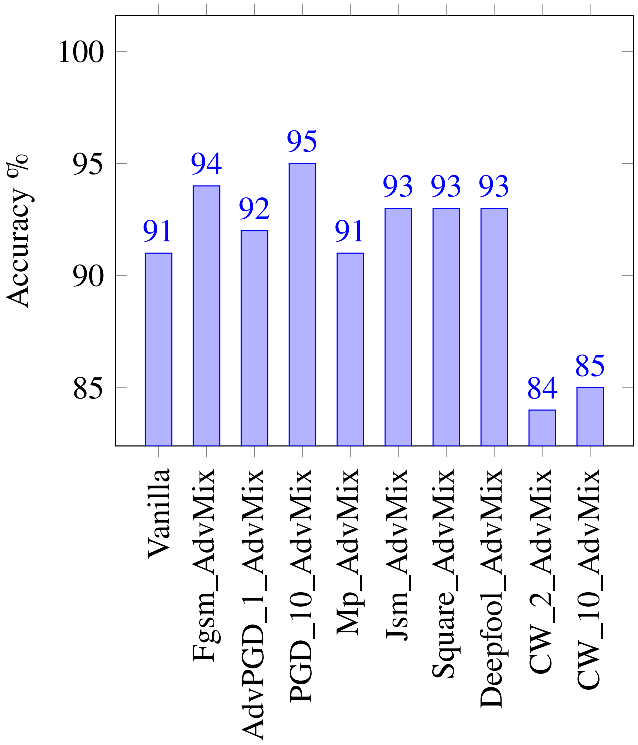


Figure 4.2. Accuracy on the benign CIFAR-10 for each trained model

## 4.2 Evaluation Results

In this subsection, we present the results for each model categorized by the maximum number of iterations, starting with the white box attacks followed by the black box attacks. Each model is evaluated against all attacks provided in table 3.2

### 4.2.1 White box evaluation

**10 iterations attacks**

Figures 4.3 and 4.4 depict the accuracy of each model on adversarial examples produced by each attack. A correct prediction is considered if the model predicts the class as its ground truth before adding perturbation. This measure helps us to observe the model's behavior in each attack compared to the ground truth label. The range of accuracy is from 0 to 89. Note that the vanilla model has 10 classes while the AdvMix models have 11. This means that we do not expect the AdvMix models to have good accuracy as they do not focus on increasing or maintaining their accuracy. Some models perform significantly better against certain attacks than others. For instance, the Vanilla model performs very well against PGD and AutoPGD attacks, with an accuracy of 89 and 82, respectively. This indicates that these attacks are not very strong, and the model classifies the inputs correctly. However, the same model performs very poorly in other attacks like Carlini and Wagner $\ell_2$ and $\ell_\infty$, with an accuracy of 6 and 5, respectively. Overall, it seems that the CW_10_AdvMix model performs the best across most of the attacks, with higher average accuracy values than the other models. It is worth recalling Figure 4.2, where the CW_10_AdvMix model had the lowest accuracy of all the other models, making it more robust against most of the attacks.

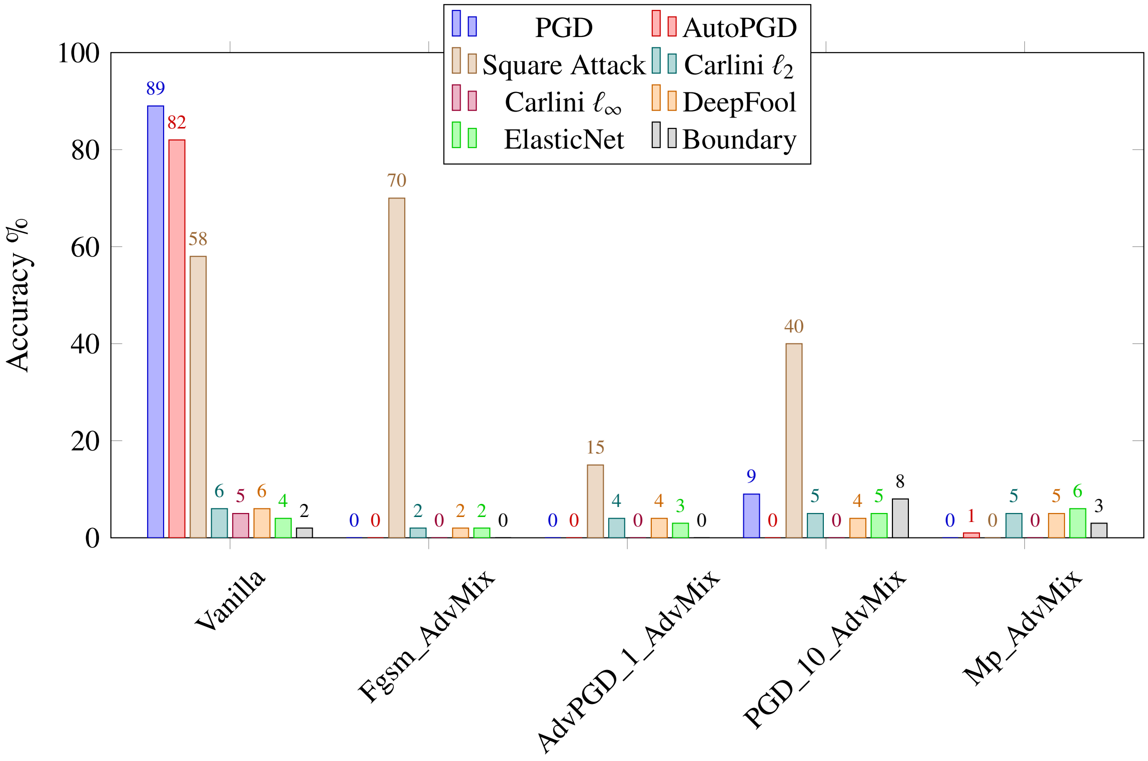


Figure 4.3. Accuracy of 5 first models on adversarial examples against attacks that use 10 iterations.

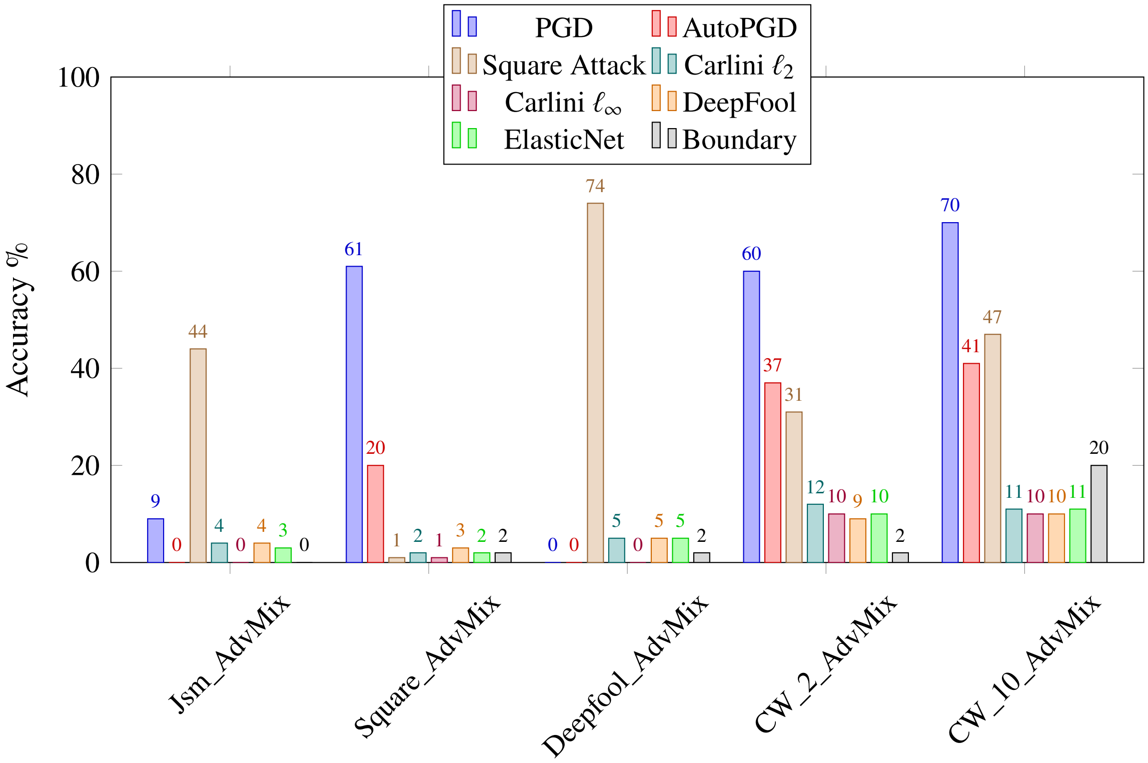


Figure 4.4. Accuracy of 5 last models on adversarial examples against attacks that use 10 iterations.

As mentioned many times, the goal of AdvMix is to detect adversarial examples and improve the robustness of a model. In Figures 4.5 and 4.6, we can observe the ASR of each attack against each model, which is a measurement of detection. When a model classifies the input as a NOTA class, it means that there is no successful attack, but at the same time, the accuracy (as mentioned above) is decreasing. The Vanilla model has the highest ASR for most of the attacks, with ASR ranging from 11% to 98%. As we know, this suggests that the Vanilla model has a high vulnerability to adversarial attacks. The AdvPGD_1_AdvMix, PGD_10_AdvMix, Square_AdvMix, and Deepfool_AdvMix models have mixed ASR. The PGD_10_AdvMix model is particularly vulnerable to the AutoPGD attack, with a success rate of 73%. That indicates that even if the model is trained to similar examples, it is still vulnerable to that kind of attack. The Mp_AdvMix and Jsm_AdvMix models have low success rates for most attacks, making them the more robust models among the other ones. The CW_2_AdvMix and CW_10_AdvMix models have relatively high success rates for all attacks except the PGD attacks, suggesting that they are more vulnerable to adversarial attacks than the other models. We can notice that all AdvMix models, even the "worst" ones, are much more robust than the vanilla model, with MP_AdvMix being the one with the best performance. The average attack success rate reduced from 68.5% of the vanilla model to 21.6% of the PGD_10_AdvMix, which is the worst AdvMix model, to 5.3% of the MP_AdvMix, which is the best one.

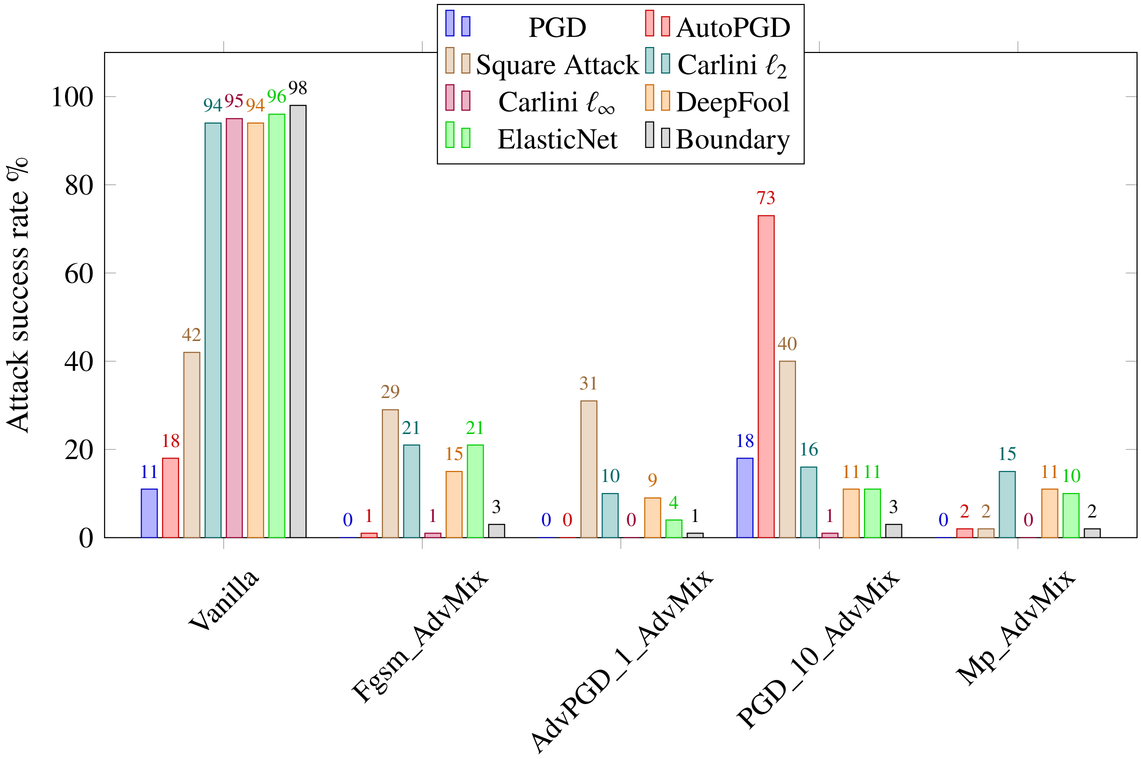


Figure 4.5. Attack success rate on 5 first models of attacks that use 10 iterations.

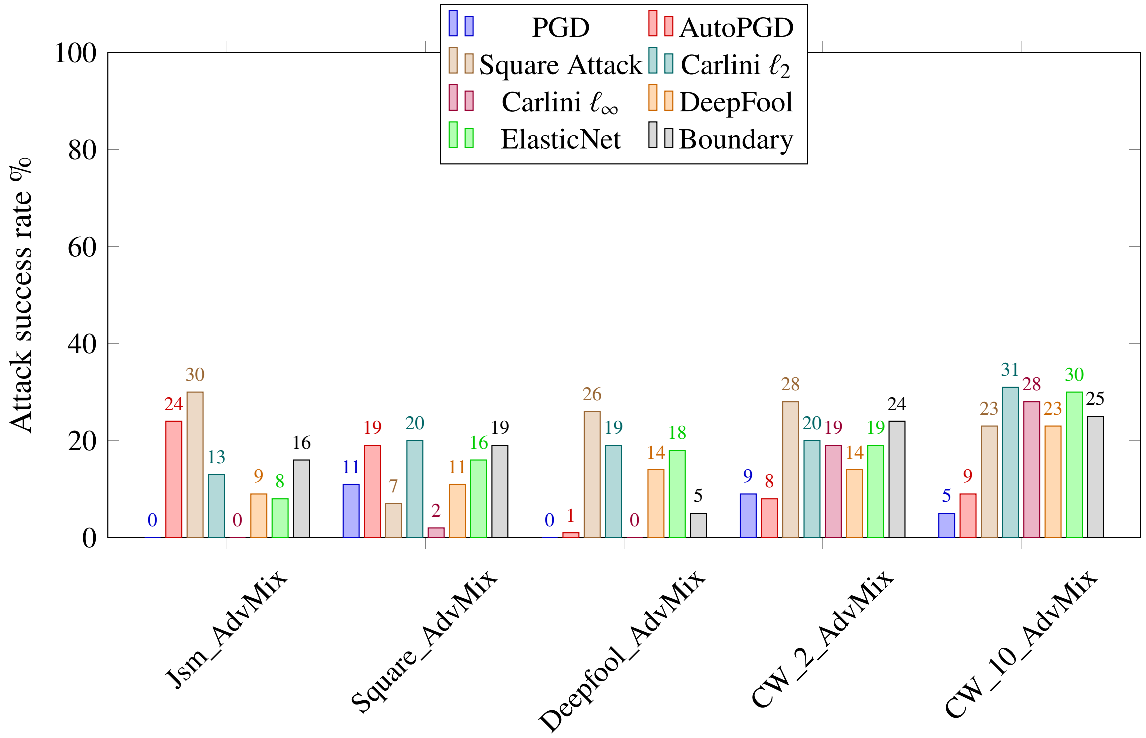


Figure 4.6. Attack success rate on last 5 models of attacks that use 10 iterations.

### 100 iterations attacks

We performed the same series of attacks using 100 maximum iterations. Figures 4.7 and 4.8 display the accuracy of each model against attacks that perform 100 maximum iterations. As discussed in Chapter 3, the depicted results were something we expected. More iterations do not necessarily mean a stronger attack. It means that the attack can find the most efficient and smallest perturbation to make the model misclassify. Some attacks, like the square, are much more effective than the one with 10 iterations. Overall, the Carlini and Wagner models appear to be the most robust models compared with all the others.

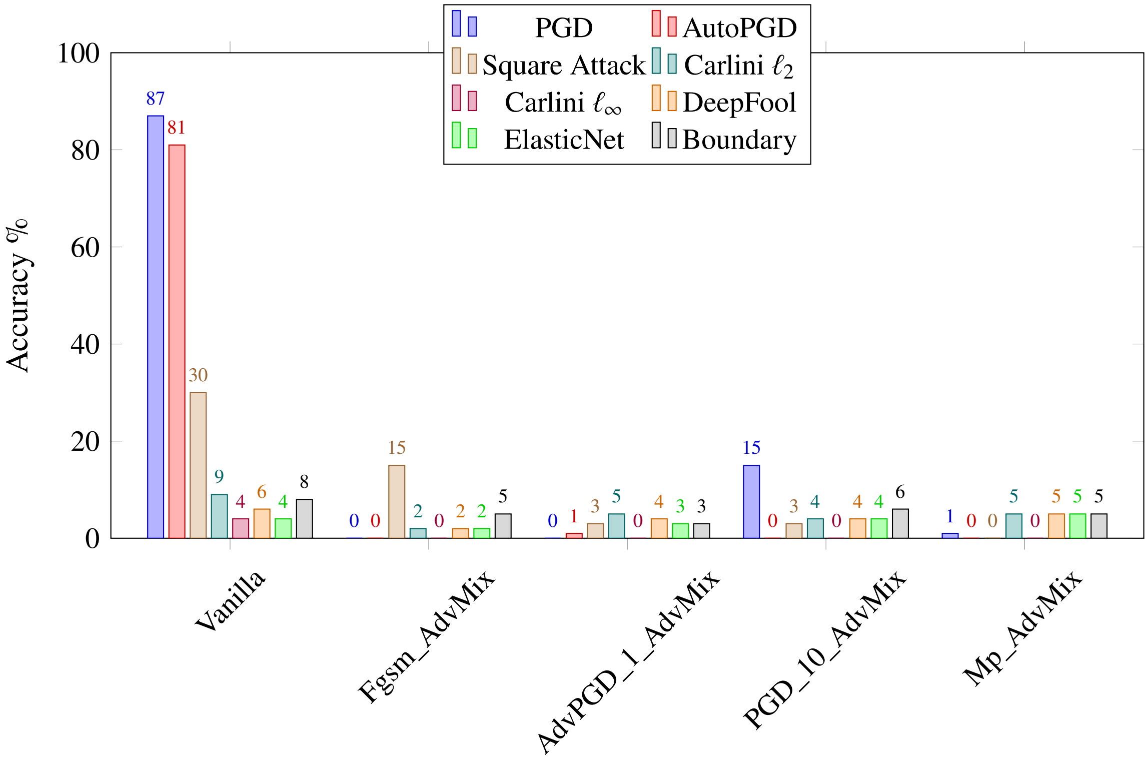


Figure 4.7. Accuracy of 5 first models on adversarial examples against attacks that use 100 iterations.

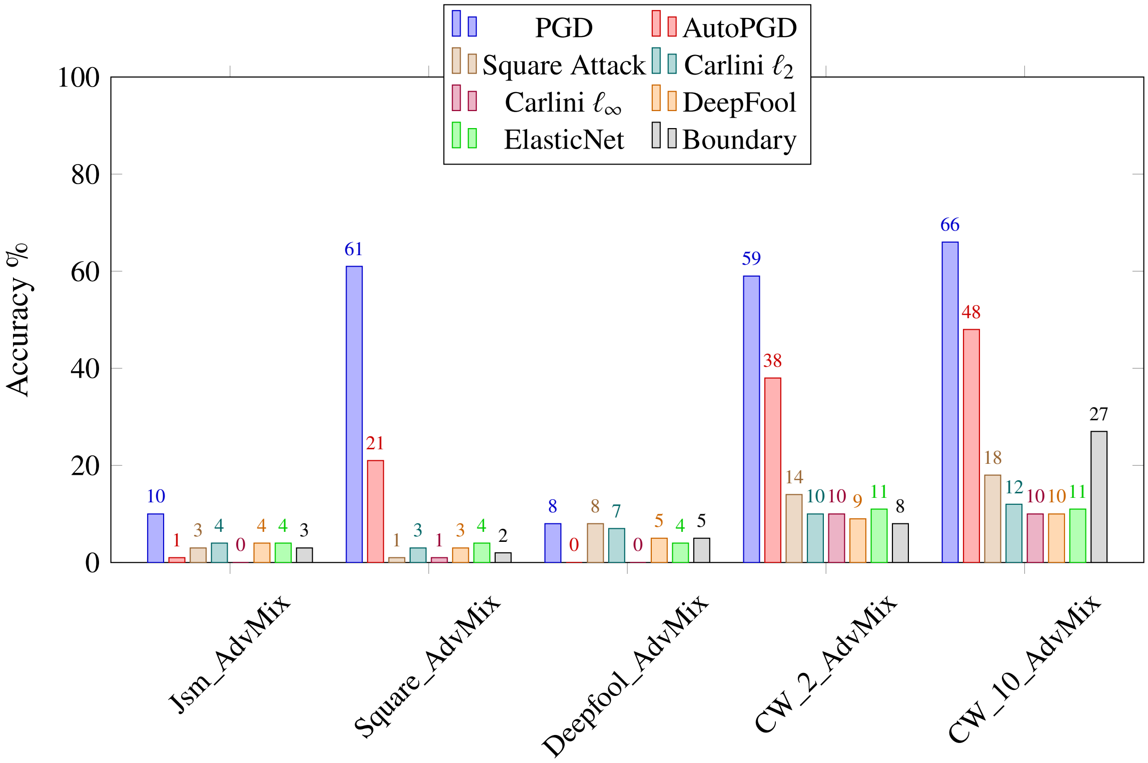


Figure 4.8. Accuracy of 5 last models on adversarial examples against attacks that use 100 iterations.

In Figures 4.9 and 4.10, we can see the success rate of each attack on all models. For the vanilla model, we can observe that the success rate for almost all attacks is almost 100%, except for the PGD and autoPGD attacks. The success rate of attacks still varies from model to model and even within the same model. For instance, the PGD attack success rate against the Vanilla model is only 13%, while against the Carlinilinf model, it is 96%. The square attack seems to have a good success rate for almost all models, ranging from 0% to 70% with an average of 36.8%. The average success rate against the Vanilla model increased from 68.5% to 71.4% compared to the attacks with 100 iterations. Overall, the Mp_AdvMix models appear to be the most robust against most attacks, with an average success rate of only 5% and relatively low success rates for all attacks. It is worth noting that while the average success rate of the Vanilla model is 71.4%, the worst AdvMix model's average success rate is only 30%, specifically the PGD_10_AdvMix.

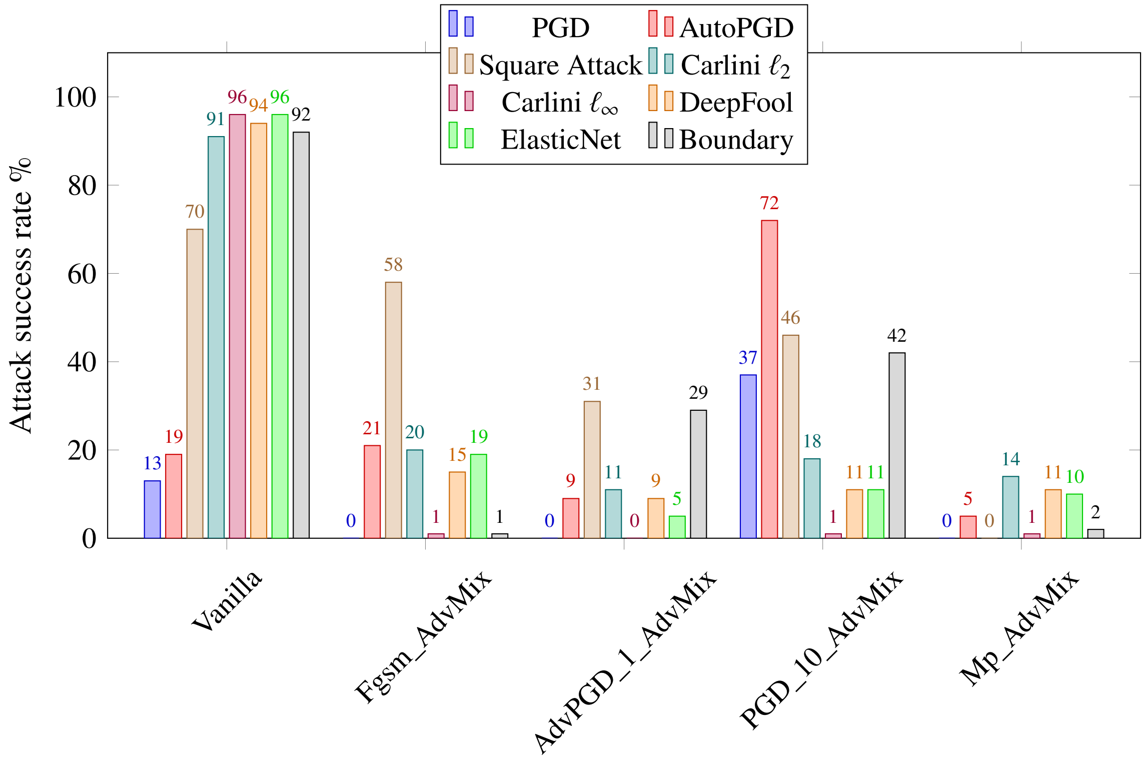


Figure 4.9. Attack success rate on 5 first models of attacks that use 100 iterations.

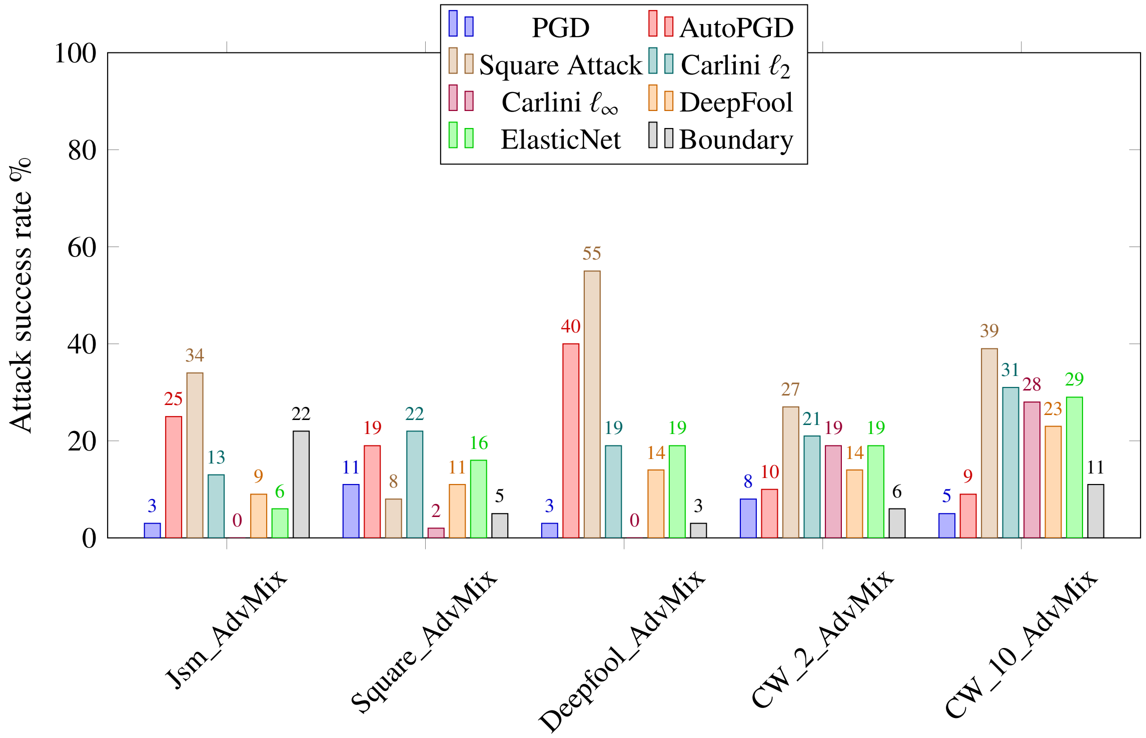


Figure 4.10. Attack success rate on last 5 models of attacks that use 100 iterations.

**1000 iterations attacks**

In this category, an unexpected problem was raised during experimentation. During the evaluation, some of the models caused the boundary attack to crash by creating an overflow problem. It appears that a boundary attack with too many iterations is not feasible using the ART library. Our solution was to limit the boundary attack to 800 iterations, which was the maximum number of iterations that did not cause a problem. In Figures 4.11, 4.12, 4.13, and 4.14 we denote these models with a star * at the end of their name (ex. Jsm_AdvMix*). As shown in Figures 4.11 and 4.12, there were no significant changes from the previous results. The AdvMix models performed the worst in classifying the adversarial example as the same as its ground truth label.

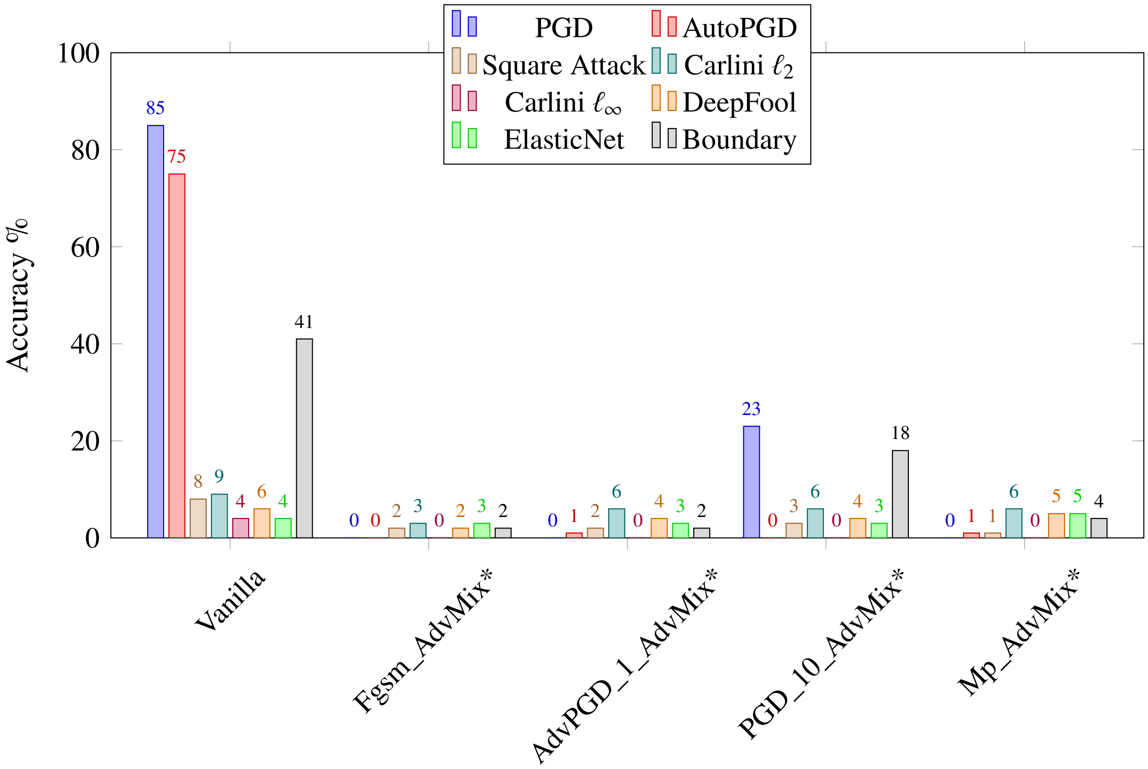


Figure 4.11. Accuracy of 5 first models on adversarial examples against attacks that use 1000 iterations.

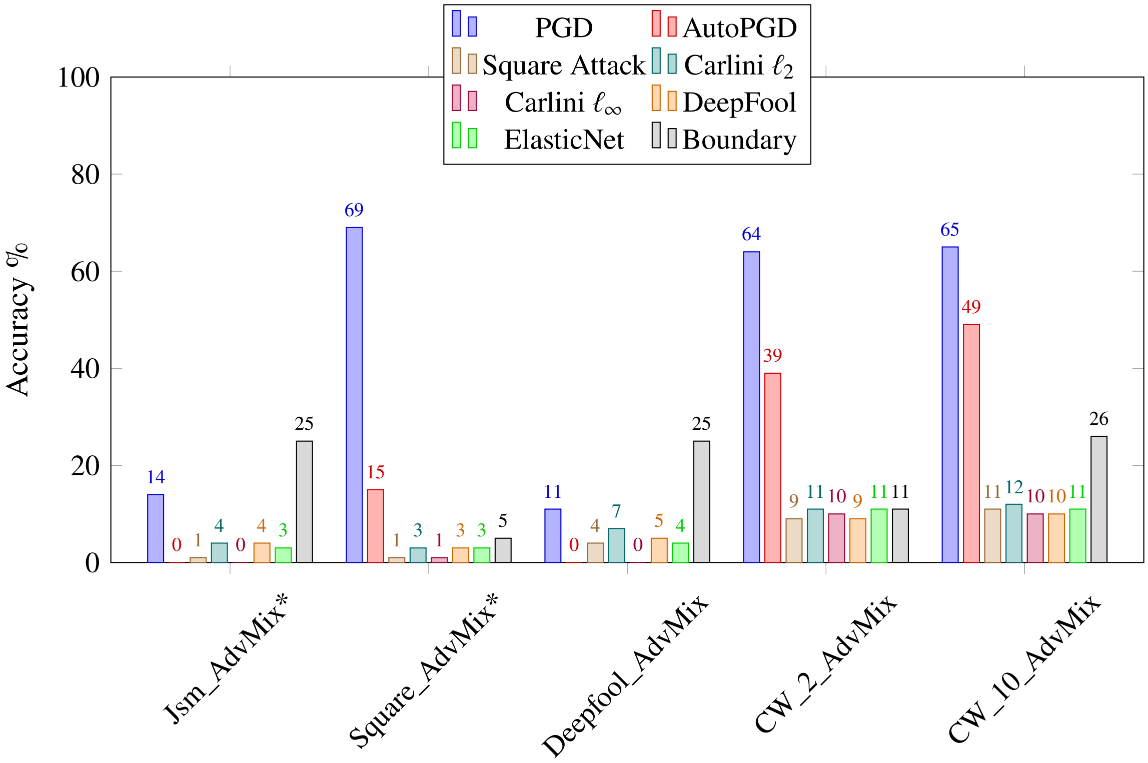


Figure 4.12. Accuracy of 5 last models on adversarial examples against attacks that use 1000 iterations.

As the maximum iterations increase, the attacks become more efficient, which is why the attack success rate increases, as shown in Figures 4.13, and 4.14. From the data, we can observe that the Vanilla model has the highest success rate on most of the attack methods, except for PGD and autoPGD, where it has a success rate of 15% and 25%, respectively. The Fgsm_AdvMix model has a low success rate against most of the attack methods, with the exception of Square, where it has a success rate of 70%. The AdvPGD_1_AdvMix model has relatively low success rates against most of the attack methods, with the most effective attack being the Square attack, where it achieves a success rate of 28%. The Mp_AdvMix model has a low success rate against most of the attack methods. It is worth mentioning that this is the only model for which the Square attack was completely defended, achieving a success rate of 0%. The Jsm_AdvMix model has a moderate success rate against most of the attack methods, with a success rate of around 20% against most of them. The Square_AdvMix model also has a good success rate against most of the attack methods, with a success rate of around 20% against most of them. Carlini $\ell_2$ has the highest success rate of 22%. The Deepfool_AdvMix model has a moderate to high success rate against most of the attack methods, with a success rate of around 17% against most of them, but it seems to be more vulnerable to the Square attack, achieving an accuracy of 69%. The CW_2_AdvMix and CW_10_AdvMix models have medium to good success rates against most of the attack methods, with an average success rate of around 16% to 23%, respectively. That is something notable because, on the attacks with fewer iterations, those models performed worst than now. Again, it seems that the most effective attack is the Square attack, where it achieves a success rate of 48%. Finally, the last row shows the average success rate for each attack method across all models. We can see that Carlini $\ell_2$ and square attacks have the highest average success rates, while Boundary has the lowest average success rate. Overall, Mp_AdvMix seems to be the most robust model with an average of 5%, followed by the AdvPGD_1_AdvMix model, which has an 8% attack success rate. The most vulnerable AdvMix model is PGD_10_AdvMix with an average success rate of 25%, which is still much lower than the Vanilla model.

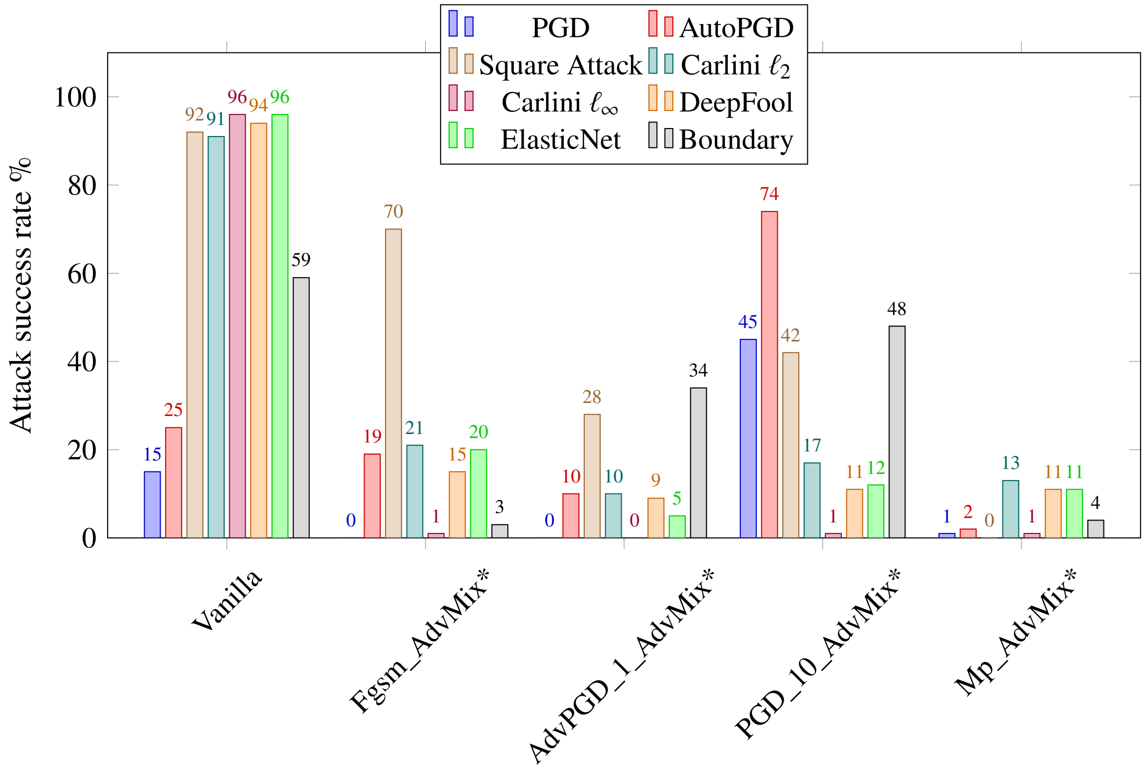


Figure 4.13. Attack success rate on 5 first models of attacks that use 1000 iterations.

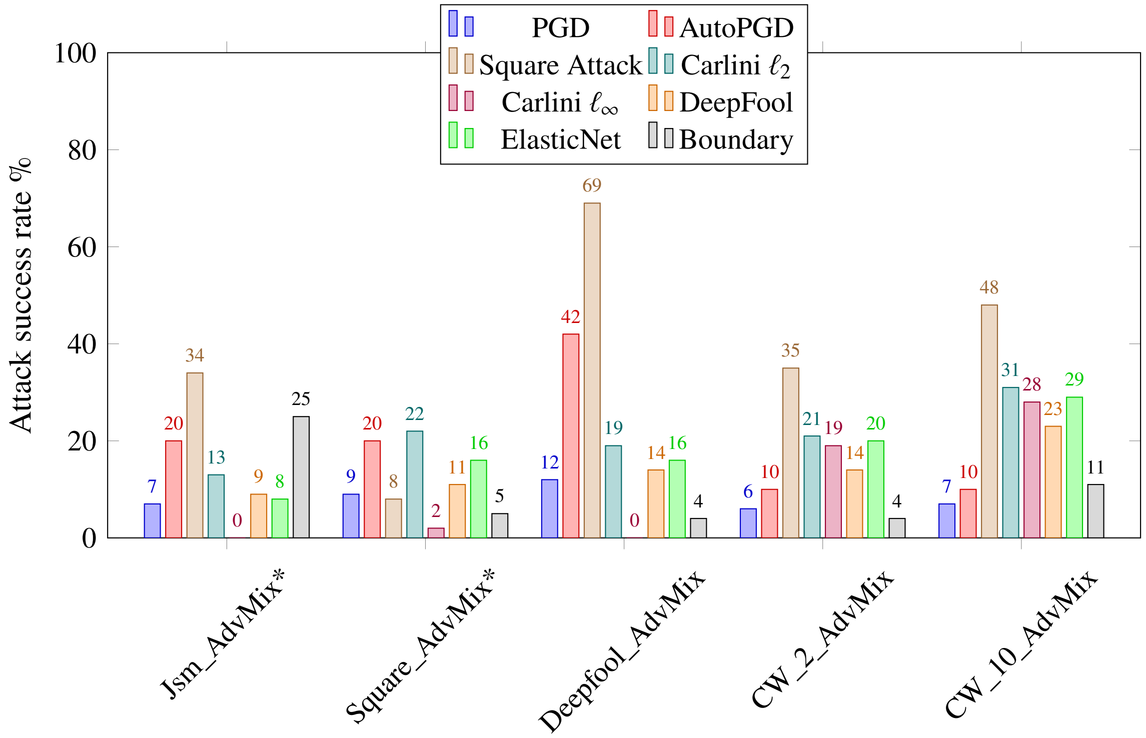


Figure 4.14. Attack success rate on last 5 models of attacks that use 1000 iterations.

**No iterations attacks**

Looking at the data in Figure 4.15, we can see that the robustness of the models drops significantly, even for the vanilla model, which wasn't the case previously, with accuracy rates of 4% or lower for most attacks. Some models appear to be more accurate compared with others, with the FGSM_AdvMix and Deepfool_AdvMix being the best model in this category, with a range accuracy between 0% to 18% and 0% to 24% respectively. It's worth reminding that the highest $\gamma$ value we have, the stronger the attack is. That means that it's not efficient, and for a value of $\gamma = 1$, we have an unbounded attack. The additional perturbation can easily be detected visually by the human eye, and that's why it's not preferred and not considered an efficient attack. However, it's still useful to consider such attacks to understand the limits of a model's robustness and develop better defense mechanisms.

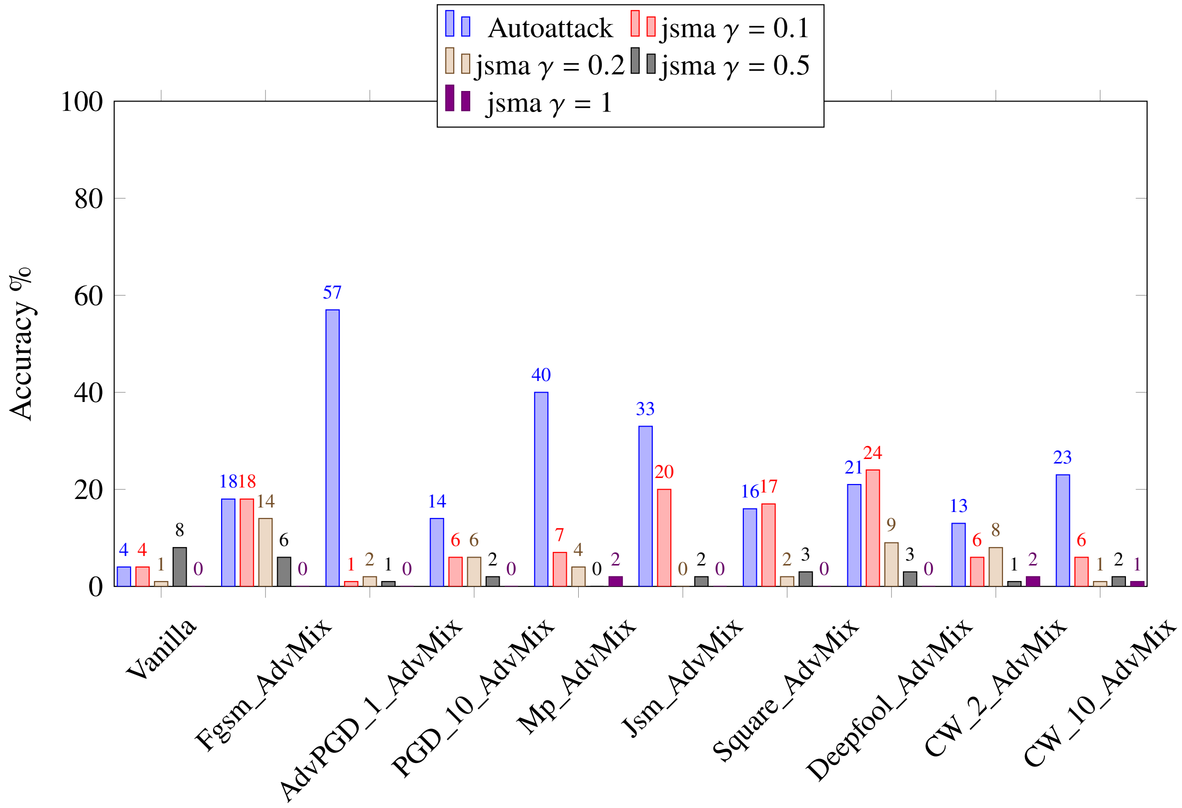


Figure 4.15. Accuracy of models on adversarial examples against attacks that use no iterations.

In figure 4.16, we can clearly see the aforementioned results of the JSMA attack with $\gamma = 1$, which appears to be highly effective, with success rates of 100% for most of the models as expected and discussed in section 3.3. JSMA attacks seem to be not as successful for a value of $\gamma = 0.1$ as for the other values of $\gamma$. The range of success rates is from 26% on AdvPGD_1_AdvMix to 84% on the CW_10_AdvMix model. In most cases, while the $\gamma$ value increases the ASR increases also by reaching almost 100%. The auto attack seems also to be one of the most effective attacks, reaching a success rate of 87% on AdvMix models and 96% on the Vanilla model. The adaptive mixup model AdvPGD_1_AdvMix seems to be more robust compared to all the others on those 2 attacks.

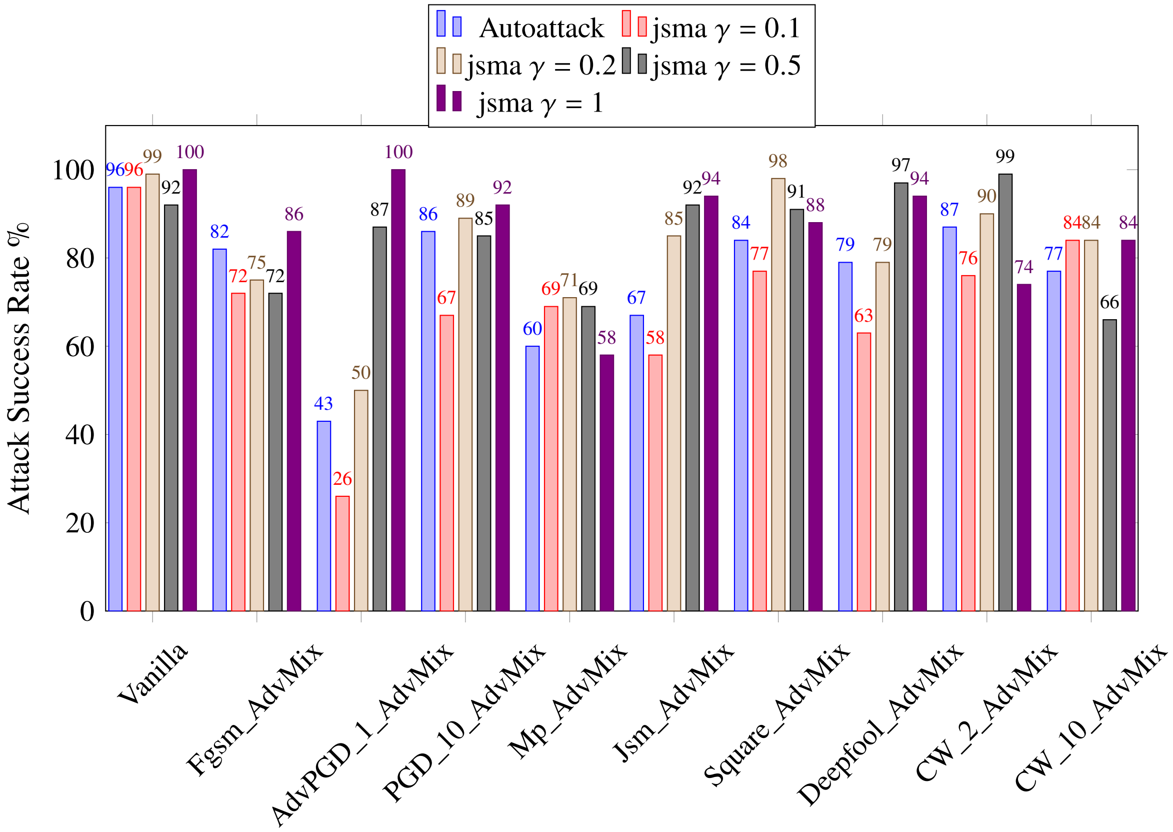


Figure 4.16. Attack success rate on models of attacks that use no iterations.

# CHAPTER 5: Discussion and future work

This chapter discusses the results obtained from the research study and provides a summary of the findings. The discussion section highlights the implications of the study, identifies the limitations, and suggests future directions for research. Additionally, a future work section proposes potential areas for improvement in the study methodology and data analysis. Finally, the conclusion summarizes the overall results and workflow of the study on AdvMix networks and emphasizes the importance of the research and how critical the application of such networks is in machine learning.

## 5.1 Discussion

### 5.1.1 Training and evaluation discussion

In our research, we trained nine different models in an AdvMix network, and we have two important findings. The first finding is that the benign accuracy of a model trained on an AdvMix network increases from 0 to 4 units, on top of an already good accuracy of the vanilla model. In Figure 4.2, we can see that the accuracy goes down only for the models that we used data augmentation on using Carlini and Wagner methods. This is very important because we can increase the benign accuracy of a model and make it generalize better while also increasing its robustness against various attacks.

The second finding is that our experiments demonstrate that the AdvMix network architecture can successfully detect adversarial examples in almost all attacks, making them inefficient. Our findings suggest that some models perform better than others under specific attack scenarios, and it is challenging to have a clear model that outperforms all attacks. As depicted in Chapter 4, some AdvMix models had poor performance among other models. Still, when we change the hyperparameters of the same attack, they perform better. In general, we can say that an AdvMix network makes a model much more robust not only against the attack that is used on data augmentation but against most attacks.

Because the purpose of this research was to detect adversarial examples with small perturbations, the findings on attacks like JSMA and square attack, where the attack success rate on AdvMix models is much higher, are only for getting an idea of how models react to higher perturbations. Our white-box evaluation results show that, in most cases, the AdvMix model with PGD data augmentation on the fly outperforms the other models. This gives us insight that using data augmentation inside the training procedure, rather than generating from a vanilla model, creates more robust models. It is more time-consuming, but the benefit is that the model creates adversarial examples for training by itself, and it is not trained on examples that a similar model produces.

### 5.1.2 Limitations

Our study has several limitations. First, we evaluated the AdvMix models only on the CIFAR-10 dataset, and it would be interesting to evaluate them on other bigger and more complex datasets to determine whether they can generalize well to different datasets. Another limitation was that we performed a brute-force approach to evaluate the models, which, combined with having limited computational resources, is a limitation because the hyperparameter space of each attack is an Non-deterministic Polynomial time (NP) problem. We focused only on tuning the hyperparameters for adding the smallest perturbation to the adversarial examples.

### 5.1.3 Unanswered questions

During the evaluation procedure of each model, we made some observations that we were not able to answer and need further investigation. Carlini and Wagner's attacking methods are one of the most efficient attacks, and we expected that using those methods for data augmentation, the benign dataset accuracy on unobserved data of the model would go up, as the other AdvMix models did. Instead, we noticed a drop of 6 to 7 units. Another observation is that for stronger attacks, such as JSMA with $\gamma = 0.1$, the attack success rate was reduced compared to the vanilla model but not as much as we expected. I expected to see a higher correlation between the data augmentation technique that we used and the attacking method during the evaluation. For example, the AdvMix model that is based on the JSMA attack, I expected to be more robust against at least the lower $\gamma$ values than the other models, but that wasn't the case. I expected that more robust behavior of the model because this is the standard process if we want to make a more robust model against a specific attack.

### 5.1.4 Real world application

Increasing the performance of models by training in an AdvMix network can be applicable in many real-world applications such as:

- Computer Vision: The use of AdvMix network in computer vision applications can enhance the performance of object detection and classification systems. It can improve the accuracy of facial recognition systems, which are sensitive to adversarial attacks.
- Cybersecurity: The AdvMix network can be used to train models that are more robust against cyber attacks and develop more robust and secure fraud detection systems. For example, for authentication, we usually use systems that need face recognition or a classification task to give permission to a user. The existing models are vulnerable at this time.
- Autonomous Vehicles: The AdvMix network can be used to train models that are more robust against adversarial attacks on autonomous vehicles. It can improve the safety and security of self-driving cars by detecting and preventing attacks on the vehicle's systems and ignoring the input of a sensor when such an example is detected.
- Healthcare: The AdvMix network can be used to develop more secure and robust medical diagnosis systems even if those attacks are not so common. It can help to detect and prevent attacks on medical devices and systems that use NNs models for classification, such as cancer detection, ensuring the safety and privacy of patient data.

Overall, the AdvMix network can be used in any application that requires robust and secure ML models. Its ability to detect and prevent adversarial attacks makes it an essential technology for the development of secure and reliable machine learning systems.

## 5.2 Future work

There are several suggestions for further development of the AdvMix network. The first suggestion is to create more test cases for data augmentation using auto-attack to generate adversarial examples. By using an ensemble of adversarial attacks, we can produce the most efficient one for data augmentation. This idea arises from the fact that auto-attack had a relatively high attack success rate compared to other methods. The second suggestion is to use more adversarial examples during data augmentation. Instead of mixing up two examples and calculating the mean of their distance, we can use more than two and calculate the centroid of their position. Then we can use that centroid for training. Inspired by ensemble ML models where many poor models are better than one very good model, we believe that using this approach will enable us to define the borders of the NOTA class better. Another suggestion is to try training a model using patch attack [38], which is more applicable in real-world scenarios, such as fooling an object detection system. Finally, we suggest experimenting with the AdvMix network and transfer learning. This would allow us to increase the robustness of already-deployed models that have high accuracy with less training.

# CHAPTER 6: Conclusion

While NNs are producing outstanding models with high accuracy on classification tasks, especially in the CV domain, they are vulnerable to adversarial attacks that add noise to an input, causing the models to misclassify with low certainty. Improving a model's robustness against all known attacks simultaneously is not trivial. We investigated the effectiveness of an AdvMix network in improving the robustness of deep neural networks against adversarial attacks by detecting the produced adversarial examples. Our findings show that the use of an AdvMix network can significantly improve the performance of models against various attacks while achieving better accuracy on benign examples. We were able to increase the accuracy of the vanilla WRN model from 91% to 95%.

To improve the robustness of the vanilla model, we found that the PGD data augmentation on the fly in an adaptive mixup model was the most effective technique. Our results show that the two most effective attacks are the auto-attack and the JSMA. In those attacks, we achieved a drop of the ASR of 53% in the auto-attack, 70% on the JSMA with a $\gamma = 0.1$, and 49% on $\gamma = 0.2$. Additionally, in all the other attacks, we almost eliminated their ASR, improving the robustness of the model even by 98% and making them inefficient. However, there are some limitations to our study, such as evaluating the models only on the CIFAR-10 dataset. It would be interesting to see if the results hold on other datasets. Another limitation is that we used a WRN architecture for the AdvMix model, and we would like to see how it performs in other architectures as well. Although we used only white-box attacks, we do not consider this a limitation, as it is generally more efficient than black-box attacks. In black-box attacks, we expect even better results.

Overall, the use of an AdvMix network has promising applications in various fields, including computer vision, cybersecurity, and autonomous vehicles. It can improve the performance of object detection and classification systems, develop more robust fraud detection systems, and enhance the safety and security of self-driving cars. In conclusion, our research demonstrates the importance of developing more robust machine learning models that can withstand adversarial attacks, and the AdvMix network provides a promising approach towards achieving this goal.

# List of References

[1] M. Haenlein and A. Kaplan, “A brief history of artificial intelligence: On the past, present, and future of artificial intelligence,” *California Management Review*, vol. 61, p. 000812561986492, 07 2019.

[2] L. Labs, “Clarifying ai, machine learning, deep learning, data science with venn diagrams,” Medium, 07 2020. Available: https://lotuslabs.medium.com/clarifying-ai-machine-learning-deep-learning-data-science-with-venn-diagrams-c94198faa063

[3] A. Géron, *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd ed. CA 95472: O’Reilly, 2019.

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[5] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning.” *Nature*, vol. 521, pp. 436–44, 5 2015.

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

[7] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” *arXiv preprint arXiv:1412.6572*, 2014.

[8] H. Kannan, A. Kurakin, and I. Goodfellow, “Adversarial logit pairing,” *arXiv preprint arXiv:1803.06373*, 2018.

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

[10] N. Carlini and D. Wagner, “Towards evaluating the robustness of neural networks,” in *2017 IEEE symposium on security and privacy (sp)*. IEEE, 2017, pp. 39–57.

[11] A. Athalye, L. Engstrom, A. Ilyas, and K. Kwok, “Synthesizing robust adversarial examples,” in *International conference on machine learning*. PMLR, 2018, pp. 284–293.

[12] A. Shafahi, W. R. Huang, C. Studer, S. Feizi, and T. Goldstein, “Are adversarial examples inevitable?” *arXiv preprint arXiv:1809.02104*, 2018.

[13] A. Barton *et al.*, “Defending neural networks against adversarial examples,” Ph.D. dissertation, University of Texas at Arlington, 2018.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Commun. ACM*, vol. 60, no. 6, p. 84–90, may 2017. Available: https://doi.org/10.1145/3065386

[15] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[16] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, 1998.

[17] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Communications of the ACM*, vol. 60, pp. 84 – 90, 2012.

[19] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *International Conference on Machine Learning*, 2010.

[20] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *International Conference on Artificial Intelligence and Statistics*, 2011.

[21] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2015.

[22] M. D. McDonnell, “Training wide residual networks for deployment using a single bit for each weight,” *arXiv preprint arXiv:1802.08530*, 2018.

[23] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, and D. Mukhopadhyay, “Adversarial attacks and defences: A survey,” *arXiv preprint arXiv:1810.00069*, 2018.

[24] A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial machine learning at scale,” *arXiv preprint arXiv:1611.01236*, 2016.

[25] F. Croce and M. Hein, “Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks,” in *International conference on machine learning*. PMLR, 2020, pp. 2206–2216.

[26] P.-Y. Chen, Y. Sharma, H. Zhang, J. Yi, and C.-J. Hsieh, “Ead: elastic-net attacks to deep neural networks via adversarial examples,” in *Proceedings of the AAAI conference on artificial intelligence*, no. 1, 2018, vol. 32.

[27] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, “Deepfool: A simple and accurate method to fool deep neural networks, 2016,” 2016.

[28] N. Carlini, A. Athalye, N. Papernot, W. Brendel, J. Rauber, D. Tsipras, I. Goodfellow, A. Madry, and A. Kurakin, “On evaluating adversarial robustness,” *arXiv preprint arXiv:1902.06705*, 2019.

[29] M. Andriushchenko, F. Croce, N. Flammarion, and M. Hein, “Square attack: a query-efficient black-box adversarial attack via random search (2020),” *arXiv preprint arXiv:1912.00049*, 1912.

[30] W. Brendel, J. Rauber, and M. Bethge, “Decision-based adversarial attacks: Reliable attacks against black-box machine learning models,” *arXiv preprint arXiv:1712.04248*, 2017.

[31] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, “The limitations of deep learning in adversarial settings,” in *2016 IEEE European symposium on security and privacy (EuroS&P)*. IEEE, 2016, pp. 372–387.

[32] X. Yuan, P. He, Q. Zhu, R. R. Bhat, and X. Li, “Adversarial examples: Attacks and defenses for deep learning. corr abs/1712.07107 (2017),” *arXiv preprint arXiv:1712.07107*, 2017.

[33] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, “Ensemble adversarial training: Attacks and defenses,” *arXiv preprint arXiv:1705.07204*, 2017.

[34] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, “mixup: Beyond empirical risk minimization,” *arXiv preprint arXiv:1710.09412*, 2017.

[35] H. Zhang, Y. Yu, J. Jiao, E. Xing, L. El Ghaoui, and M. Jordan, “Theoretically principled trade-off between robustness and accuracy. arxiv 2019,” *arXiv preprint arXiv:1901.08573*, 2019.

[36] A. Ross and F. Doshi-Velez, “Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, no. 1, 2018, vol. 32.

[37] A. Krizhevsky, “Learning multiple layers of features from tiny images,” in *Technical Report TR-2009*, 2009.

[38] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer, “Adversarial patch,” *arXiv preprint arXiv:1712.09665*, 2017.

# Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California