7-20-2021

Robust filtering schemes for machine learning systems to defend Adversarial Attacks

Kishor Datta Gupta

Robust filtering schemes for machine learning systems to defend Adversarial Attacks

by

Kishor Datta Gupta

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

Major: Computer Science

The University of Memphis

August 2021

Acknowledgements

First and foremost, I want to thank my advisor, Dr. Dipankar Dasgupta, for your support throughout my smooth journey to Ph.D. During my Ph.D., I have enjoyed the privilege of interchanging ideas and having productive feedback from you. Your advice on research as well as on my career has been precious. My special thanks to my committee members Dr. Kan Yang, Dr. Deepak Venugopal, and Dr. Nirman Kumar, for sacrificing their scarce time. I also want to thank my MS supervisor Dr. Stefan Andrei for his encouragement to pursue Ph.D. and my undergrad supervisor Dr. Kazi Md. Rokibul Alam, and Dr. Sk. Mohammad Masudul Ahsan.

Needless to say, I have had tremendous support from my family in achieving this dissertation. Most importantly, I want to thank my loving and supportive wife, my soulmate Trisha Bhowmick, for being so understanding and putting up with me through the most challenging moments of my life. I am so thankful to my parents, especially my sister Uma for her complete confidence in me. I also want to thanks my lab mates, friends from home, and here for their supports. Without them, it would not be possible to achieve.

Abstract

Defenses against adversarial attacks are essential to ensure the reliability of machine learning models as their applications are expanding in different domains. Existing ML defense techniques have several limitations in practical use. I proposed a trustworthy framework that employs an adaptive strategy to inspect both inputs and decisions. In particular, data streams are examined by a series of diverse filters before sending to the learning system and then crossed checked its output through a diverse set of filters before making the final decision. My experimental results illustrated that the proposed active learning-based defense strategy could mitigate adaptive or advanced adversarial manipulations both in input and after with the model decision for a wide range of ML attacks by higher accuracy. Moreover, the output decision boundary inspection using a classification technique automatically reaffirms the reliability and increases the trustworthiness of any ML-Based decision support system. Unlike other defense strategies, my defense technique does not require adversarial sample generation, and updating the decision boundary for detection makes the defense systems robust to traditional adaptive attacks.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Machine Learning (ML) techniques have recently attained impressive performances on diverse and challenging problems such as malware/intrusion detection, image classification, object detection, speech recognition, face recognition in-the-wild, self-driving vehicles, just to name a few. In spite of their major breakthroughs in solving complex tasks, it has been lately discovered that ML techniques (especially artificial neural networks and data-driven artificial intelligence) are highly vulnerable to deliberately crafted samples either at training or at test time, which can easily subvert ML techniques' outcomes. The samples with deliberate perturbations are usually referred as 'adversarial examples' (a.k.a. wild pattern or adversarial attack), i.e., carefully-perturbed samples aimed to mislead the ML techniques. For instance, the arbitrary perturbations added in the benign malware binary vector/file can lead to a significant drop in accuracy of DNNs-based malware detection systems. Similarly, for image classification ML techniques, an adversarial example can be generated by adding some indiscernible perturbations into a given image. The resultant adversarial image is misclassified by the well-known ML classifiers, while a human being can still classify it correctly without spotting the deliberate added perturbations. In case of automatic speech-to-text transcription, a small perturbation (e.g., an arbitrary waveform) when added to the original waveform can cause it to be transcribed as any phrase malicious adversary chooses. The audio adversarial examples that are perceived one way

by a human but transcribed differently by a state-of-the-art speech-to-text transcription neural network. Also, an adversary can malignly modify labels of the samples to be used for (re-)training of ML techniques, which is known as poisoning attacks.

To safeguard ML techniques against malicious adversary, several countermeasure schemes have been proposed, which roughly fall within two categories: adversarial defense and adversarial detection. Frameworks in first category aim at improving the DNNs' robustness to classify AEs correctly, e.g., adversarial training, i.e., training the ML techniques with clean and malicious samples. While the frameworks in second category attempt to detect malicious samples before they are fed to ML technique's main architecture such as augmenting the ML technique's main model with a small "detector" sub-ML technique trained on both adversarial and original clean samples, which can be utilized to distinguish whether the input sample is an adversarial attack or not. Despite the current progress on increasing robustness of ML techniques against malicious attacks, the majority of existing countermeasures still do not scale well and have low generalization. Namely, adversaries (adversarial samples/input) yet pose great threats to machine learning (ML) and artificial intelligence (AI).

In figure 1.1,1.3 and 1.2, I illustrated three examples of adversarial attacks. In the first example, one person can change just by wearing an adversarial patch printed glass, In 1.2 example, a simple change of word changing the output class of the paragraph. and in the 1.3, it is illustrated that normal day to day speech can change by little noise.

Figure 1.1: Adversarial attack examples on face biometric. Adding a special sunglass can able to fool the face biometric.

Modern society immensely relies on highly interconnected cyberspace, which is prone to

Original Text Prediction: **Entailment** (Confidence = 86%)
Premise: A large group of protesters are walking down the street with signs. **Hypothesis:** Some people are holding up **signs** of protest in the street.
Adversarial Text Prediction: **Contradiction** (Confidence = 43%)
Premise: A large group of protesters are walking down the street with signs. **Hypothesis:** Some people are holding up **signals** of protest in the street.

Figure 1.2: Adversarial attack example on text (the green color word 'signs' changed to 'signal' which change the output class 'Entailment' to 'Contradiction'

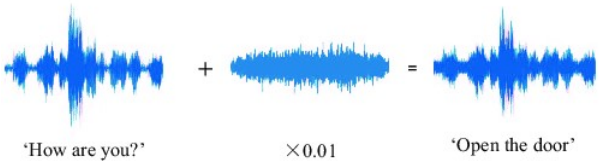


Figure 1.3: Adversarial attack example on Audio (normal audio record can be used as a voice command to access smart home)

vulnerabilities. For instance, from daily online shopping, cloud computing platforms to remotely controlled devices (i.e., Internet of Things) can be attacked or manipulated by adversaries. In particular, sample manipulations (adversarial attacks) may lead societies and individuals to untold risks with severe consequences. Sophisticated cyber adversaries are very difficult to detect and have severe negative impact on automated ML/AI. ML/AI based systems are exponentially being applied in diverse set of applications such as border crossing, autonomous vehicles, thus their security is paramount.

The proposed framework will be a game-changer in developing trustworthy ML systems and is very relevant to secure and trustworthy cyberspace programs. The successful outcome of this research will provide an additional layer of defense shield against attacks on learning systems. The outcome of this research will make it harder to attack any ML system without revealing specific attack methods. It will save a biometric (face+voice) recognition-based authentication system being attacked by hackers. ML-based applications such as intrusion detection techniques, news classifications, search engine optimizations, email spam filtering, video surveillance, Object detection applications are vulnerable to adversarial attacks. My findings will able to protect these applications from being exposed to adversarial attacks.

1.0.1 Contribution

We contributed a robust filtering schemes for machine learning systems to defend Adversarial Attacks which provide end to end protection. Our scheme incorporated with a diversity preserving variable-length MOGA for search set of Filters that are effective against a different type of AAs input as input filers and devised an adaptive negative filtering methodology to detect adversarial attacks that do not modify the ML model or information about the ML model but consistent with ML model outputs that able to capture TROJAI or backdoor. Our strategy can be implemented in any ML-based system without expensive retraining. Current Adaptive attacks are ineffective in our negative filtering approach as they are regenerating for each input. To summarize, the dissertation will address the aforementioned limitations in current state-of-the-art adversarial defenses and make the following contributions.

- *Identify natures of adversarial attacks.* I will identify some natural phenomenon of adversarial attacks and established benchmarks of adversarial attacks and defense.
- *Devise a defense strategy independent of MLM knowledge* I will design a defense method which will not require any MLM knowledge.
- *Explore input filters using Genetic Algorithm* The primary purpose of input filters is to prevent adversarial input data in such a way that can differentiate data manipulation from the trained data. It will be examining the input by deploying application-specific filter sequence. A set of filter sequences are selected (from a given library of filters) using an efficient search and optimization algorithm, called multi-objective genetic algorithm (MOGA). The MOGA can find a sequence of filters (where each filter can detect adversarial traits/noises) satisfying constrains and three objectives: detection of the maximum number of attacks with higher accuracy (above a specific threshold), with minimum processing time, and shorter sequence of ensemble filters. By utilizing the Pareto-set from MOGA runs, and picking a filter sequence dynamically at different times,

make filter selections unpredictable and use an active learning approach in order to protect the ML from adaptive attacks.

- *Output filter after MLM:* Employ several class-specific latent space-based transformation for outlier detection. After MLM provides an output class label, it is then verified if the output falls in that class's latent space or not. I will make an ensemble of different outlier detection methods and sequence dynamically and also retrain the outlier methods runtime.

Publications

Publications resulted from my research as below:

- **Patent**: System for Dual-Filtering for Learning Systems to Prevent Adversarial Attacks. 63/022,323
- **Conference:**
  - Gupta, Kishor Datta, Dipankar Dasgupta, and Zahid Akhtar. "Adversarial Input Detection Using Image Processing Techniques (IPT)." In 2020 11th IEEE Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON), pp. 0309-0315. IEEE, 2020. https://doi.org/10.1109/UEMCON51285.2020.9298060 [75]
  - Gupta, Kishor Datta, Dipankar Dasgupta, and Zahid Akhtar. "Applicability issues of evasion-based adversarial attacks and mitigation techniques." In 2020 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1506-1515. IEEE, 2020. https://doi.org/10.1109/SSCI47803.2020.9308589 [76]
  - Gupta, Kishor Datta, and Dipankar Dasgupta. "Using Negative Detectors for Identifying Adversarial Data Manipulation in Machine Learning" In 2021 International Joint Conference on Neural Networks (IJCNN), Shenzhen, China, July18–22, 2021. [73]

- **Journal**:
  - Gupta, Kishor Datta and Dipankar Dasgupta. “Dual-Filtering (DF) Schemes for Learning Systems to prevent Adversarial Attacks” Journal: Springer Complex Intelligent Systems, Manuscript ID: CAIS-D-21-00347, Submission date: March 2021. (under review)
  - Gupta, Kishor Datta, and Dipankar Dasgupta. “Adaptive Ensemble of Filters (AEF) to Detect Adversarial Inputs” Journal:IEEE Transactions on Emerging Topics in Computational Intelligence (TETCI) (under Review)
  - Gupta, Kishor Datta, Dipankar Dasgupta, and Zahid Akhtar. “Determining Sequence of Image Processing Technique (IPT) to Detect Adversarial Attacks” Journal: Springer Nature Computer Science, Manuscript ID: SNCS-D-20-01775, Submission date: October 2020. (Accepted) [77],
  - Gupta, Kishor Datta, Dipankar Dasgupta, and Zahid Akhtar. Negative Selection Algorithm Research and Applications in the last decade: A Review” Journal: IEEE Transaction of Artificial Intelligence, Submission date: May 2021. (Under second review),

Dissertation Organization

The rest of the dissertation is organized as follows: The second chapter discusses the preliminary topics which are related to this dissertation. An extensive literature study on adversarial attack and defenses are presented here.

In the third chapter, I detailed my goals and objectives. I also introduced the basic structure of my proposed defense system. I also presented the data set used to conduct experiments. I also briefly described my threat model, which I use to evaluate my defense strategy. This section also has detailed preliminary investigations and findings of my study.

In the fourth chapter, I explained how I ensemble input filter library and what are the

features used for that. The Multi-objective Genetic algorithm employed to search a suitable set of the sequence is described herein detailed. This section also discusses how outlier detection methodology can work for adversarial input detection tasks.

Chapter five briefly detailed all our research findings and how they will be employed to develop an end-to-end filtering scheme. I described the architecture and workflow in this section. Also, compare with other defense methods and gave a complete result analysis.

In chapter six, I concluded my dissertation with a proof of concept application and summary of dissertation.

Chapter 2

Background

In this chapter, I will discuss different adversarial attacks, their properties, filter techniques, adversarial defense techniques, and other terminology used in this proposal.

2.1 Artificial intelligence (AI)

If any machine demonstrates that it can make a decision based on its perception of its environment, then this demonstration is known as Artificial Intelligence[135]. Machine Learning, Natural Language Processing, Evolutionary algorithms, Search algorithm, Mathematical Optimization all considered part of Artificial Intelligence.

2.1.1 Machine Learning (ML)

Machine Learning considered a subpart of the Artificial Intelligence domain. The difference between machine learning and other types of artificial intelligence is that machine learning is data-centric, not decision-based, and it focuses on accuracy rather than success. Machine Learning is the learning in which machines can learn on their own without being explicitly programmed. It is a form of AI that renders the device with the capability to learn and develop from events automatically. It has a self-learning algorithm, and it has extensive relationships with

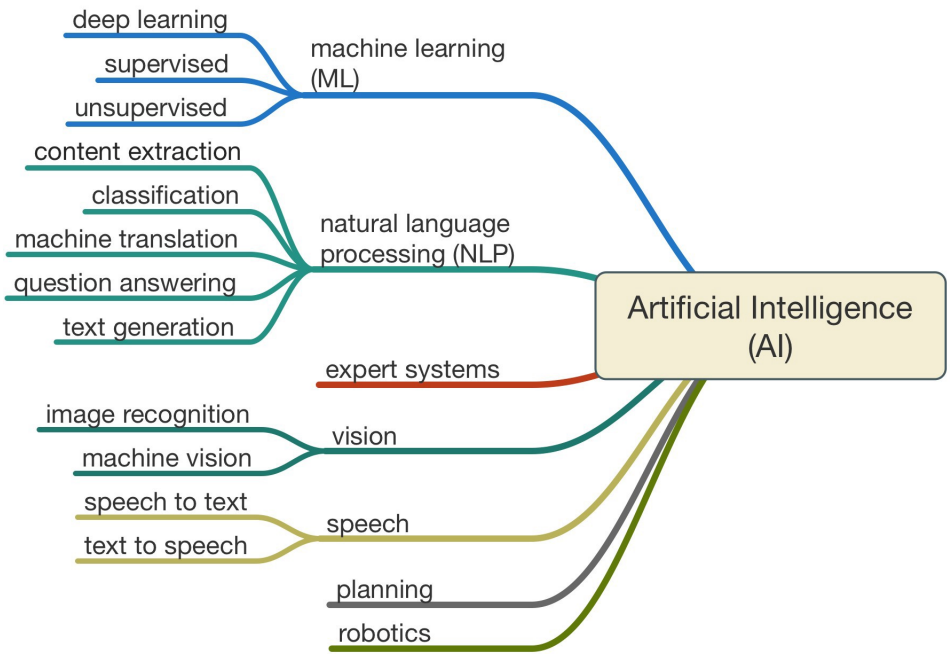


Figure 2.1: AI classification [111]

statistics [134].

2.1.2 Neural Networks (NN)

Neural Networks is a set of the algorithm used to recognize pattern based on numerous iteration of training data which claims to be similar to human brain function. Here initial a supervised dataset used to train a set of nodes. Where nodes have initial weight values are random. By each iteration from input data to output class, error differences affect the weight values after a certain number of repetitions and backpropagation weight values are set in a way that any new input data that weights can forward them to the right output. [89]. In the Figure 2.4, output value is male and female based on age, height and empathy values which a neural network model is classifying the input data. Neural networks uses some activation function which determines what a output node will generate some examples of these functions are TANH, SOFTMAX, RELU etc.

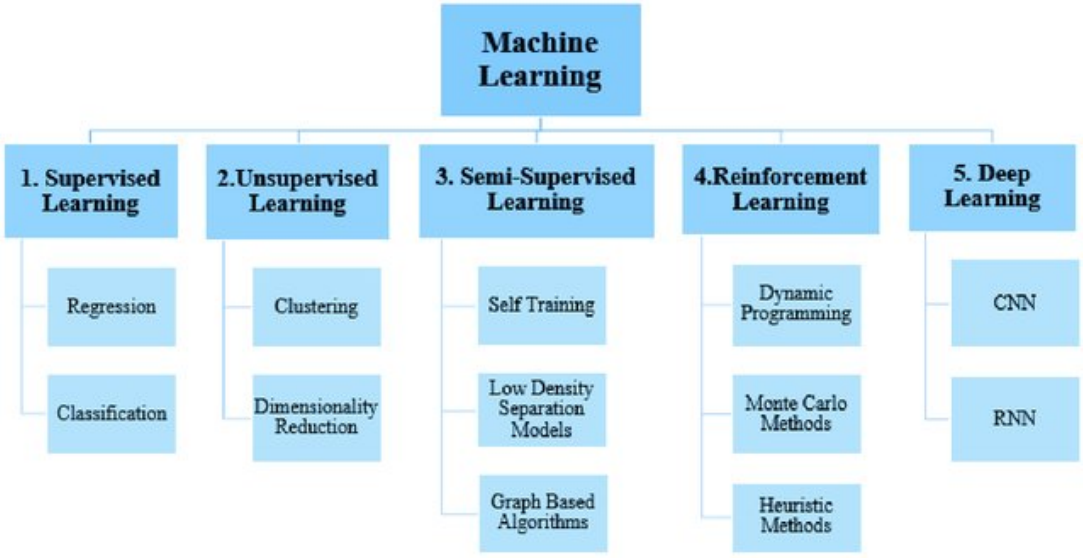


Figure 2.2: ML classification [187]

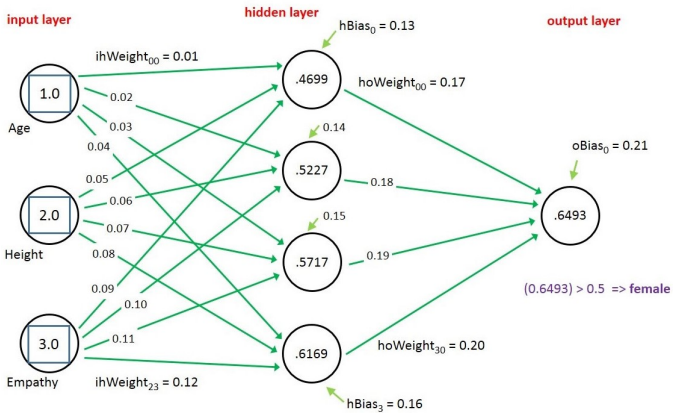


Figure 2.3: Neural Network example [89]

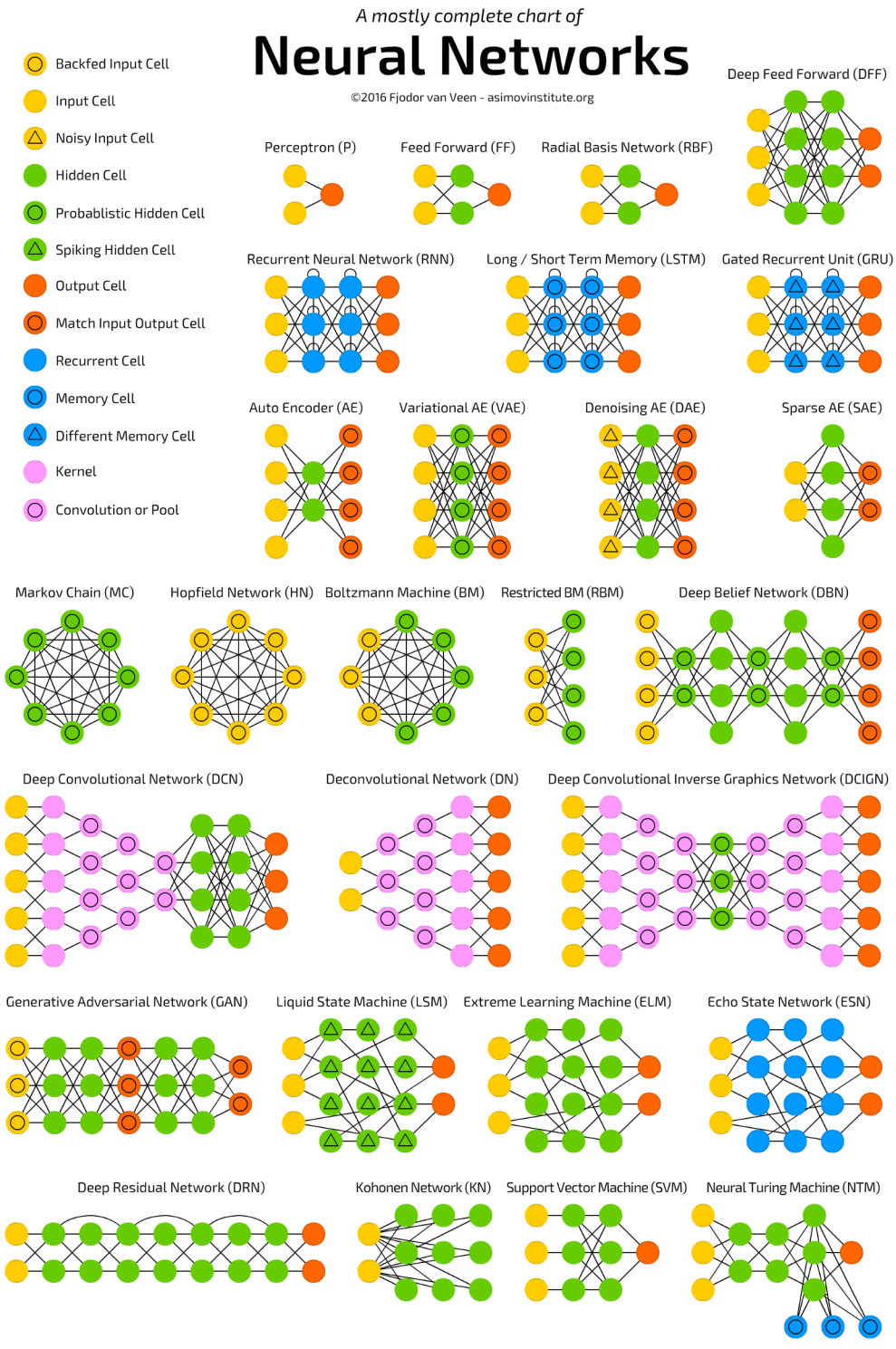


Figure 2.4: Neural Network Classification [193]

2.1.3 Deep Neural Networks (DNN)

Deep Neural Networks is a subset of neural network families that can use a higher level of raw feature data from input to generate output. As an example, to detect a face in an image standard neural network will take information such as histogram data, edge data, number of object data and their positions, etc. etc. But deep learning neural networks can receive a full image as input and can detect the face [115]. Four fundamental deep learning networks are:

- Unsupervised Pre-trained Networks
- Convolutional Neural Networks
- Recurrent Neural Networks
- Recursive Neural Networks

Unsupervised Pre-trained Networks instates a discriminative neural net from one which was prepared utilizing an unaided basis, A Convolutional Neural Network (CNN) is essentially a standard neural system that has been stretched out crosswise over space utilizing shared loads. CNN is intended to perceive pictures by having convolutions inside, which see the edges of an article perceived on the picture. A Recurrent Neural Network(RNN) is fundamentally a standard neural system that has been stretched out crosswise over time by having edges which feed into whenever step rather than into the following layer in a similar time step. RNN is intended to perceive arrangements, for instance, a discourse signal or a content. It has cycles inside that infers the nearness of short memory in the net. A Recursive Neural Network is progressively similar to a various leveled arrange where there is actually no time viewpoint to the information grouping however the info must be handled progressively in a tree design.

2.1.4 Convolutional Neural Network (CNN)

Convolutional neural network is a one type of deep neural network where a set of processing layer added before a fully connected neural network. This convolutional network preporcess the image

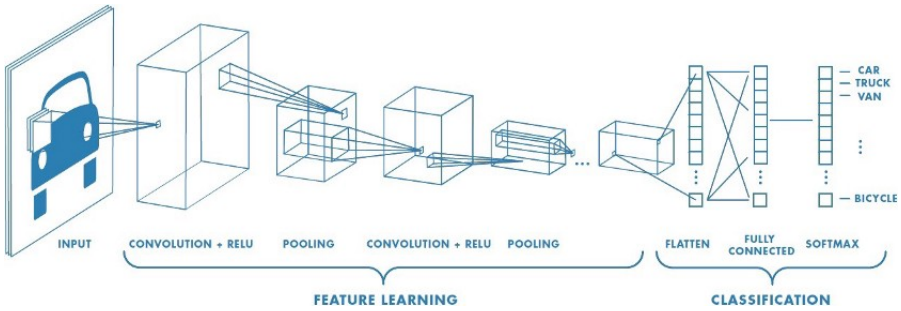


Figure 2.5: A simple convolutional neural network[170]

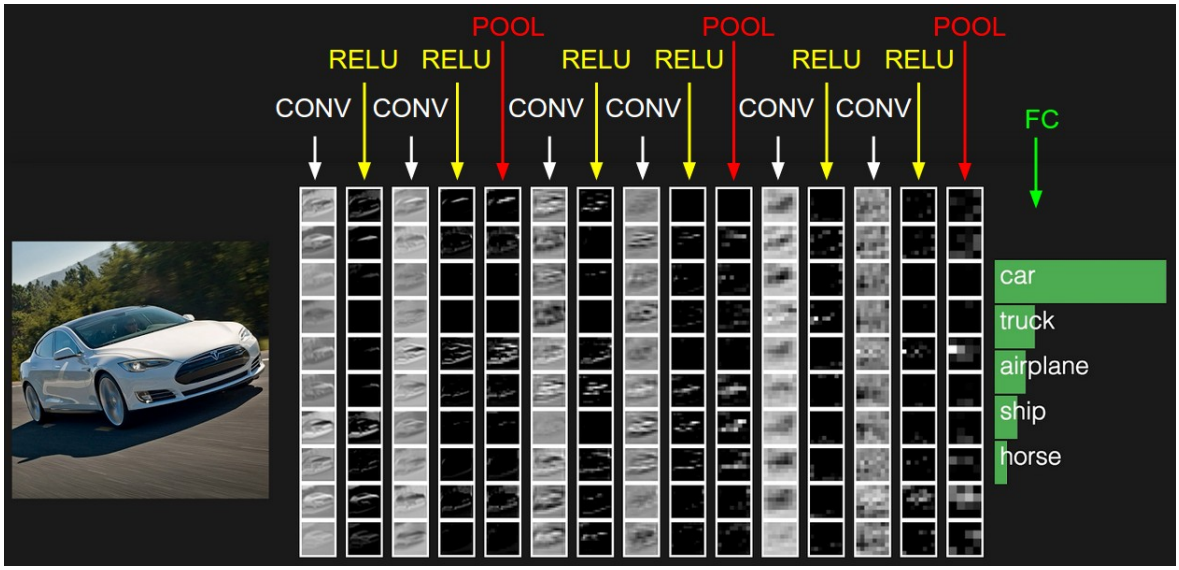


Figure 2.6: A simple example[170]

thus no image feature extraction needed out side this network. in the figure 2.5 we can see an example of basic diagram of a CNN. First layer of CNN is a Convolutional layer, In convolutional layer a matrix know as kernel or filter applied into the image and create an image, next step is pooling step, this step reduce the size of the image, there are max pooling or average pooling and some other types of pooling exist. After several convolutional and pooling step image features get flatten one dimentional input and it applied to a fully connected neural network part. Sometime with convolutional neural network several activation functions are used like Rectified linear Unit, TanH function etc. In the example 2.6 , we can see first an image converted to differnet model than a RELU activation functional changed that after again some pooling applied on it, after sever of this process repetation in the end we get a feature list to give fully connected layer, and that

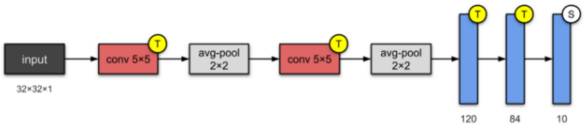


Figure 2.7: A simple Lenet-5[98]

layer give 'car' most confidence than other output class. There are different models of CNN currenly in use, but some of the most famous models are

- LeNet-5
- AlexNet
- VGG-16
- Inception-v1
- Inception-v3
- ResNet-50
- Xception
- Inception-v4
- Inception-ResNets
- ResNeXt-50

We will describe Lenet5, ResNet and VGG-16 and as they are most widely use for adversarial attack research.

LeNet-5

Lenet is most simple of all, it has 2 conv and 3 FC layer. In the figure 2.7 a Lenet-5 has shown.

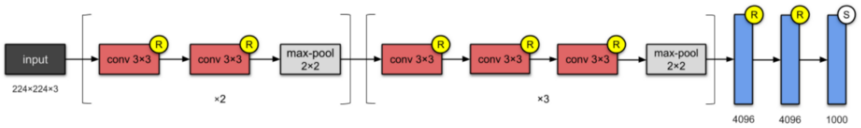


Figure 2.8: A simple VGG-16[98]

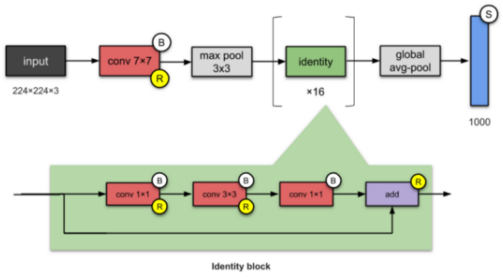


Figure 2.9: A simple Resnet[98]

VGG-16

Visuak Geometry Group develop VGG-16 neural network architecture which has 13 conv and 3 FC, each conv attached with a RELU layer. It use very simple size kernels suc as 3X3 oe 2X2. In the figure 2.8 a VGG-16 has shown.

Resnet

Resnet introduces a identity block which is used with conv and max pool layer before connected to FC layer. Most of the adversarial examlples we going to introduce in next few chapters have been developed on Resnet architecture. . In the figure 2.9 a ResNet has shown.

2.2 Adversarial Machine Learning

Based on NIST [189] definition, adversarial machine learning is the manipulation of training data, ML model architecture, or manipulate testing data in a way that will result in wrong output from ML. The rationale behind AA's success has no conclusive explanation. In 2014, [188, 74] states the reason is non-linearity, but [65] proclaims it is for too much linearity. Another theory by

[190], proposed a tilted boundary theory and insisted that it is never feasible to fit a model completely, and that's why Adversarial attacks exist. Some MIT researchers stated that all adversarial features are not noise, rather these data cannot be properly classified because human sensors are not sophisticated enough to associate a class for these data, however this argument is disputed by other researchers[87].

From the NIST definition, I can define three basic types of AAs as[86]

1. Poisoning attack: In this attack, the attacker can corrupt training data and create adversarial examples later to work on the model. It happens in training time.

2. Evasion attack: In this attack, testing inputs change in a way that they miss-classify to another random or targeted class.

3. Trojan AI attack: In this attack, the AI model's architecture changes in a way it miss-classifies the input.

Generally speaking, adversarial examples are input data which get miss-classified by an AI method but not by a human eye. In mathematical definition:

For a ML model M, if A is Non adversarial input and right class label is C_R, added noise is ϵ, Now,adversarial example $A_x = A + \epsilon$, A_x classify by M as class C_W where $(C_W \neq C_R)$, But if in human eye $A_x \approx A$ and A_x classify as C_R,

Poisoning Attack

In training time, the attacker can corrupt training data and create adversarial examples later to work on the model. In figure 2.10, It is illustrated that malicious training data was given to the model with training data until the desired outcome start to happen. Outilier detection is the most common approach to tackle this attack.

Evasion Attack

In this attack, testing images are change in a way that they miss-classify to another random or targeted class. This attack is transferable to any ML model. It can observe that to classify an object in an image; the neural network doesn't seem able to identify the object features. Such as, the difference between plane and car doesn't depend on the background is sky or ground or which one has wings and which one has large windows. It depends on color pixel values in certain positions.

Figure 2.10: Evasion attack and poisoning attack [86]

This evasion attack can divide into two sides based on knowledge, which is also known as the white-box-black box model as illustrated in figure 2.11. In a white-box attacker knows details of the model in black box attacker only knows the output decision. Also, based on the misclassification class, it can be separated into two types one is targeted, and another is non-targeted. When anyone manipulate an image from one level to sure, another class can be said targeted adversarial attack. And when miss-classify to any other type can be assumed non-targeted attack.

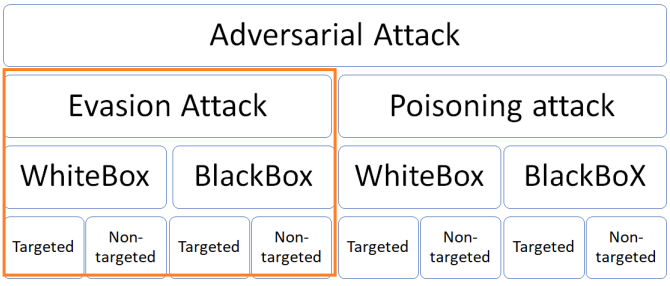


Figure 2.11: Adversarial Attack Types [86]

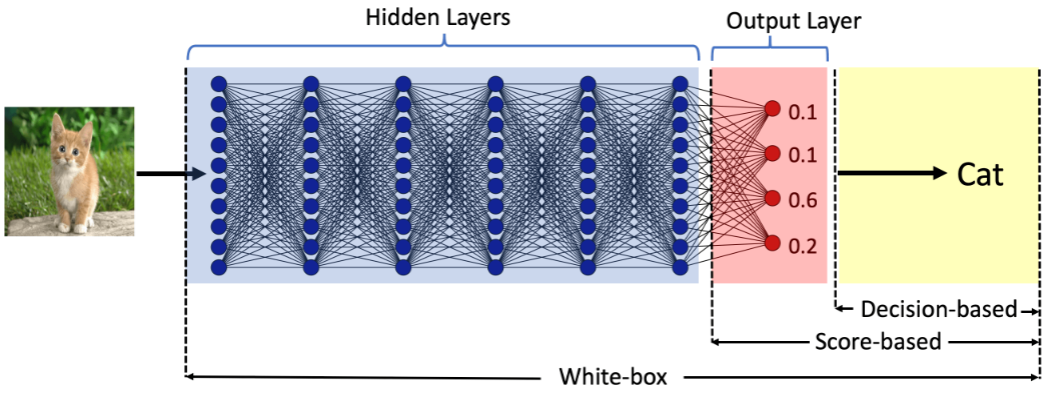


Figure 2.12: Adversarial Attack Generation methods types in whitebox model [35]

Troj AI

If a deep learning network architecture is modified by and adversarial after its training completed, then that deep learning network has a Troj-AI[216]. In figure 2.13, illustrated how a trojan-trigger in the network changing the output of results by manipulating the network. It first gathers information about what should be the input data needed to be a definite class. Based on that it changes the architectural weights. Another One of the Troj-AI examples is Backdoor system [69] In Figure 2.14, It is illustrated how a back door can be added to generate new results.

2.2.1 Adversarial Sample Generation Methods

For a successful attack, attacker need to generate adversarial samples/inputs. Generally speaking, Adversarial samples/Inputs are input data which get miss-classified by an Artificial Intelligence but not by a human eye.

Rauber et al.[162] proposed three basic methods of attack they are gradient-based, score-based, and decision based. In figure 2.15, different types based on their methods is illustrated. In figure 2.12, it is presented that, in the white box model, an attacker can use gradients of neural nodes. For score-based, it only has the output layer class values, and for a decision based on it can only have the final result. In the black-box model, he doesn't have the target ML model to try his attack samples.

Fast Gradient Sign Method (FGSM)[65]	Ground-Truth Attack (GTA) [151]
Iterative Gradient Sign Method (IGSM) [28]	Zero-Query Attacks (ZQA) [151]
Jacobian Saliency Map Attack (JSMA) [154]	Natural Evolution Strategies (NES) [88]
DeepFool (DF) [143]	Boundary Attack (BA) [20]
One-Step Target Class Method (OSTCM) [112]	Greedy Search Algorithm (GSA) [110]
Basic Iterative Method (BIM) [112]	Genetic Attack (GA) [8]
Iterative Least-Likely Class Method (ILLC) [112]	Improved Genetic Algorithm (IGA) [205]
Compositional Pattern-Producing Network-Encoded Evolutionary Algorithm (CPPN EA) [147]	Probability Weighted Word Saliency (PWWS) [164]
Carlini and Wagner's Attack (C&W) [30]	Replacement, Insertion and Removal of Words (RI&RoW) [113]
Zeroth Order Optimization (ZOO) [37]	Real-World Noise (RWN) [113]
Universal Perturbation (UP) [144]	Targeted Audio Adversarial Examples (TAAE) [29]
One Pixel Attack (OPA) [186]	Genetic Algorithms and Gradient Estimation (GA&GE) [192]
Feature Adversary (FA) [168]	HopSkipJumpAttack (HSJ) [35]
Hot/Cold method (H/C) [186]	Backward Pass Differentiable Approximation (BPDA) [32]
Natural GAN (NGAN) [231]	Adversarial Patch Attack (DPATCH)[125]
Model-based Ensembling Attack (MEA) [126]	LaVAN: Localized and Visible Adversarial Noise [99]

Table 2.1: Some Notable Adversarial Attack methods

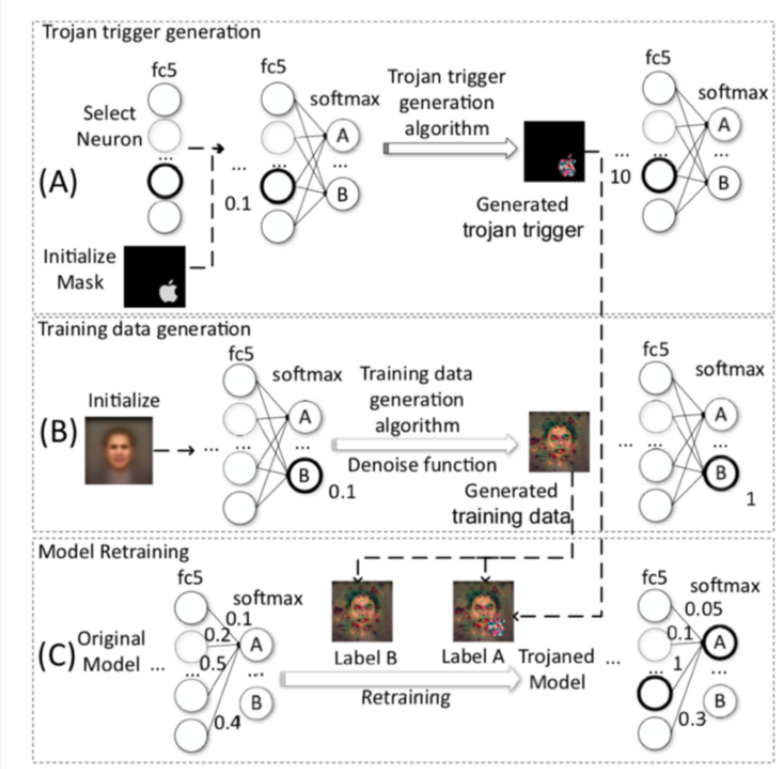


Figure 2.13: Trojan AI[128]

Gradient Based Attacks

In 2014 Fast gradient Sign method was proposed by Ian Goodfellow which is most known Gradient Based Attack [65]. This method computes an adversarial image by adding a pixel-wide perturbation of magnitude in the direction of the gradient. This perturbation is computed with a single step, thus is very efficient in terms of computation time. A simple formulation if Here, $x^{'}$ is the adversarial example that should look similar to x when ϵ is small, and y is the models output. ϵ is a small constant that controls the magnitude of the perturbation, and J denotes the loss function of the model.

$$x^{'} = x + \epsilon \times sign(\Delta_x J(x, y)) \tag{2.1}$$

This attack type is white-box attack, cause attacker need to know the neural network and the gradient value so he can perform back propagation to calculate the derivative of the entropy w.r.t to its input.

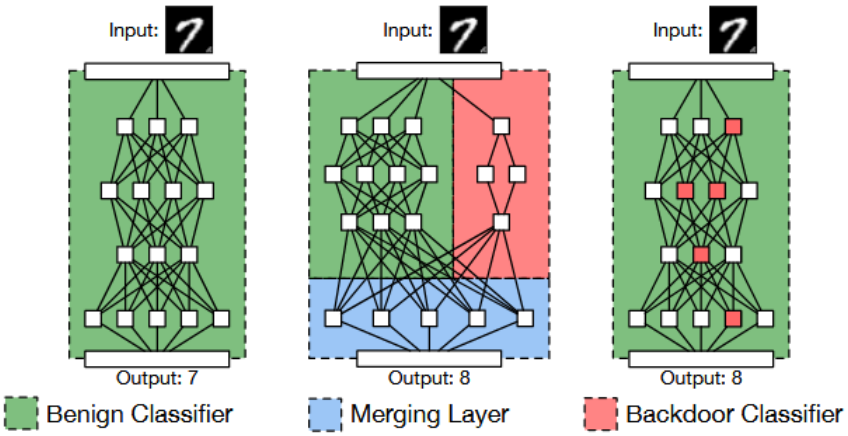


Figure 2.14: Backdoor attack[69]

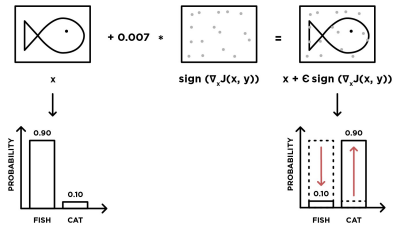


Figure 2.15: Fast sign Gradient method [106]

In figure 2.16, it is illustrated that additive adversarial perturbations based on dL/dx and iterative optimization based attacks. Once dL/dx is calculated (step 1), one may view the attack process as a game where a player (the attacker) can adjust the pixel values (step 2) of the input based on some hints, i.e. the gradient dL/dx, to fool a model (step 3). Another up-gradation of this type is deepfool attack. DeepFool is a simple yet very effective attack [143] . In each iteration it computes for each class $l \neg l_0$ the minimum distance $d(l \neg l_0)$ that it takes to reach the class boundary by approximating the model classifier with a linear classifier. It then makes a corresponding step in the direction of the class with the smallest distance.

Saliency Map Attack

The Jacobian-based Saliency Map Attack is a class of adversarial attack techniques for deceiving classification models. In the digital vision, a saliency map is an image that shows each pixel's

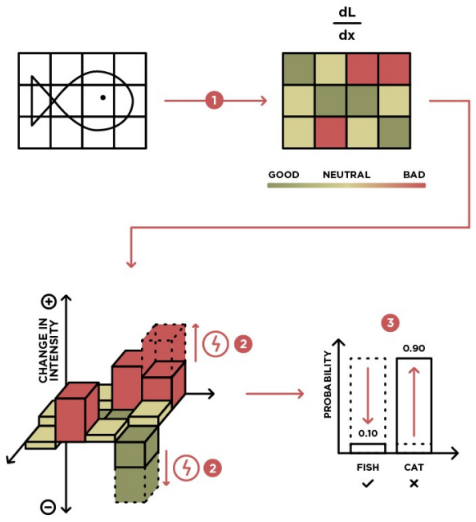


Figure 2.16: Basic Iterative Method [106]

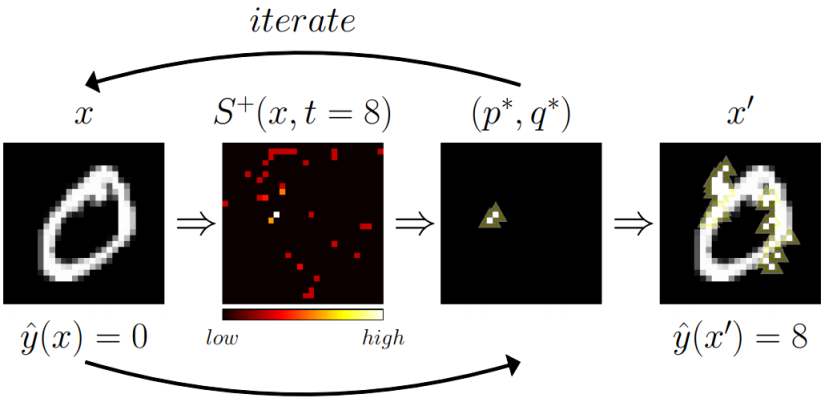


Figure 2.17: JSMA Attack generation [154]

unique quality. The purpose of a saliency map is to simplify and improve the representation of a picture into something more significant and more suitable to analyze. For instance, if a pixel has a high grey level or other unique color quality in a color image, that pixel's class will show in the saliency map and in an obvious way. Saliency is a kind of picture segmentation.

In the figure 2.17, first detect the saliency of image, From the saliency map of the image, it get highest bit positions, As in pictures yellow points are highest, it now increase pixels values arround these point until it got targeted class change.

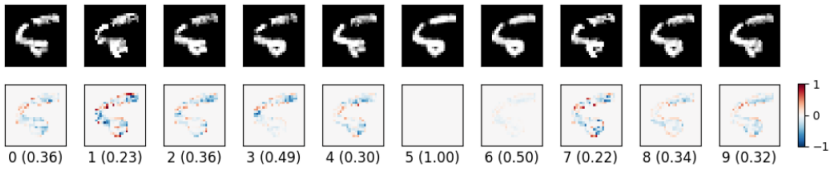


Figure 2.18: CW attack Examples [30]

Optimization based attack

The Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) is a non-linear gradient based numerical optimization algorithm. This method extended in CW method by modifying the loss function. The Carlini and Wagner (CW) (Carlini and Wagner, 2017) method is an iterative attack that constructs adversarial examples by approximately solving the minimization problem [215]. CW is a bit different from the above gradient-based methods in that it is an optimization-based attack. This formulation of the loss function in CW attack can be stated as

$$f(x^{'}) = max(max\{Z(x_i^{'}) : i \neq t\} - Z(x_t^{'}), -k) \tag{2.2}$$

Here, $Z(x^{'})$ denotes the logits (the outputs of a neural network before the softmax layer) when passing adversarial input $(x^{'})$ and t represents the target misclassification label (the label that want the adversary to be misclassified as), while k is a constant that controls the desired confidence score . The intuition for this objective function is to optimize for the distance between the target class t and the most-likely class. If t currently has the highest logit value, then the difference of the logits will be negative, and so the optimization will stop when the logit difference between t and the runner-up class is at most k. In other words, k controls the desired confidence for the adversarial example (e.g. when k is small, the adversarial example generated will be a low confidence adversarial example). On the other hand, if t does not have the highest logit, then minimizing f brings the gap between the highest class' logit and the target class' logit closer together.In figure 2.18 some examples of CW attack presented.

Score Based attack

Score based attack like One-pixel attack works by changing only one pixel in a image[186]. Other usual adversarial images are constructed by perturbing all pixels with an overall constraint on the strength of accumulated modification which they tried to make smaller as possible. But in One pixel or few pixel attack attacker tried to change as much as possible to convert the images to an adversarial image. Here differential evolution (DE) is used, which is a population based optimization algorithm for solving complex multi-modal optimization problems . At first , Encode the perturbation into an array which is optimized (evolved) by differential evolution. One candidate solution contains a fixed number of perturbations and each perturbation is a tuple holding five elements: x-y coordinates and RGB value of the perturbation. One perturbation modifies one pixel. The initial number of candidate solutions (population) is 400 and at each iteration another 400 candidate solutions (children) will be produced by using the usual DE formula:

$$x_i(g+1) = x_{r1}(g) + F(x_{r2}(g) - x_{r3}(g)) \tag{2.3}$$

$here, r1 \neq r2 \neq r3$, where xi is an element of the candidate solution, r1, r2, r3 are random numbers, F is the scale parameter set to be 0.5, g is the current index of generation. Once generated, each candidate solution compete with their corresponding parents according to the index of the population and the winner survive for next iteration. Here fitness function is simply the probabilistic label of the target class .

Decision Based Attack

Boundary attack is one of the decsion based adversarial attack, it initialized from a point that is already adversarial and then performs a random walk between the adversarial and non-adversarial region in a way that it fill up below criteria,

- (1) It stays in the adversarial region.
- (1) Distance between two image is reduced [20].

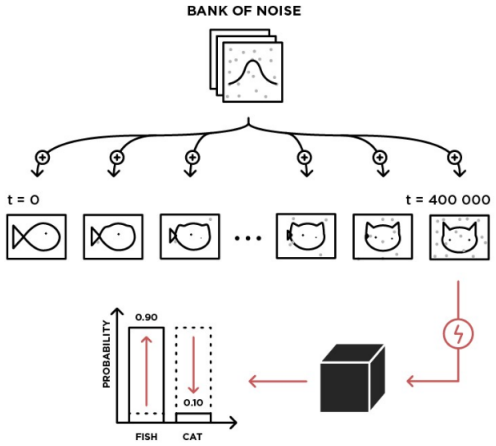


Figure 2.19: Boundary Based Attack

In figure 2.19, it is illustrated how a targeted boundary attack generate an adversarial example only by keep adding noise 'n' sampled from some predefined noise to a benign example until the image looks like another image from a different class, while still be classified as the true class of the original image.

Other notable attacks

Universal Perturbation (UP) is a universal image-agnostic perturbation attack method that fools classifiers by single adversarial perturbation to all images.[144]. Feature Adversary (FA) method minimizes the distance of the representation of internal neural network layers instead of the output layer to produce AA[168]. Hot/Cold method (H/C) method finds multiple AA for every single image input. It first aligns the modified image with the original image (cold) and then measure the similarity between the perturbed image (hot)[186].Targeted Audio Adversarial Examples (TAAE) is an iterative optimization-based targeted attack to a state-of-the-art speech-to-text transcription neural network via optimization based on the MFC pre-processing transformation[29].Zeroth Order Optimization (ZOO) another attack that does not require gradients and utilizes hinge like loss function and symmetric difference quotient to generate AA[37]. Natural GAN (NGAN) utilizes generative adversarial networks (GANs) that minimizes the distance of the inner

representations to generate AAs.[231] . Genetic Attack (GA)exploits population-based gradient free optimization via genetic algorithms to replace words with their synonyms so as to generate semantically and syntactically similar AAs[8] . Improved Genetic Algorithm (IGA) procedure adopts the genetic metaheuristic for synonyms substitution to attain AAs[205]. Replacement, Insertion and Removal of Words (RI and RoW) is an iterative method that combines three different kinds of modifications to alter a regular input into an AA by replacement, insertion and removal of words into the text[113]. Real-World Noise (RWN) technique adds real-world scenario noises such as café, meeting, and station to generate AAs[113]. Genetic Algorithms and Gradient Estimation (GA and GE)combines genetic algorithms and gradient estimation to construct AAs. The attack is first carried out by gradient-free genetic algorithms, then gradient estimation is utilized to determine careful noise placement[192].

2.2.2 Defense Against Adversarial Attacks

There are mainly two kinds of way when making defence against adversarial samples, one is Proactive and another is Reactive. Reactive is detecting the adversarial example before it enter in ML models, another is make ML model better so it can identify the right class of the adversarial sample from targeted class[197].

Defense techniques against adversarial methods can be summarized in three types

- Denoising strategy or Gradient masking : Try to remove the distortions of the image.
- Basic adversarial training : Train the neural network with adversarial example
- Ensembling methods : Add multiple neural network with transformed dataset to combine a majority result

Prepare training data for a machine learning model need to done using careful consideration and examined process. As the accuracy of machine learning models depends much on the quality of training data, It is must to train the machine learning models with filtered data. There were many processes invented by the researcher to filter out training data before train the models. Wilson is

Defense Method	Approach/Scheme
Training	Ensemble Adversarial Training, a training methodology that incorporates perturbed inputs transferred from other pre-trained models [196]
	Extended adversarial and virtual adversarial training as a means of regularizing a text classifier by stabilizing the classification function [138]
	Training the state-of-the-art speech emotion recognition on the mixture of clean and adversarial examples to help regularization [29]
Distillation	The main idea used is training the model twice, initially using the one-hot ground truth labels but ultimately using the initial model's probability as outputs to enhance robustness [152][180]
Pre-Processing	Using PCA, low-pass filtering, JPEG compression, soft thresholding techniques as pre-processing technique to improve robustness [176]
	Use of use two randomisation operations: (1) random resizing of input images and (2) random padding with zeros around the input images [214]
Structure modify	Synonyms encoding method that inserts an encoder before the input layer of the model and then trains the model to eliminate adversarial perturbations [205]
	An architecture using Bayesian classifiers (Gaussian processes with RBF kernels) to build more robust neural networks [18]
Network verify	A verification algorithm for DNNs with ReLU function was proposed in [100] verified the neural networks utilizing Satisfiability Modulo Theory (SMT) solver
	The method in [100] was modified in $max(x, y) = ReLU(x - y) + y$ and $\|\|x\|\| = ReLu(2x) - x$ to reduce the computational time
Ensembling	The proposed strategy used an ensemble of classifiers with weighted/unweighted average of their prediction to increase robustness against attacks [185]
	A probabilistic ensemble framework against adversarial examples that capitalizes on intrinsic depth properties (e.g., probability divergence) of DNNs [1]
Detection	First, the features are squeezed either by decreasing each pixel's color bit depth or smoothing the sample using a spatial filter. Then, a binary classifier that uses as features the predictions of a target model before and after squeezing of the input sample [215]
	A framework that utilizes ten nonintrusive image quality features to distinguish between legitimate and AA samples [5]
	Multiversion Programming based an audio AE detection approach, which utilizes multiple off-the-shelf Automatic Speech Recognition systems to determine whether an audio input is an AE [221]

Table 2.2: Summary of countermeasures against adversarial examples[47].

first researcher, who in 1972 tried to add K-NN filtering for training data for use in neural network training [207]. Later in 1976, Tomek [206], improved Wilson's work when he able to correlate K with neural network efficiency. Until 2000, most of the filter research work is to prune the data or detect the noise. Some notable works come from Hansen and Salamon [79] in 1990

when they used an ensemble classier which detects mislabeled instances by constructing a set of base-level detectors (classifiers) and then using their classification errors to identify mislabeled examples. Next year, Aha, Kibler, and Albert [166] looked at the training data and tried to identify dominant instances to rule out noisy data. In 1992, Srinibason, Muggleton, bain tried an information theocratic approach to differentiate noise and exception [183]. Next few years some more technique developed by researchers, Zaho and Nishida used fuzzy logic in 1995 [227], same year Dietterich and bakiri [52] [53] developed a method for learning classifiers for multiple classes in which error-correcting output codes are employed as a distributed output representation. Another notable is in 1996, Gamberger, lavarc, Dzeroski tried to detect inconsistent example in training data using a user set threshold [61]. In 1999 Brodley and Friedl able to Combine filtering and voting approach which provides excellent results [23].

In 2005, A Angelova, Y Abu-Mostafam used Pruning Training Sets for Learning of Object Categories they applied to bootstrap and Naïve Bayes algorithm [9]. Michael Brückner and Tobias Scheffer use of game theory in 2011 also shows a diverse approach in developing input filters [25]. In 2015, Ian J Goodfellow tried to training on adversarial inputs pro-actively [65], Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami performed defensive distillation [151] and T. Miyato, S.-i. Maeda, M. Koyama, K. Nakae, and S. Ishii training the network with enhanced training data all to create a protection against adversarial example[139].

In 2017, Grosse et el. did statistical tests using a complementary approach to identify specific inputs, that are adversarial [68]. Wong et el. showed convex outer adversarial polytope can be a proven defense [211]. Lu et el. (2017) checked whether the depth map is consistent or not (only for image) to detect adversarial examples [129]. Metzen et el. implemented deep neural networks with a small "detector" sub-network were trained on the binary classification task of distinguishing factual data from data containing adversarial perturbations [137]. The same year, Madry et el. (2017) published a paper on adversarial robustness of neural networks through the lens of robust optimization [132]. Chen et al. tried to devise adversarial examples with another guardian neural net distillation as a defense from adversarial attacks[36]. In 2018, Wu et al.

developed Highly Confident Near Neighbor (HCNN), a framework that combines confidence information and nearest neighbor search, to reinforce adversarial robustness of a base model[212]. Also in 2018, Paudice et al. applied Anamoly Detection[155] and Zhang et al. detected adversarial examples by identifying significant pixels for prediction which only work for images [224]. Other researchers such as Wang et al. tried with mutation testing [204] and Zhao et al. developed key-based network, a new detection-based defense mechanism to distinguish adversarial examples from normal ones based on error correcting output codes, using the binary code vectors produced by multiple binary classifiers applied to randomly chosen label-sets as signatures to match standard images and reject adversarial examples [226]. Later that year Liu et al. tried to use steganalysis[124] and Katzir et al. implemented a filter by constructing euclidean spaces out of the activation values of each of the deep neural network layers with k-nearest neighbor classifiers (k-NN) [101]. A different notable strategy was taken by researchers Pang et al. They used thresholding approach as the detector to filter out adversarial examples for reliable predictions[150]. For an image classification problem, Tian et al. did image transformation operations such as rotation and shifting to detect adversarial examples[194] and Xu et al.[215] simply reduced the feature space to protect against adversary. In 2019, Monteiro et al [141] developed inputfiler which is based on Bi-model Decision Mismatch of image. Sumanth Dathathri showed whether prediction behavior is consistent with a set of fingerprints (a data set of NN) named NFP method [51]. Same year, Crecchi et al. used non-linear dimensionality reduction and density estimation techniques [44] and Aigrain et al. tried to use confidence value in CNN[4]. Some other notable works in that year were meta-learning based robust detection method to detect new adversarial attacks with limited examples developed by Ma et al. [131]. Another important and effective work was done by Chen et al., where they tried to keep the records of query and used KNN to co-relate that with adversarial examples [38] In some adversarial defense techniques well-known robust recognition models are trained on adversarial inputs proactively [65], performing defensive distillation [151], training the network with enhanced training data all to create a protection against adversarial example [139].To detect adversarial inputs Image

histogram-based [158] methods are also used. In 2018 Akhter et el [6] proposed an adversarial attack detection scheme based on image quality related features to detect various adversarial attacks. In 2017, Carlini et al. [28] tested ten defense techniques, by detailed evaluation they showed that pre-processing techniques can be easily bypassed.

2.3 Preliminaries

2.3.1 Metrics

Signal to Noise (SNR)

SNR is frequently defined as the ratio of the signal power and the noise power [217]. For an image calculate the ϕ_{signal} as the mean of pixel values. calculate the ϕ_{noise} and the standard deviation or error value of the pixel values. from these derive the below equation for SNR.

$$SNR = 10log10(\phi_{signal}/\phi_{noise}) \tag{2.4}$$

to express the result in decibel.

Peak Signal Noise Ratio(PSNR) and Root Mean Sqaure Error (RMSE)

Given a noise-free $m \times n$monochrome image I and its noisy approximation K Mean squared error(MSE) is defined as:

$$MSE = \frac{1}{m\,n}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2 \tag{2.5}$$

So $RMSE = \sqrt{MSE}$ And PSNR is

$$PSNR = 20 \cdot \log_{10}(MAX_I) - 10 \cdot \log_{10}(MSE) \tag{2.6}$$

Here, MAX_I is the maximum possible pixel value of the image. When the pixels are represented using 8 bits per sample, this is 255. More generally, when samples are represented using linear

Mean Absolute Error (MAE)

It calculate for difference between two images with same size. For image X and Y all pixel, Equation is

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} y_i - x_i}{n} \tag{2.7}$$

Histogram

Histogram represents the distribution of each color in the image [96][34]. Histogram doesn't concern about shape, size or any attribute rather than color distribution of an image. Calculation of histogram has color range from x axis and y axis has the number of pixel with that color. For a grayscale image I with intensity values in the range $I(u, v) \in [0, K-1]$ holds exactly K entries, where $K = 28 = 256$ for a typical 8-bit grayscale image. Each single histogram entry is defined as

$h(i)$ =the number of pixels in I with the intensity value i,

for all $0 \leq i < K$. More formally stated,

$$h(i) = card(u, v)|I(u, v) = i[26] \tag{2.8}$$

Therefore, h(0) is the number of pixels with the value 0, h(1) the number of pixels with the value 1, and so forth. Finally, h(255) is the number of all white pixels with the maximum intensity value 255 = K-1. The result of the histogram computation is a 1D vector h of length K.

Local Binary patter(LBP)

Local binary patterns (LBP) is a type of visual descriptor[70]. $LBPP_{u2}$ The subscript represents using the operator in a (P, R) neighborhood. $u2$ stands for using only uniform patterns and

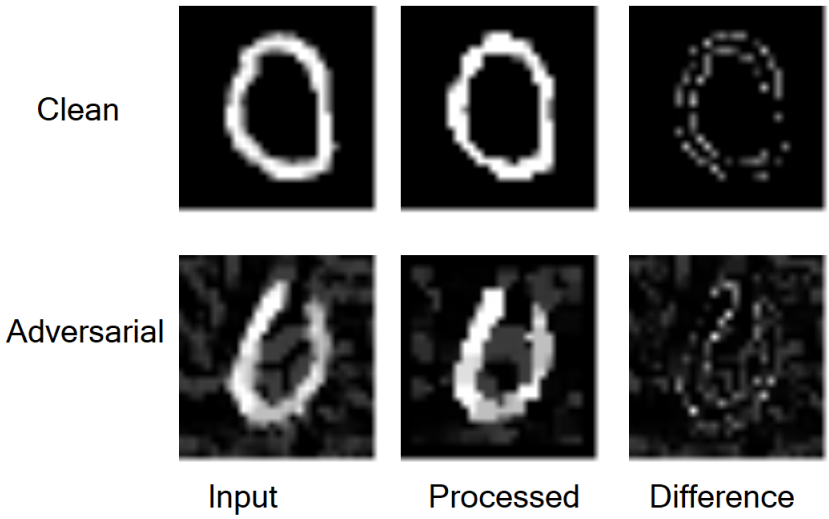


Figure 2.20: Adaptive filtering

labeling all remaining patterns with a single label. After the LBP labeled image $fl(x, y)$ has been obtained, the LBP histogram can be defined as

$$H_i = \sum_{x,y} I\left\{f_l(x, y) = i\right\}, i = 0, \ldots, n-1, \tag{2.9}$$

in which n is the number of different labels produced by the LBP operator, and IA is 1 if A is true and 0 if A is false.

2.3.2 Filter Techniques

Smoothing

Smoothing is an Edge Detection Technique Using the Facet Model and Parameterized Relaxation Labeling [233]. Figure 2.20 shows an implementation of adaptive smoothing filtering. The filter is aimed to perform image smoothing, but keeping sharp edges. This makes it applicable to additive noise removal and smoothing objects' interiors, but not applicable for spikes (salt and pepper noise) removal.

suppose a $(2n+1)(2n+1)$ (where n is an integer) matrix, pixel point represent by i,j the

mean color represent by c(i,J). so sliding window can be estimated as:

$$c(i,j) = \frac{1}{2n+1^2} \sum_{k=i-n}^{i+n} \sum_{l=j-n}^{j+n} c(k,l) \tag{2.10}$$

for left

$$c(i,j) = \frac{1}{2n+1n+1} \sum_{k=i-n}^{i+n} \sum_{l=i-j}^{j} c(k,l) \tag{2.11}$$

for right

$$c(i,j) = \frac{1}{2n+1n+1} \sum_{k=i-n}^{i+n} \sum_{l=j}^{j+n} c(k,l) \tag{2.12}$$

for up

$$c(i,j) = \frac{1}{2n+1n+1} \sum_{k=i-n}^{i} \sum_{l=j-n}^{j+n} c(k,l) \tag{2.13}$$

for down

$$c(i,j) = \frac{1}{2n+1n+1} \sum_{k=i}^{i+n} \sum_{l=j-n}^{j+n} c(k,l) \tag{2.14}$$

The next calculations are done for each pixel:

weights are calculate for 9 pixels - pixel itself and 8 neighbors:

$$w(x,y) = 1 - exp(\frac{(Gx^2 + Gy^2)}{(2 \times factor^2)}) \tag{2.15}$$

$$Gx(x,y) = \frac{(I(x+1,y) - I(x-1,y))}{2} \tag{2.16}$$

$$Gy(x,y) = \frac{(I(x,y+1) - I(x,y-1))}{2} \tag{2.17}$$

, where factor is a configurable value determining smoothing's quality. sum of 9 weights is calclated (weightTotal);

sum of 9 weighted pixel values is calculatd (total);

destination pixel is calculated as $\frac{total}{weight}$ Total.

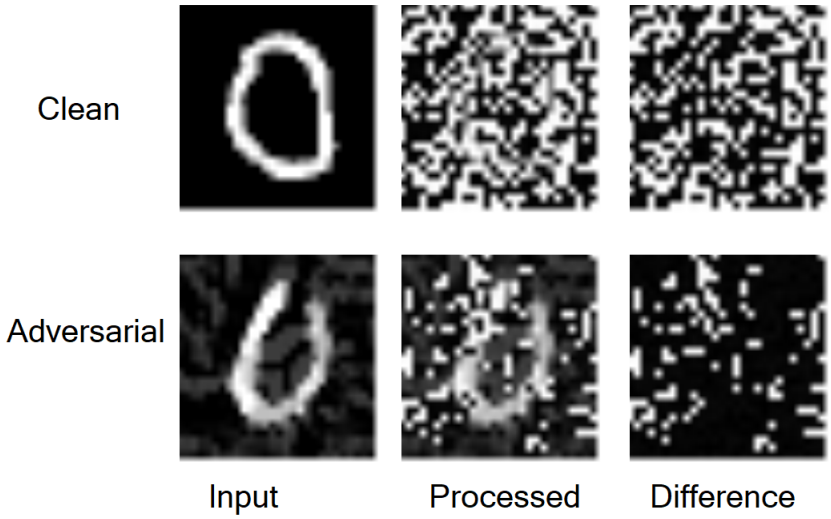


Figure 2.21: Additive Noise

Noise Adding filters

Additive white Gaussian noise (AWGN) is a basic noise model used in Information theory to mimic the effect of many random processes that occur in nature [103].

More precisely, for an image adaptive noise add the extra pixel value in the position based on the neighbor hood pixel distribution. So for a image matrix like a =

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

after added noise image matrix will be

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

In additive noise filter, added random noise in the image as an example in figure 2.21

Thinning (TN)

"Thinning is a morphological operation that is used to remove selected foreground pixels from binary images, somewhat like erosion or opening. It can be used for several applications, but is particularly useful for skeletonization[201]. In this mode it is commonly used to tidy up the output of edge detectors by reducing all lines to single pixel thickness. Thinning is normally only applied to binary images, and produces another binary image as output."

Let $E = Z^2 E = Z^2$,

and consider the eight composite structuring elements, composed by:

$C_1 = \{(0,0), (-1,-1), (0,-1), (1,-1)\}$

$D_1 = \{(-1,1), (0,1), (1,1)\}$

$C_2 = \{(-1,0), (0,0), (-1,-1), (0,-1)\} and$

$D_2 = \{(0,1), (1,1), (1,0)\}$

and the three rotations of each by$90^o 90^o$, $180^o 180^o$, and $270^o 270^o$.The corresponding composite structuring elements are denoted $B_1, \ldots, B_8 B_1, \ldots, B_8$.

For any i between 1 and 8, and any binary image X, define

$X \otimes B_i = X \setminus (X \odot B_i) X \otimes B_i = X \setminus (X \odot B_i)$,

where $\setminus \setminus denotes the set - theoretical difference and \odot \odot$ denotes the hit-or-miss transform.

The thinning of an image A is obtained by cyclically iterating until convergence:

$A \otimes B_1 \otimes B_2 \otimes \ldots \otimes B_8 \otimes B_1 \otimes B_2 \otimes \ldots A \otimes B_1 \otimes B_2 \otimes \ldots \otimes B_8 \otimes B_1 \otimes B_2 \otimes \ldots$.

Sharpening

Sharpening is to enhance line structures or other details in an image[201]. here, Enhanced image = original image + scaled version of the line structures and edges in the image. Line structures and edges can be obtained by applying a difference operator (=high pass filter) on the image. Combined operation is still a weighted averaging operation, but some weights can be negative, and the sum=1. In frequency domain, the filter has the “high-emphasis” character.

Gaussian blur (GS)

Gaussian blur or Gaussian smoothing is technique to reduce the image quality. In reduced image quality the noise also get reduced much more than non-noise pixels. Convolving image with Gaussian function is way to perform this operation. Gaussian blur known as low pass filter as it reduce the images higher frequency components[203]. The Gaussian normal distribution equation for two dimension is

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \tag{2.18}$$

Acronym	FullMeaning	Acronym	Full Meaning
BS	Bilateral smoothing	BPDA	BackPass Differential Approximation
AS	Adaptive Smoothing	PGD	Projected Gradient disent
AN	Additive Noise	BIM	Basic Iterative method
FGSM	First Gradient Sign Method	MBIM	Momentum Basic Iterative Method
JSMA	Jacob Saliency Map Method	SIPTS	Set of Image Processing Technique Sequence
DF	Deep Fool method	DI	Pixel Difference between before and after IPTS applied
HSJ	HopSkipJump	ED	Euclidean distance
MLM	Machine Learning Model	LBP	Local binary pattern
IPT	Image Processing Techniques	PDE	Probability Density Equation
IPTS	Image Processing Technique Sequence	TN	Thining
PX	Pixellate	GS	Grey-scaled

Table 2.3: Acronym used in this proposal

Wavelet Transform

Wavelets are data series which starts at 0 , increases and decrease to 0 again. This Data series can be converted to square integral function. In discrete wavelet transform , wavelets are randomly sampled[10]. For an input represented by a list of 2^n numbers, the Haar wavelet transform pair up input values, storing the difference and passing the sum. This process is repeated recursively, pairing up the sums to prove the next scale, which leads to $2^n - 1$ differences and a final sum.

If a matrix A is: $H_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}$

Its Haar transform will be:

$H_4 = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{bmatrix}$

Distance Transform

Transforming the pixel in the image according to perspective distances of the corresponding objects in the image is distance transform[21]. Signed distance functions definitions:

If Ω is a subset of a metric space, "X", with metric, "d", then the "signed distance function", "f", is defined by :$f(x) = \begin{cases} d(x, \partial\Omega) & \text{if } x \in \Omega \\ -d(x, \partial\Omega) & \text{if } x \in \Omega^c \end{cases}$

where $\partial\Omega$ denotes the boundary ofΩ. For any $x \in X$,

: $d(x, \partial\Omega) := \inf_{y \in \partial\Omega} d(x, y)$

where "inf" denotes the infimum.

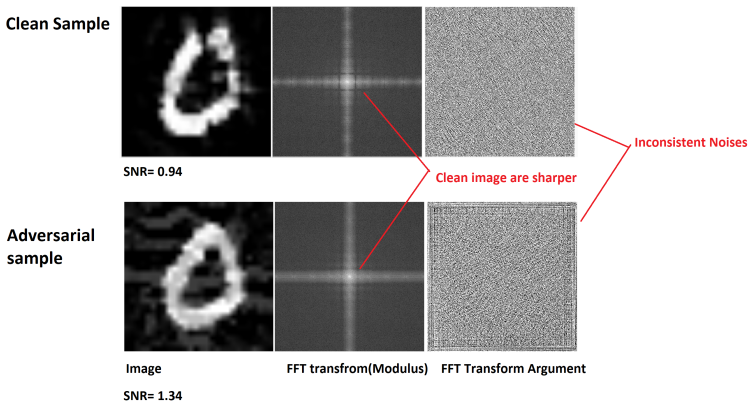


Figure 2.22: Fourier transformation effect

Fourier transformation

Fourier transformation converts any signal form to sinusoidal form [159][123] . Fourier transform denoted as :

$$F(s) \equiv \int_{-\infty}^{\infty} f(x)\, e^{-2\pi i s x}\, dx \ , \tag{2.19}$$

log-polar Transform

The log-polar transform is performed by remapping points from the 2D Cartesian coordinate system (x,y) to the 2D log-polar coordinate system (σ, θ)

$$\sigma = \log \sqrt{(x - x_c)^2 + (y - y_c)^2} \tag{2.20}$$

$$\theta = a \tanh 2((y - y_c), (x - x_c)) \tag{2.21}$$

where σ is the logarithm of the distance of a given point, (x,y), in the image from the centre, (x_c, y_c), and θ is the angle of the line through the point and the centre[210].

Census Transform

The census transform (CT) is an image operator that associates to each pixel of a grayscale image a binary string, encoding whether the pixel has smaller intensity than each of its neighbours, one

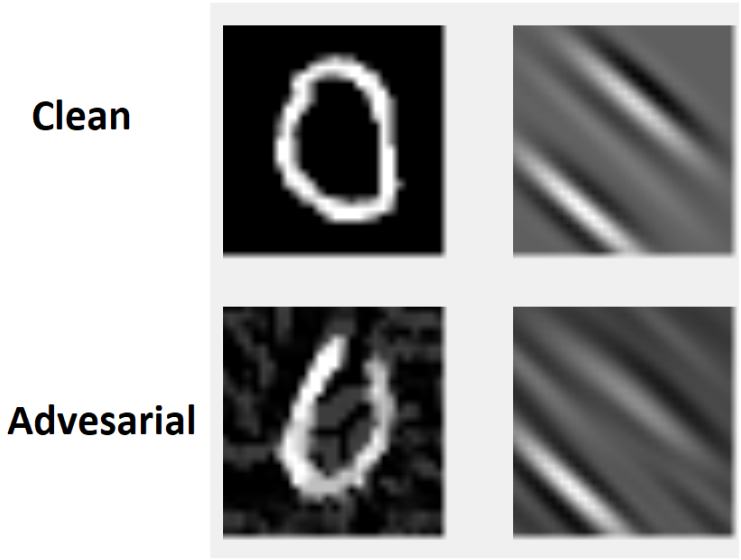


Figure 2.23: Gabor transformation effect

for each bit[184].

The most common version of the census transform uses a 3x3 window, comparing each pixel p with all its 8-connected neighbours with a function ξ defined as

$$\xi(p, p') = \begin{cases} 0 & \text{if } p > p' \\ & \\ 1 & \text{if } p \leq p' \end{cases}. \tag{2.22}$$

The results of these comparisons are concatenated and the value of the transform is an 8-bit value, that can be easily encoded in a byte.

$$\begin{pmatrix} 124 & 74 & 32 \\ 124 & 64 & 18 \\ 157 & 116 & 84 \end{pmatrix} \longrightarrow \begin{pmatrix} 1 & 1 & 0 \\ 1 & x & 0 \\ 1 & 1 & 1 \end{pmatrix} \longrightarrow 11010111 \tag{2.23}$$

Gabor/Morlet wavelet transform

Gabor weblet is a wavelet composed of a complex exponential (carrier) multiplied by a Gaussian window[179]. Its equation

$$\Psi_\sigma(t) = c_\sigma \pi^{-\frac{1}{4}} e^{-\frac{1}{2}t^2}(e^{i\sigma t} - \kappa_\sigma) \tag{2.24}$$

where $\kappa_\sigma = e^{-\frac{1}{2}\sigma^2}$ is defined by the admissibility criterion, and the normalisation constant c_σ is:

$$c_\sigma = \left(1 + e^{-\sigma^2} - 2e^{-\frac{3}{4}\sigma^2}\right)^{-\frac{1}{2}} \tag{2.25}$$

The Fourier transform of the Morlet wavelet is:

$$\hat{\Psi}_\sigma(\omega) = c_\sigma \pi^{-\frac{1}{4}} \left(e^{-\frac{1}{2}(\sigma-\omega)^2} - \kappa_\sigma e^{-\frac{1}{2}\omega^2}\right) \tag{2.26}$$

2.3.3 Genetic Algorithm (GA)

GA is an evolutionary heuristic search algorithm. It is a population-based algorithm that helps find the best result using biological phenomena such as reproduction, mutation, recombination, and selection. Genetic Programming, Gene Expression Programming, and the Strength Pareto Evolutionary Algorithm are prominent examples of GAs. Genetic programming mainly works by encoded computer problem solutions pool known as the population to gene like structure and try to find out the best solution by using different evolutionary techniques. At first, a set of random solutions is generated from all possible solutions in representation space. Then an evaluation of these solutions based on some measures is performed. Based on these measures, the best solutions are the crossover and mutation to generate a new set of solutions. The same procedure is iterated repeatedly until a near-optimal solution is found or the termination condition is met. During this process, an answer with the best measures is selected as a result. The set of genes creates individuals as their solution. The collection of individuals is known as population. When two individuals create a new individual, it is known as a crossover operation. When a bit of an

individual is changed, it is known as a mutation in the GA. GA has been used for Computer Vision (regenerate images [71]), Computational AI domain (searching recommendations[78][169],Cybersecurity Domain (smart grid management [174]),and many other domains.

2.3.4 Negative Selection Algorithm

Immunological Computation a.k.a. Artificial Immune System (AIS) is inspired by the human immune system (HIS) mechanism and utilizes to solve computational problems[48]. One of the fundamental aspects of the HIS is self/non-self discrimination. The Human immune system can identify which cells are own (self) and can differentiate foreign entities (non-self)[202]. Therefore, it can strengthen its defense versus the adversarial rather of hurting the self cell. The most popular AIS research methods include the Negative Selection Algorithm (NSA), clonal selection, immune network theory, danger theory, and positive selection [49]. The NSA is one of the most studied and researched algorithms, particularly for anomaly detection[93, 72]. In 1994 [59], introduced NSA for solving computer security problems.

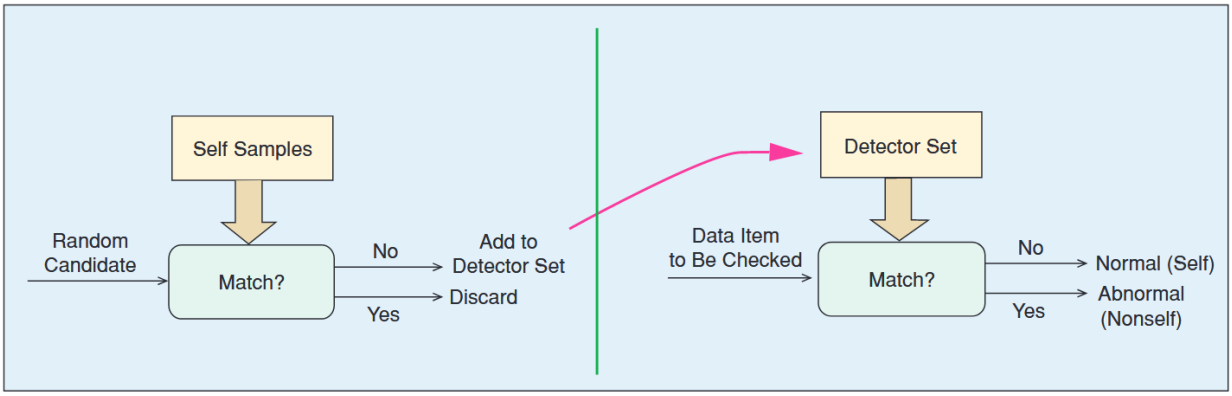


Figure 2.24: The basic Negative Selection Algorithm (NSA) [45] similar to any two-phase supervised learning algorithms. The left diagram shows detector generation in the complementary space (training phase) and the right illustrates the use of detectors (testing phase).

Basic concepts of NSA

Negative Selection Algorithm (NSA) uses complementary representation space of positive data features or profile. The basic concept of NSA is illustrated in figure 2.24, where negative detectors are generated in the complementary space. Here the positive data are the features of training/clean/normal data which may be represented in real, binary, or strings. Given this positive profile, an NSA can generate detectors/filters/clusters in negative or complementary space. So, these detectors must not match any clean/self data sample, and can be used to classify input data as adversarial or not (similar to self and non-self-discrimination). Different matching methods were used to measure the distance between the self and non-self in binary representation of feature space including Euclidean distance, Manhattan distance, R-bit chunk matching, Hamming Distance, etc. Also different variants of NSA have been used many application domains and demonstrated as advantageous in one class classification problems, outlier detection, fault detection, intrusion/anomaly detection[3, 93], etc.

NSA Terminology

- Self: Representation of a data class. NSA will identify a given data is from self class or not. This self data can be a set of real values or a set binary value, or a string.
- Detector: A set of data which matched with non-self data.
- Distance Measure/Matching rules: The formula/method used to measure the distance between two data points in representation space. Commonly in the NSA, it was used to measure the detector's distance from a data point. It is also known as matching rules, primarily when data is represented in string representation. Examples are Euclidean distance, Manhattan distance, R-bit chunk matching, hamming Distance, etc.

Most of these terminology and other necessary mathematics related to distance measures have been detailed by the [93] with a statistical explanation.

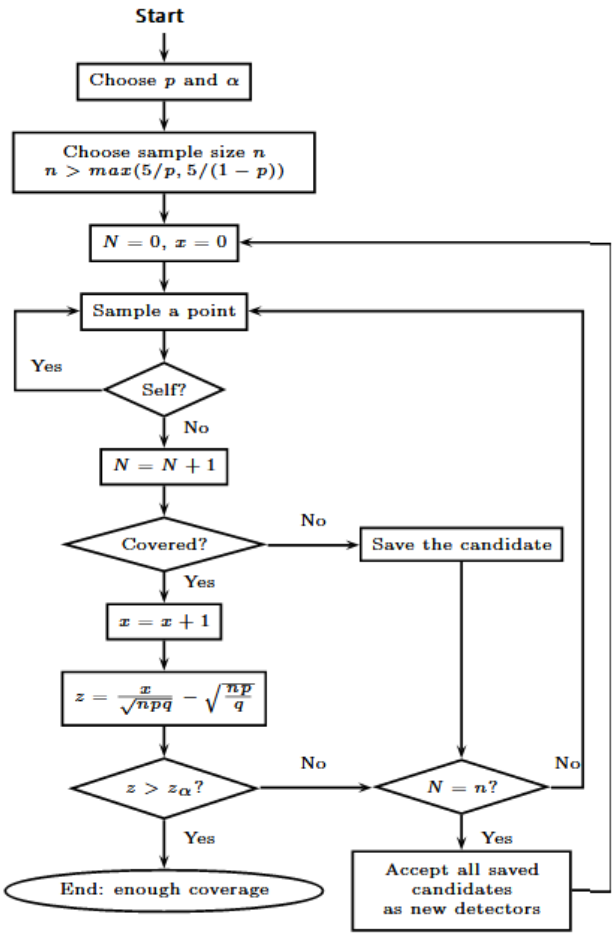


Figure 2.25: A real-valued NSA for generating variable-size detectors (V-detector) with statistical estimate of complementary space coverage[94]. Here N number of the detector, 'x' is number of positive point, 'P' is probability and α is target coverage, n is sample size.

Based on feature value, we can have two kinds of NSA: the binary NSA (BNSA) and the Real-Value NSA (RNSA). Real-value NSA can be variable size or constant size. The BNSA use r-contiguous bits(rcb), r-chunks, landscape-affinity matching, Hamming distance to match the similarity, where RNSA uses mostly derivations of Euclidean distances. One example of a variable size detector or V-detector, V-detector's aim, deals with constant size detectors' drawbacks. In this algorithm, the size radius of detectors is changing from one to the others[94]. The NSA detectors can be represented by string for (binary) BNSA or by a vector in multidimensional space for real values. Later, grid-based representations were also introduced by Yang[218]. A variation of grid representation [232] is known as matrix representation also

introduced as an additional approach. Detector initialization was random in most of the early variations of NSA. Later, researchers tried some heuristic methods to use pseudo-randomness and some evolutionary computation-based adaptiveness technique to initialize detector position. Detector size can be fixed or can change through the generation process. Also, the different detector can take the detector's variable size, and these sizes can be dynamically changes throughout the generation process.

Real-Valued NSA for generating V-detectors

One of the widely-used NSA, called variable-size detectors (V-detector) algorithm was proposed by Ji and Dasgupta[92], which used variable-sized detector radii to cover the complementary space. The V-detector algorithm was improved in subsequent works [95] to limit the number of detectors and adopting boundary-aware strategy [91]. The difference between traditional NSA[64] and V-detector NSA was in V-detector, the detector size in representation spaces are different than each other, and they are aware of each other existence, so they don't overlap. Figure 2.25 illustrates the NSA V-detector algorithm for generating negative detectors.

2.3.5 One class classifications (OCC)

One class classifications (OCC) solve problems where the training datasets only contain samples of one class, and learning models have to identify new data whether belong to that class or not. it is also known as unary classification or class-modelling problem. Most common approach for solving one class problem is one class support vector machine (OCSVM[39]). Other similar approaches are Minimum Co-variance Method (MCM) [85], Gaussian Mixture Model (GMM)[165], Dirichlet Process Mixture Model (DPMM)[17], Kernel Density Estimator (KDE)[142], Robust KDE[104]. GWR-Netwrok[133], Deep Support Vector Data Description (SVDD), Deep Auto Encoder based Methods[84], Generative Adversarial Net Based approaches (eg: [119, 173], etc. These OCC techniques can be classified in 6 types. Minimum Covariance Determinant(MCD[80]), OCSVM, Deviation-based(LMDD[12]) considered as linear model

based OCC techniques. Another type is proximity based which includes Local Outlier Factor(LOF)[22], Connectivity-Based (COF[191]), Clustering-Based LOF(CBLOF[83]), Histogram-based (HBOS[62]), K-Nearest Neighbors (kNN[161]), Subspace Outlier Detection(SOD [109]).

Angle-Based (ABOD[108]) , Copula-Based (COPOD[121], Stochastic Selection(SOS[90]) are known as probabilistic techniques used for OCC. Combining several methods of OCC are known as ensemble techniques those include Isolation Forest (IF[195]), Locally Selective Combination of Parallel Outlier Ensembles(LSCP [230]), Feature Bagging(FB [114]), Extreme Boosting Based (XGBOD [228]), etc.

With the improvement of deeplearning methods, OCC problems were solved using different neural network models such as Fully connected AutoEncoder (AE[2]), Variational AutoEncoder(VAE[105]), Single-Objective GAN (SO-GAAL [127]), Multiple-Objective GAN(MO-GAAL[127]), etc.

Chapter 3

Proposed research

3.1 Goal and Objectives

Researchers have suggested several benchmarks [32, 13, 31, 28, 27, 198] and use these benchmark to evaluate defense techniques. They also suggested guidelines on how to develop efficient defense techniques. Most of their benchmark focused on how many attacks a defense technique can defend. Some of them prioritized testing against adaptive attacks. As these benchmarks were mostly dependent on dataset and attack set, how these defense techniques compared to other techniques, such as compliance, computational cost, cybersecurity practices, etc are mostly overlooked. Also, these evaluations did not provide a metric for measurement. I concluded that these benchmarks are very good to evaluate defense techniques' performances and novelty but these benchmarks do not help a learning model developer to pick a defense technique appropriate for specific problems. I tried to answer these limitations by My benchmarking system.

I simulated some use cases where a Machine Learning model requires a defense technique to protect against adversarial attacks.

- First case: I considered that I will protect a Pytorch[102] Resnet model for Predicting handwriting digit tool for android devices.
- Second case: I considered that I will protect a Pytorch Resnet model for Predicting handwriting

tools running on an Amazon Web service[146] using the Django framework[58].

- Third case: I considered protecting a MNIST TensorFlow project using ML.NET project[117] as a console application.
- Fourth Case: I considered that I will employ an MNIST dataset using Deeplearning4j[107] from java library as a desktop project.
- Fifth Case: I considered protecting Close circuit camera used in a parking lot for reading car number-plates using deep-learning.

I tried to secure the learning model for the above-mentioned cases with standard cybersecurity practices such as keeping data privacy, secure authentication, and access policy. I considered most of the mentioned defense techniques discussed in section **??**. I observed that many defense techniques were not suitable for some of My cases. For example, image prepossessing based defense is not suitable for mobile apps and adversarial training reduces the accuracy for number plate recognition. Adversarial training based defenses were not suitable if training data is sensitive to share. GAN based defense requires high computation cost, which seems very unreasonable for simple AI tasks where the higher error rate is tolerable. From these case studies, I understood which defense techniques were better suited and what factors were more important than others. I weighted them based on My observations. These factors are:

- F1:Tested against multiple data set. $W = 10$
- F2:Tested against black-box -white box attack. targeted-non targeted, gradient based-non gradient based attacks $W = 5$
- F3: Have low computational overhead cost.$W = 5$
- F4: Cross-models and multi domain applicability .$W = 5$
- F5: Tested against adaptive attack.$W = 10$
- F6: No machine learning involved in the defense technique.$W = 5$

- F7: Randomness exist to answer obscurity.$W = 10$
- F8: No training data needed.$W = 10$
- F9: No Knowledge of the learning model needed to know.$W = 10$
- F10:No modification of the learning model needed.$W = 10$
- F11:No accuracy drop of learning model needed.$W = 10$
- F12:No adversarial knowledge needed to generate defense.$W = 10$

F1 is very important for understanding the effectiveness of an adversarial attack. I weighted these by 10. Number F2 is about the diverseness of attack types. I weighted this by 5. Learning models usually have large computational complexity; thus if the protecting tool requires higher computational overhead it will make full system impractical to use due to both time and cost. I weighted it by 5. Some defense techniques can only work for a specific domain; for instance My proposed methods are only for computer vision domain, and I tested on against attack samples from Resnet and VGG-16. As adversarial samples have transfer-ability to another learning model. It is expected that defense methods will supports cross-models and multi-domains. I weighted this factor only by 5 as defense techniques are easily customize-able as per requirement. Defending against an adaptive attack is very important, as the attacker can try continuously until succeeding. So F5 is important as F2. If there is a Machine learning involved in the defense technique than that technique can be also susceptible to adversarial attack. So involving defending Machine learning with another machine learning will not improve security rather it creates another door way of the same vulnerabilities. I weighted F6 as 5. I have to assume that My defense technique details are known to the attacker. That's why randomness is needed which will make it hard for predict what defense configuration will be set for each attack time. I weighted this as 10. F8, F9, F10, and F11 are related to CIA models of information security. All information security policy measures try to address three goals known as confidentiality (protect the confidentiality of assets), integrity(preserve the integrity of assets), and availability (promote the availability of assets for

authorized users). These goals form the CIA model[7] which is the basis of all security programs[172]. Here, ML is considered as a digital asset, I ensure its confidentiality as I am not consuming or accessing any architectural information of ML. The same way integrity is preserved as I don't need to modify or tune anything in ML architectures. If training data or learning models are needed to generate defense it will violate confidentiality if modification of the learning model is needed it will violate integrity and if the accuracy drop it will violate the availability. Because of this dilemma, I weighted F8, F9, F10, F11 by 10. Zero-day vulnerability can also be present in adversarial defenses if there are no safeguard against unknown adversarial attack methods. There is high probability chance of unknown attacks that's why I give this factor 10 weight value. I disregarded several factors such as easy maintenance or update facility, time to implement, etc, as they also depend on the learning model itself. The total score of a defense technique can be

Defense Techniques	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F_score
Adversarial Training [140] [197] [113]	10	5	5	5		5			10	10	10		0.6
Image Prepossessing [215]	10	5				5		10	10	10	0		0.5
Input Reconstruction[136]	10	5	5			5			10	10	0		0.45
Distillation Techniques [151],[180]	10	5	5	5		5		10	10			10	0.6
Transform Function [176]	10	5				5		10	10	10	10		0.6
Defense GAN[171]	10	5	5	5	10	5							0.4
Model Robustifying[19][185]	10	5	5	5		5			10	10	10		0.6

Table 3.1: Here, I scored based on equation of F_{score} from My study of different defense techniques. (Some of these scores are an approximation based My understanding) In the last column, the overall score is presented. Based on the overall score I can see Some of the techniques have better usability than others.

measured by below equation

$$F_{score} = \frac{\sum F_{i=1..12}}{100} \tag{3.1}$$

The Maximum score is possible up to 1 and the lowest score is possible as low as 0. Based on these factors, I create a radar map as shown in figure 3.3 with 3 defense techniques. Here, the effectiveness represents by the factor F1 and F2. Computation feasibility represents by factor F3. F4 represents by platform-independent. Vulnerability represents factor F5,F6,F7. These factors cover potential vulnerability such as a zero-day attack, advanced persistent attack, and insider attacks. F8-F12 are represented by cybersecurity Compliance. I further analyzed other defense methods and presented their benchmark result in table 3.1. I can see that none of the defense technique have a better score than 0.6. As the max possible score is 1, there are opportunities to

improve.

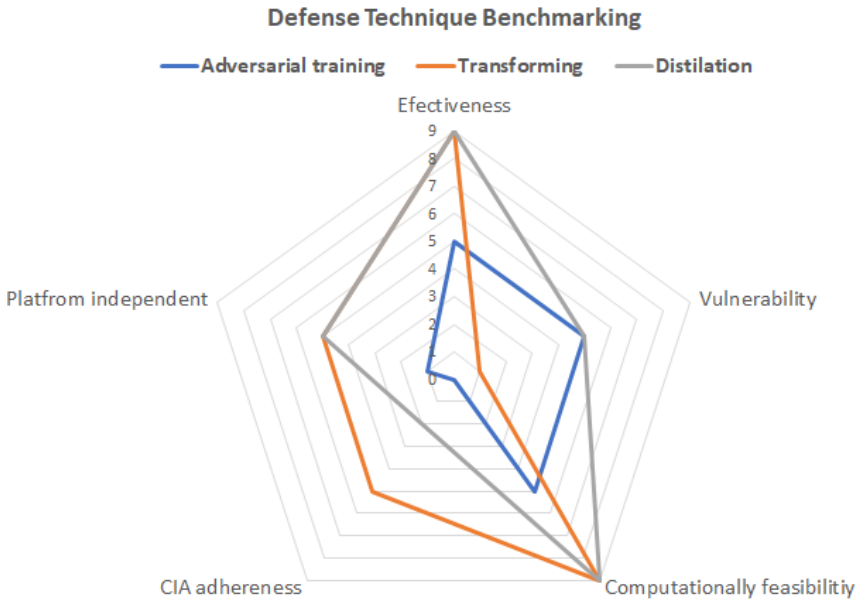


Figure 3.1: My Proposed Bench-marking Process with sample data. Here I tried to represent some common defense techniques in a radar map.

I aim to provide an adversarial defense system which will meet the below objectives:

- Defense needs to work against a diverse set of attack types. My provided defense technique should work against Gradient or no-gradient, white-box or black-box, targeted or no targeted, adaptive attacks [32].
- Defense should not reduce the accuracy of ML models. The model accuracy should not get effected after deploying My defense technique.
- Defense needs to identify threats faster. If a defense system takes sizeable computational time and resources, it will lose the practicability. For example, if the defense is employed in an autonomous car sensor, the input responses need to evaluate first. Otherwise, an accident can happen.
- Defense should not modify ML architecture. Defense should work for both the white-box and black-box models. A trained ML architectural information is usually black-box. So, it

is expected that the defense framework will comply with that.

- Defense should be adaptive in nature and dynamic to prevent the adaptive attacks.
- Defense should not need to update if ML changes (Resnet to VGG or ANN to RNN), and it should be cross-domain (image, audio, text) supported.

3.2 Basic Architecture

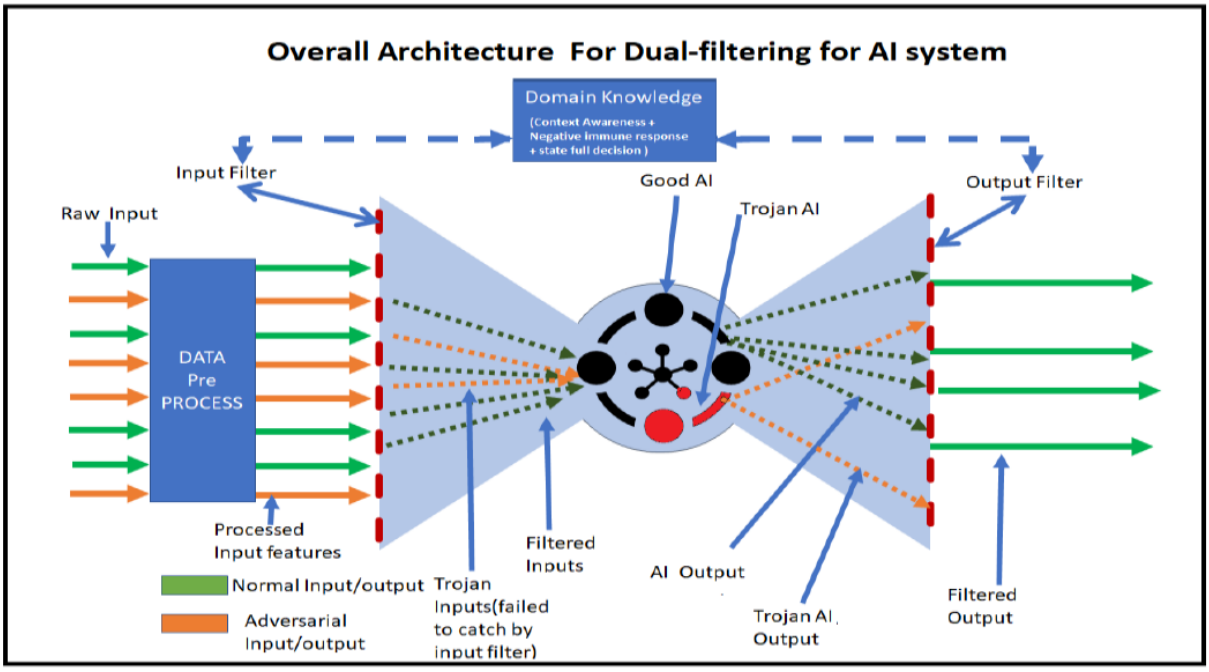


Figure 3.2: Schematic of the proposed Dual-Filtering (DF) framework.

To build a robust ML/AI-based system against malicious adversaries, I designed a dual-filtering (i.e., commutative filtering) scheme, which employs two filtering mechanisms: one at the input stage (before samples are fed to the core learning model) and other at the output of ML (before the decision module). These two filters can function independently as well as dependently (i.e., in a commutative fashion). Specifically, the input filter's main aim is to filter misleading and out of distribution inputs (e.g., image of animal but not human face in a face recognition system). The

output filter's goal is handling larger variations and restricting misclassification rates in order to improve overall accuracy of the system. The proposed dual-filtering strategy can be used both in training and testing phases of ML. For instance, the independent input filter may be used to detect and deter the poising attacks in a supervised ML. Likewise, dual-commutative filters may help addressing adversaries both in supervised and unsupervised ML. A machine learning framework usually consists of four main modules: feature extraction, feature selection (optional), classification/clustering, and decision. As depicted in Figure 5.3, the input filters are placed after pre-processing of data stream/feature selection to feed to core learning model and the output filters are placed after classification/clustering/raw decision module, respectively.

As can been seen in Figure 5.3, the raw input sample is first pre-processed and then fed to the input filter to determine if the received feature/sample is benign or attack and reject accordingly. The outcome (i.e., raw decision) by ML system is given to the output filter for further scrutiny. The output filter uses context-information and/or communicates with the input filter to make the correct final decision. An ensemble of different noise removal or detection filters was applied to detect AAs in a recent work [47]. Other techniques focused mostly on adding extra layer on a ML module by adversarial sample training or modification of deep learning models. These defense methods have some constraints, and exposed ML models to new vulnerabilities [76].

In 2019, some works reported launching adaptive attacks where they could bypass known defenses [32]. To alleviate the situation, I consider a non-deterministic (white-box) approach where the attackers cannot perceive My defenses to launch adaptive attacks. Accordingly, I investigated an active learning[175] based dual-validation scheme which work as an extra security (filtering) layer and improve the learning model's trustworthiness.

Accordingly, My defensive measures for machine learning model (MLM) have the following tasks:

- **Input filter before MLM:** The primary purpose of input filters is to prevent adversarial input data in such a way that can differentiate data manipulation from the trained data. It

will be examining the input by deploying application-specific filter sequence. A set of filter sequences are selected (from a given library of filters) using an efficient search and optimization algorithm, called multi-objective genetic algorithm (MOGA). The MOGA can find a sequence of filters (where each filter can detect adversarial traits/noises) satisfying constrains and three objectives: detection of the maximum number of attacks with higher accuracy (above a specific threshold), with minimum processing time, and shorter sequence of ensemble filters. By utilizing the Pareto-set from MOGA runs, and picking a filter sequence dynamically at different times, make filter selections unpredictable and use an active learning approach in order to protect the ML from adaptive attacks.

- **Output filter after MLM:** Employ several class-specific latent space-based transformation for outlier detection. After MLM provides an output class label, it is then verified if the output falls in that class's latent space or not. I will make an ensemble of different outlier detection methods and sequence dynamically and also retrain the outlier methods runtime.

3.3 Use Standard Data sets

3.3.1 ML Dataset

For bench-marking, MNIST and CIFAR are mostly used by academicians over the world. There are many other datasets for neural network experiments, and I have used the below data set

- MNIST
- CIFAR-10
- ImageNet
- SVHN
- NPD

Modified National Institute of Standards and Technology database (MNIST)

MNIST consists of 60000 training and 10000 testing data of single digits zero to nine. These are all black background images with written numbers in white colors. Using a linear classifier, it has 83% accuracy. But with using CNN, accuracy is above 99.6%.

Canadian Institute For Advanced Research (CIFAR)

CIFAR dataset includes 60,000 32x32 color images in 10 distinct classes (6,000 images of each class) . These classes are planes, cars, birds, deer, dogs, frogs, horses, cats, ships, and trucks. The regular CNN has near 80% accuracy, but ResNet has accuracy near 98%, and some efficient DNN methods reach the accuracy of 99% [208].

ImageNet Dataset (IND)

ImageNet Dataset has more than 14million images, which have been handpicked and classified. It has more than 20000 categories, for example, ballon, strawberry, etc. Using CNN made it possible to achieve 85% accuracy here. Due to algorithmic bias use of ImageNet is not always applicable as a standard dataset.

Street View House Numbers (SVHN)

SVHN is a dataset for house numbers; it has ten classes, 1 for each digit. With 73257 digits for training, 26032 digits for testing, and 531131 additional unlabeled data. Using CNN, accuracy is over 98%.

Number Plate Dataset (NPD)

NumberPlate Dataset [181] contains 100000 number plates from the USA and Europe. Using CNN, the accuracy of this Dataset is above 98%

3.3.2 Generated Adversarial Dataset

I generated 60000 adversarial images from FGSM, JSMA, and CW attacks for MNIST and CIFAR datasets. 10000from each attack type and 1000 from each class label. This dataset is the largest adversarial dataset on MNIST and CIFAR, as best of My knowledge from My extensive literature survey. I published these datasets in Kaggle publicly for result reproduction and further study by other researchers.

3.4 Threat models

	ATTACK TYPE		JSMA	FGSM	DF	HSJ	BPDA
Threat Model	Knowledge	WhiteBox	*	*	*		
		BlackBox				*	
	Specificity	Targeted	*			*	
		Non-Targeted		*	*		
	Frequency	One time		*			
		Iterative	*				
	Methodology	Gradient based	*	*	*		
		Gradient free				*	
	Adaptive						*
Benchmark	Dataset	MNIST	*	*	*		
		CIFAR	*	*	*	*	
		IMAGENET	*	*			
	ML Model	ResNet	*	*			
		Simple CNN			*	*	*
		VGG		*			

Table 3.2: Summary of Threat Models and associated manipulation strategies and dataset used for experimentation.

3.4.1 Attack types

Yuan et al. (2018)[220] suggested making threat models consist of Adversarial Falsification (False negative, False Positive), white-box, BlackBox, targeted, non- targeted, onetime and iterative attacks. Carlini et al.[32], suggested that adversarial attack and defense models need to

be tested against a diverse set of attacks. Also, they need to be evaluated against adaptive attacks. Moreover, Tramer et al. [198] suggested different themes to evaluate a defense model. Keeping these guidelines in mind, I developed My threat model inclusive of basic, advanced attack and adaptive attack (against My defenses) types. Carlini et al. [28] also recommended using at least one gradient-free and one hard-label attack. To address that concern, I evaluated My proposed method with gradient-free attacks such as local search attack [145] and hop-skip-jump attack [35]. For testing against an adaptive attack, I used BPDA (Backward Pass Differential Approximation [14]), which can be used to attack non-differential prepossessing-based defenses. Uesato el al.[199] advised to consider obscurity of adversarial attack when considering the defenses. [32] pointed out that testing a defense in one dataset is not enough, therefore I chose multiple datasets (i.e., MNIST, CIFAR-10, and ImageNet). I considered a standard distortion $\epsilon = 0.3$ for MNIST and $\epsilon = 8/255$ for CIFAR-10, as current state-of-the-art [198] recommended. Thus, My threat model is a combination of gradient-based, gradient-free, and adaptive evasion based adversarial attacks on multiple datasets. These attacks studied in this work are a combination of White-box, Black-Box, targeted and non-targeted attacks. Also, the presented defense will be able to defend against attacks that are completely unknown to the proposed defense scheme.

Table 3.2 summaries the threat models I investigated in My work. Here the second column shows different strategies used in threat models, the third column mentions usability tactics and the rest of the columns provide specific image manipulation techniques.

3.4.2 Attack Samples generation

My attack samples are PGD, BIM, MBIM, FGSM, JSMA, DF, HopSkipJump, Localsearch, and CW methods. I generated a minimum of 1000 adversarial samples from each attack type to ran My experiments. I find out a sequence of blur+AS+pixellete works for PGD, BIM, FGSM, MBIM, which I used as a defended model in BPDA adaptive attack and generated new attack samples. I generated FGSM, JSMA, CW, and DF using Pytorch [63]. I used BIM, MBIM, PGD, local Search attack, and HopSkipJump using IBM-ART-Toolbox[149] and Cleverhans adversarial

library[153]. I used modified advertorch[54] to generate BPDA attack samples. I noticed that the destruction rate (i.e., the rate of failure of adversarial attack when it is converted to visual form) [112] is higher in advanced attack types. I disregarded those images from attack samples. Also, due to My restriction of ϵ = 8/255 for CIFAR-10 as maximum noise value, I had to discard some samples from My dataset. I also experimented with three types of adversarial patch's on ImageNet dataset. I used 10-20 adversarial patches for My experiment using different adversarial patch attack like DPATCH [125], LAVAN[99].

3.5 Preliminary Experiments

Types	**Normal**	**Resize**			**Rotate**			**Motion**			**Illumination**		
		2x	0.5x	4x	5'	15'	45'	15x	20x	50x	5	15	25
Dpatch [125]	**10**	0	5	0	0	5	5	35	35	80	0	0	5
Lavan [99]	**15**	0	5	0	0	5	5	35	35	85	0	0	5
Patch [24]	**18**	3	6	3	0	6	6	55	60	70	0	0	5
FGSM[65]	**43**	20	18	20	0	10	15	60	77	85	5	10	12
BIM [132]	**32**	28	16	28	5	12	18	66	76	86	10	15	20
JSMA [209]	**28**	22	10	22	0	10	25	60	75	85	15	25	30
CW [30]	**61**	55	75	55	60	75	90	100	100	100	70	80	100
DF [143]	**85**	90	95	90	72	90	95	100	100	100	85	90	100

Table 3.3: Destruction rates of various attack types under different environmental conditions. The values in column "Normal" indicate destruction rates under raw image and following columns represent where the successful attack types in normal conditions are experimented with other conditions. In resize, I re-scaled the image and turn backward to original sizes. To simulate motion effect I used Gaussian blur with different sigma value. For illumination effect I increased the brightness. I can see the destruction rates gets higher when rotate and motions are higher. Adversarial patch'es have lower destruction rate.

3.5.1 Applicability issue of Adversarial Attacks

To explain the applicability issues of adversarial attacks, I need first to understand the detection and destruction rate of adversarial attacks.

- **Detection Rate**: Adversarial attack is possible to distinguish from the non-adversarial

samples by different detection methods. Different techniques have different efficiency to detect the adversarial samples. This efficiency to detect adversarial samples or adversarial image (AI) can be measured by the detection rate. In short, My detection rate can simply put as:

$$Detection_{Rate} = \frac{\sum AI_{detected} + \sum Non_{AI} - \sum False_{detected}}{\sum AI + \sum NonAI}\% \tag{3.2}$$

- **Destruction Rate**: Most of the adversarial attack studies ignore the destruction rate, but I prioritize the destruction rate to evaluate practicability. There could be many reasons that an adversarial attack sample fails to perform as adversarial samples. The rate of this failure can be represented by the destruction rate. Kurakin and Yan Goodfellow [112] provided an equation when adversarial images failed to identified as successful adversarial attack after they converted to PNG or printed on a paper. I used the same equation to provide destruction rates of attack samples. In short, My destruction rate can simply put as:

$$Destruction_{Rate} = \frac{\sum FailedAdversarialImages}{\sum AdversarialImages}\% \tag{3.3}$$

Kurakin[112] represented destruction rate phenomena by the following equation

$$d = \frac{\sum_{k=1}^{n} C(X^k, y^k{}_{True})\overline{C(X^k_{adv}, y^k{}_{True})}C(T(X^k_{adv}), y^k{}_{True})}{\sum_{k=1}^{n} C(X^k, y^k{}_{True})C(X^k_{adv}, y^k{}_{True})} \tag{3.4}$$

Here, destruction rate is the fraction of adversarial images which are no longer misclassified in real world scenario.

It is well observed, the adversarial images generated do not remain as adversarial when converted to the visual format of images such as 'PNG'. Deep learning models typically use floating values instead of using integer values which are present in image RGB format. This is because it helps better convergence. Real numbers have infinite range and depend on the precision, where as integers have finite range. The activation functions perform better in achieving global optima, like sigmoid activation/Tanh works better with floating values. If I use integer

values there is a chance I will miss many local optima and there is a chance I never get a global optimum. Common hardware is equipped to run deep learning with floating-point values. This tendency of preferring floating conversion from integer values of RGB creates the practicability problem for adversarial images. Due to the conversion of float to integer, added noise/perturb disappears. In the paper, Kevin Eykholt et al.[56] reported that the effect of adversarial example get minimized due to several factors in real world scenario. They are environmental conditions, spatial constraints, physical limits on imperceptibility and fabrication Error. They provided a sticker based adversarial attack which works in real world from different angle and distances. However, this attack seems easily identifiable by the human eye due to the shape and intensity of embedded noise. Sharif et al.[177] used adversarial perturbations on the lens of eyeglasses to attack the face recognition system. However, such attack didn't mention destruction rates of adversarial perturbation also their experiment was in a constrained environment. In 2019, Zeng et al.[222] added perturb in 3D models instead of 2D images and showing successful adversarial attacks but there was also no discussion about destruction rate, which makes it difficult to reproduce with the same accuracy. In 2017, Kurakin et al.[112], showed that in the digital version and printed version destruction rate exist, and for advanced attack methods the destruction rate gets higher. They tried to justify their argument with FGSM, BIM, and least likely iterative methods. Lu et al.[130] in their paper experimented with FGSM, BIM, and LBFGS methods and showed the destruction rate can be achieved up of 100% based on distances that invalidating these attacks. Pierazzi et al.[157] shows that, "it is feasible to create adversarial examples in the problem space (realizable attacks) and that it seems there is no correlation between the ability of the classifier to detect such attacks and the disruption of the adversarial examples". However, the problem with Pierazzi et al.[157]. works is that they restricted the problem space which are not applicable for real situation when other environmental factors are present.

As an example, an image of class label 'A', pixel values are $\begin{pmatrix} 127 & 243 & 47 \end{pmatrix}$ will divided by 255(8bit range) and converted to $\begin{pmatrix} 0.4980392157 & 0.9529411765 & 0.1843137255 \end{pmatrix}$

$.The perturbs\epsilon$ are

	Destruction Rate				Destruction Rate(with threshold)				Destruction Rate(threshold+30)%			
Dataset ->	**MNIST**		**CIFAR**		MNIST		CIFAR		MNIST		CIFAR	
Attack types	**Digital**	**Print**	**Digital**	**Print**	Digital	Print	Digital	Print	Digital	Print	Digital	Print
FGSM [65]	**43**	**22**	**53**	**24**	2	33	6	24	1	23	3	24
BIM [132]	**32**	**26**	**68**	**26**	1	21	4	26	1	26	4	26
MBIM [55]	**46**	**15**	**-**	**-**	2	12	-	-	1	15	-	-
JSMA [209]	**28**	**31**	**76**	**31**	2	22	11	21	1	31	10	-
CW [30]	**61**	**25**	**92**	**32**	12	29	17	27	8	25	26	-
Deepfool [143]	**85**	**22**	**96**	**32**	15	14	32	30	10	16	28	-
HSJ [35]	**45**	**23**	**-**	**-**	5	12	-	-	4	15	-	-

Table 3.4: In first four columns under the Destruction rate without threshold, I show percentages of adversarial samples failed to remain as adversarial. Next four columns show the adversarial rate among the adversarial samples which satisfied My threshold value. In the last four columns I provided the adversarial samples destruction rate when threshold value increases 30% more.

$\begin{bmatrix} 0.00000000007 & -0.00000000004 & 0.000000000089 \end{bmatrix}$, which makes the image values as $\begin{bmatrix} 0.4980392164 & 0.9529411761 & 0.1843137268 \end{bmatrix}$.

Now it is an adversarial image that will classify as label B. But when I convert it by multiplying 255 will return the $\begin{bmatrix} 127 & 243 & 47 \end{bmatrix}$ which has class label A. This adversarial operation does not exist in real world due to noise value being too small. As all adversarial attack types aim to reduce the epsilon value the practicability issue rises more in advanced attack types.

I proposed a method of minimum threshold of perturbing value which guarantees that noises will affect when the adversarial sample is converted to any image format. However, it does not differentiate whether the perturbed image will be an adversarial or not, it assures that this noise will affect when the image will be an input for an ML model. I tested, attack samples with minimum threshold value's. I determined the destruction rate and compare the result with attacks sample's conventional destruction rates. Assuming, vector spaces of pixel values are converted between 0 to 1. If the standard floor and the ceil math function are used to reconvert vector values to the image, I can have the below formulation for the single color channel.

Let, a pixel non floating value X for a single color channel.

Image each channel are N bit. Thus perturb value ϵ need to bigger than a threshold value, which can be derived from below equation.

$$(\frac{x}{2^N - 1} + \epsilon) \times (2^N - 1) \geq x + 0.5 \tag{3.5}$$

From Equation 3.5, I get

$$\epsilon \geq \frac{0.5}{2^N - 1} \tag{3.6}$$

For, 8 bit single channel color, the minimum threshold value will be $T \approx \frac{0.5}{2^8-1} \approx 0.00196078431$

From printed version if an average accuracy drop is δ, for a printed version the threshold needs to increase by

$$T_p \approx \epsilon + \frac{\epsilon \times \delta}{100} \tag{3.7}$$

As an example, I observed 20% drop of FGSM method for 8bit MNIST grey channel adversarial, So here the minimum threshold should be
$T_p \approx 0.00196078431 + 0.00039215686 \approx 0.00235294117$ Any noise ϵ need to be greater than T to have a chance of becoming a digital adversarial image and be greater than T_p to have any chance in becoming a printed adversarial image.

I experimented with basic attack types such as FGSM, JSMA also as and advanced attack types such as CW, Deepfool, etc and used MNIST, CIFAR, and ImageNet data sets. At first I converted all adversarial samples to PNG and observed destruction rates. The successful attack PNG's are printed and scanned as PNG again (similar as [112]. Then I cropped and resized them as original training samples and tried again with the ML model again, I calculating which images were correctly classified as an adversarial attack. Using equation 3.5, I calculated the destruction rates under different environment constraints(example: Rotate, Resize, Illumination, Motion). I simulated motion by blurring the image and I increased the brightness for illumination effect. The results are presented in table 3.3. Here for adversarial patch attack, I used ImageNet and for other attacks, I used average destruction rate of CIFAR and MNIST combination. In the 6th row, FGSM has a 43% destruction rate when it converted to image format. The remaining images have a destruction rate of 10% when their brightness increased 15%. This table shows that Adversarial patch-based attacks have higher tolerance and CW, Deepfool has lower tolerance from the environmental factor.

I used equation 3.5 with δ value 20% and considered image's with at least 70% of noise

above the calculated threshold. I converted the images in PNG and measured the destruction rates. After that, I printed the successful PNG to print and measure the destruction rates. The experimented results are shown in table 3.4. I used an image difference (Pixel value difference) technique to calculate the δ value as Gupta et al.[77] shows in their paper.

From table 3.3, I can see motion and rotation has more effects. I observed that CW and Deep-Fool attacks had a 100% destruction rate if they rotated too much. It is also observable that patch-based attacks have good performance in real world conditions. These results proved that if I use some reflecting technique I can avoid advanced perturb based attacks.

From table 3.4, I can see that when only considering adversarial images with noise above My threshold, the drop of accuracy sharply declines from the previous result in the table3.3 where all the adversarial samples destruction rate were shown. From the table, it appears destruction in the printed version does not change much based on the attack type. Destruction rate is pretty consistent with any normal clean image detection failure rate, as well,

3.5.2 Adversarial Input detection

I applied 4 transformation techniques such as Fourier transform [219], Census Transform[60], Gabor Transform[182] and Wavelet Transform[156] on adversarial images and also applied them on clean data set. I calculated average SNR value from all clean images and use that as threshold value. I observed SNR values are higher in adversarial images and using that threshold value I can detect basic adversarial attacks. In the image, SNR values are calculated by using the mean of pixels as a signal (S) and std deviations of pixels as noise (N) with below equation

$$SNR = 10 \times \log_{10} \frac{S}{N} \tag{3.8}$$

I presented these results in table 3.5 for MNIST data-set. I not conducted detection experiment for adversarial patch attacks but as they are visible to human eye and the experiments of researcher Chiang et al.[42] shows that adversarial patch attacks are also detectable, I set the detection rate

	Detection accuracy rate by SNR value using simple transformation			
	Fourier	**Census**	**Wavelet**	**Gabor**
FGSM	89	99	99	70
BIM	85	80	95	65
JSMA	75	70	85	55
HSJ	55	55	65	0
CW	30	10	15	0
DEEPFOOL	15	15	15	0

Table 3.5: Using four common transformation technique to distinguish between adversarial samples and common samples (From MNIST). I used 20000 clean images avg Signal to noise ratio(SNR) as threshold. Attack images which have higher SNR than the threshold are identified as adversarial images . Here I provided the detection rates. I can observe that CW and Deepfool Detecting is harder.

higher as FGSM.

when I observed the detection rates of different attacks in table 3.5. Here in the1st row, if FGSM samples transformed using Fourier transform and calculate its SNR value that 89% of FGSM images are outside of Clean images SNR value range. I can see the CW, Deep-Fool attacks are hard to detect from SNR values. From the results of the above experiments I draw the graph in figure 6.1. Here, In the graph, I can see the destruction rate and detection rate are co-related each other. Based on these rates I calculated feasibility scores of different attack types for MNIST dataset.

3.5.3 Identifying Vulnerability

Based on our previous discussion, we can see evasion-based adversarial attacks are difficult to formulate in the real world. Environmental factors can easily cancel out adversarial noises. The noises which are not canceled out are also very easily detected by common filters. Most of the defense technique has 99% accuracy rate of adversarial detection for high noise adversarial attacks (example: FGSM, JSMA, BIM). So, this question can simply arise that do we need to care about adversarial attacks?

To answer this question, let's observe where an adversarial attack can be formulated. In

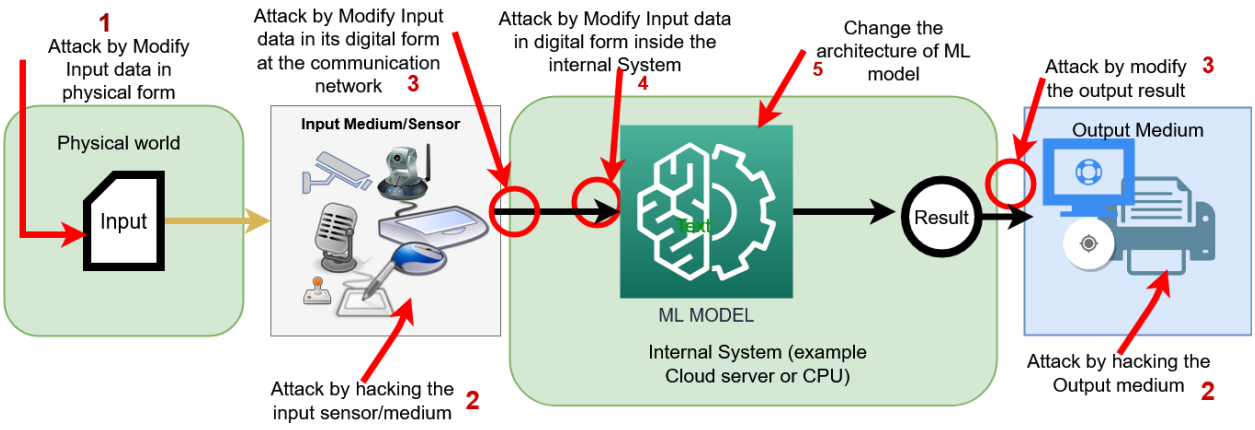


Figure 3.3: Generalized adversarial attack points on ML model

the figure 3.3, we illustrated general points where adversarial manipulation could happen on a system where an ML service is deployed. We will consider the ML model as a black box. We marked the attack points by number.

First, when the input is on the physical world, an adversary can change the input as a famous adversarial example of changing stop sign to speed limit sign conversion. Here, we have to note that this attack needs a higher physical degree of modification. Hence, if the noise amount is low, there is a high chance that environmental factors will nullify the attacks based on our literature review.

The second point is primarily I/O layers. For example, if someone changes one of the sensors so that sensors can add adversarial noises while converting physical input data into a digital form. One prominent example is putting an adversarial patch sticker on the CC cam lens to manipulate ML output[**li2019adversarial**]. This attack can also happen when an output action was presented or happening.

The third point is where data turned to a digital format (binary/floating/etc.) and sending to the ML system or output receiving from ML (Assuming the ML system is on a cloud server or some computational hardware). We can say it's more of a communication channel. It could argue that if adversarial can control this channel, he can modify the data going to the ML or coming from ML. Low perturb-based attacks can occur here as no environmental factors will diminish the

noise. However, we can also say that if someone can control this network, he might not need to add adversarial noises; he can directly add his desired class input for his expecting output. Also, if he can modify the output result no need for adversarial perturb noises. Yet, this scenario should be considered as might the attacker don't have complete control of the communication channel and can only add slight perturbs.

In the fourth point, the attack can also happen if the system (example: server) is compromised. This scenario, also similar to the third point. It more of an intrusion in the system. Low perturbs attack can happen, but we can see that if the attacker already compromised the server, he doesn't need to compromise the input. It is easier for him to change the output label, as he already compromised the system. We can argue that the intruder's ability is limited, so that he will prefer adversarial attacks.

The fifth point is very different compare to other points, and this attack can happen before the ML model has been set up on the server. Some Trojans or backdoor can be added in the ML before deployment, which is only activated for specific inputs.

Additional point which is not presented in the figure was adaptive attack. An attacker can run different test input to determine the decision boundary of ML model. Also, as adversarial attacks are transferable attack generated in another system will work in here.

From our above discussion, it is evident that adversarial attacks can happen outside the physical world that can make an ML model provide wrong output.

Based on our previous discussion, we can see adversarial defenses each have a different implementation. For example, adversarial training or distillation technique requires training the ML model, resulting in data scientist purview. A prepossessing approach such as feature squeezing types of defense may employ by system engineering in run-time. But the limitations of these defense techniques create a challenge for the concerned parties. Pre-processing methods can protect most of the standard adversarial attack, and it can employ when input sensors are converting to digital format.

Second point attack, as described in figure 3.3 may concern with knowledge concerning

hardware security or IoT device security, or firmware security. This part interacts with humans, so human-computer interaction-based defenses (trust models?) may fall in this area.
Pre-processing-based adversarial defense technique can be employed here before sending over the communication channel.

Third point attacks, as described in figure 3.3 are in the communication channel, so an attack on this channel should be concerned by network security experts. As if this channel is compromised, an attacker can create different attacks rather than adversarial attacks. Some of these attacks may be a more severe threat than adversarial attacks. Computer network layer security, data layer security also concerned here. It is in question that is it possible to use any of the adversarial defense technique can be employed in this stage or not. It should be noted that traditional adversarial defenses, as far as our literature review, are not applicable in this area, and here are new research opportunities for the interested cybersecurity experts as they can explore from adversarial attack perspectives with traditional attack types.

The fourth point directly falls in Operating system security or cloud security based on ML implementation. It could argue that prepossess-based defense technique can also be employed here, but if there is an intrusion in here, that we have more vital concerns than adversarial attack as the system is already compromised. The attacker is effectively controlling the system. Many proposed adversarial defense techniques can be implemented here. Still, all of these have a limitation: an attacker can modify data in the OS label. He can probably corrupt the employed defense system there too.

The fifth point of attack can be defended by checking the output with other AI systems or conducting thorough testing. Effectively as the ML model is, black box system engineers or cybersecurity has a minimal role here. Data scientists can develop some algorithms to test the ML model after deployment to discover this issue. We considered this to be the most concerning attack points as it is hard to address and may need extensive regular testing.

The adaptive attack is another grave concern, which is also hard to defend as adversarial inputs are transferable. It can tackle with dynamic ML model configuration, regular update, or

active learning system. It can also help if employ a defense system to identify attack query pattern, which can relate to DDoS attacks. However, one downside of adaptive attack is it is a computationally expensive and time-consuming attack for a system that is regularly updating. From the above discussion, we prepared the table 3.6, where we illustrated what part of defense

Attack point	Defense	Role
Input Modification	Preprocess technique	System/Software devs
Sensor,I/O device	Testing and Reliability	Devops /AIops
Communication Channel	Network defense, Cryptography	Network and IT security
Internal OS system	Intrusion detection, Access control	Network and IT security
Model Structure	Formal methods	Data/ML scientist
Adaptive attack	Dynamic and active learning	Data/ML scientist/Network

Table 3.6: Different attack points and relevant responsible professional

falls in which security domain. This table is an attempt to divide the roles, and we can assume there was more intercommunication between different roles required to implement a robust defense system.

It is evident that in the literature survey we did in previous chapter, most of the defense strategies except prepossessing techniques are not suitable in deployed environments. Developing an adversarial defense technique that can work deployed environment and cover all the attack points still a challenge to researchers.

Chapter 4

Step by Step Investigation

4.1 Input Filters (library)

4.1.1 Feature Extraction strategy

Adversarial samples are distorted versions of non-adversarial samples. This distortion can measure in euclidean distances or pixel difference distance. However, I assume some image processing algorithms can signify the misuse of an adversarial example in a way that significance can be used to filter out adversarial examples. As an example, if an adversarial image has added perturb pixel all around its main edges, an edge-preserving technique algorithm can remove these perturb. So differences after-before image have a significant value which makes it distinguishable than a non-adversarial picture.

Let, Adversarial image sets are $A_s + \epsilon_s$, and clean image set is A_s, and Filter Set denoted by $F_{FilterSet}$, ϵ_s is total added perturb of all adversarial images So, after applied Filter Set on adversarial and clean image set I will get,

$F_{FilterSet}(A_s + \epsilon_s) \approx A_s + \epsilon_s + K_a$

$F_{FilterSet}(A_s) \approx A_s + K$

where K, K_a is the approximate effect of Filter Set in clean and adversarial image set So

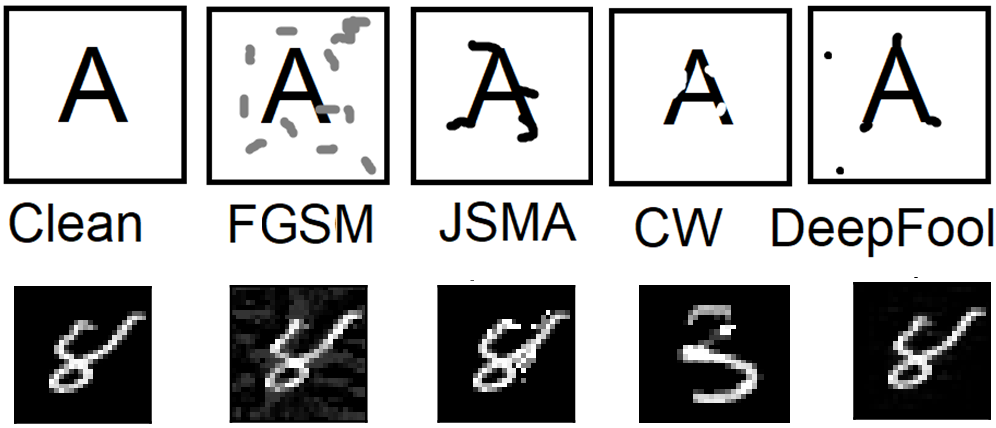


Figure 4.1: First rows show how different types of attack change a clean image (for visual purpose, the effects are exaggerated. This row is created from observation and not real sample). The second rows are actual corresponding real attacks on the MNIST dataset. Where 8 recognized as a different label.

the difference DI is

$$DI \approx |(A_s + \epsilon_s) - (A_s + \epsilon_s + K_a)) - (A_s - A_s - K)|$$

$DI \approx A_s + \epsilon_s - A_s - \epsilon_s - K_a - A_s + A_s + K|$

$DI \approx |K - K_a|$ I can see that in the DI equation no image (A or A_s) is present. Here, aim is analyze the effects not the core image/image content but pre-processing effects. Note: K, K_a is the generalized approximate effect of Filter Set in clean and adversarial image set,

In testing time, I can calculate the k_i value for the testimage i by applying $F_{FilterSet}$.Here k_i is the generalize effects from applied Filter Set. Now i will be a adversarial image if $|K - k_i| > |k_a - k_i|$.

.

I developed a denoising approach to detect adversarial inputs using a sequence of image filters. This work inspired by Prakas et al.[158]. Adversarial samples are distorted versions of non-adversarial samples, such distortions can be measure in Euclidean distances or pixel difference distance. My hypothesis is that, some Filters can signify the distortions of an adversarial example in a way that significance can be used to filter out adversarial examples. For instance, if an adversarial image has added perturbed pixels all around its main edges, an edge-preserving technique can remove these perturbations. The differences before and after

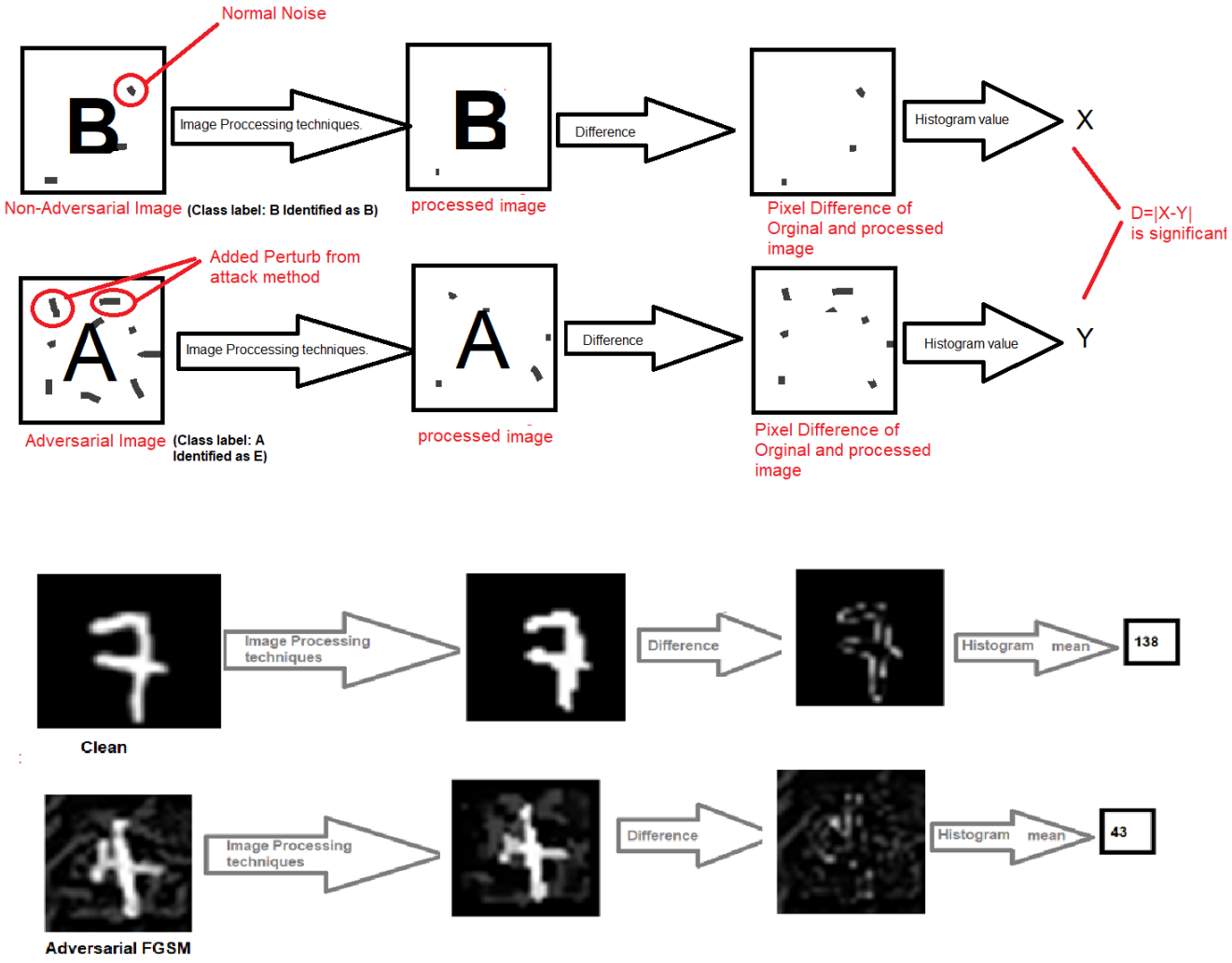


Figure 4.2: In the top side, Clean (class A) vs Adversarial (B which classified as other class) images differences after Filter Set applied, Histogram calculation on the difference, In bottom, MNIST with FGSM example has shown

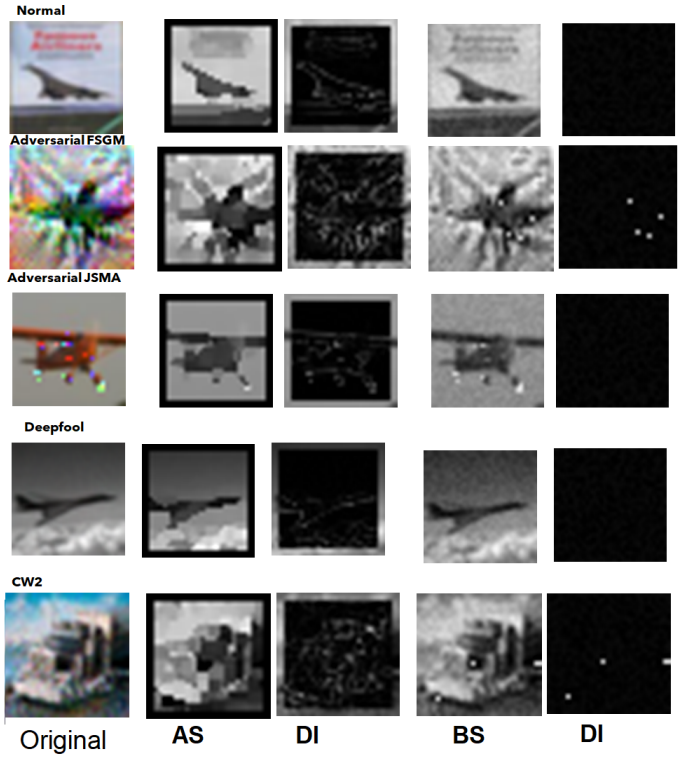


Figure 4.3: Effects of Several Filters techniques on CIFAR dataset.after applying different Filters, DI was done from grey-scaled image

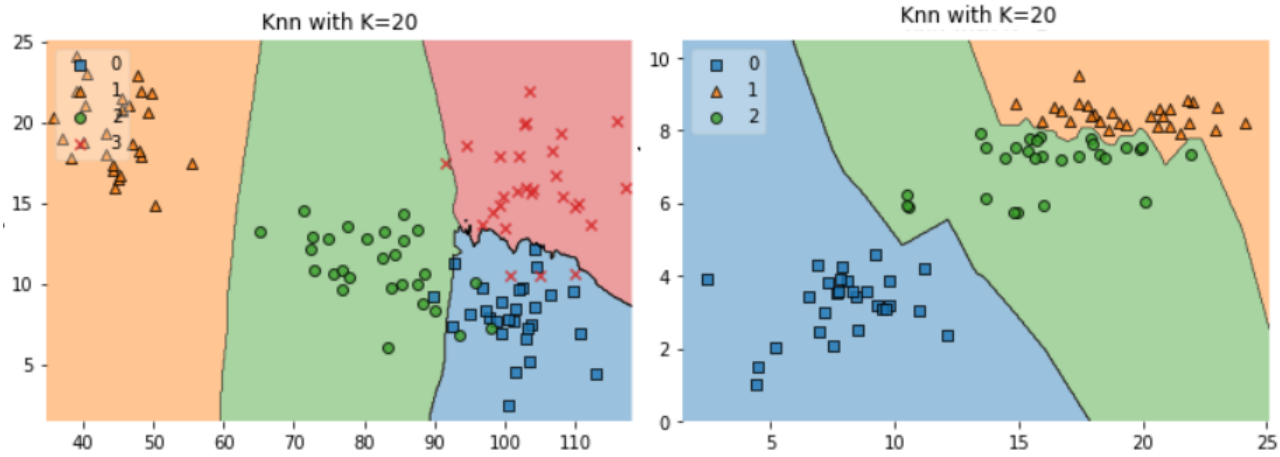


Figure 4.4: In the right side, clean (0), FGSM(1) and JSMA(2) samples were applied with AS Filters and using KNN[225] with the Histogram average value of DI and euclidean distance value,Similarly in the left side, clean (0), FGSM(1) and JSMA(3) and DF(2) samples were applied with AN + AS Filters s effects. I can see DF, Jsma is overlapping there.

Data Manipulation attack	Sequence for Dataset (MNIST)	Sequence for Dataset (CIFAR)	Reason
FGSM	BS+BS	Greysale+AN+AN	perturbation which added in image is not an edge, so edge-preserving algorithm remove these, so differences of before after adversarial image inputs are higher than non=adverse image
JSMA	TN+BS	PX+grey-scale+BS	JSMA extend some edge which reduced when I do thinning
CW	TN+TN+TN or GS+GS+GS+TBS	GS+greyscale +GS+TBS +TN+TN	In CW object edge getting thinning and blur,so applying more blur and thinning algorithm difference will amplify
DeepFool	GS+AN+GS+AN	GreySale+AN+AN+AN	Deepfool create few pixel arround border to effect the model,using additive noise boost this effect and I can have a difference

Table 4.1: Different ilters for different adversarial attack type

Figure 4.5: 4 Filters applied to 5 types of the adversarial set with a clean sample set for the MNIST dataset. In X-axis, no of each sample shows and Y axis, Before and after effects histogram average has been illustrated. From this small set I can see, different Filters has different effects for different attack types

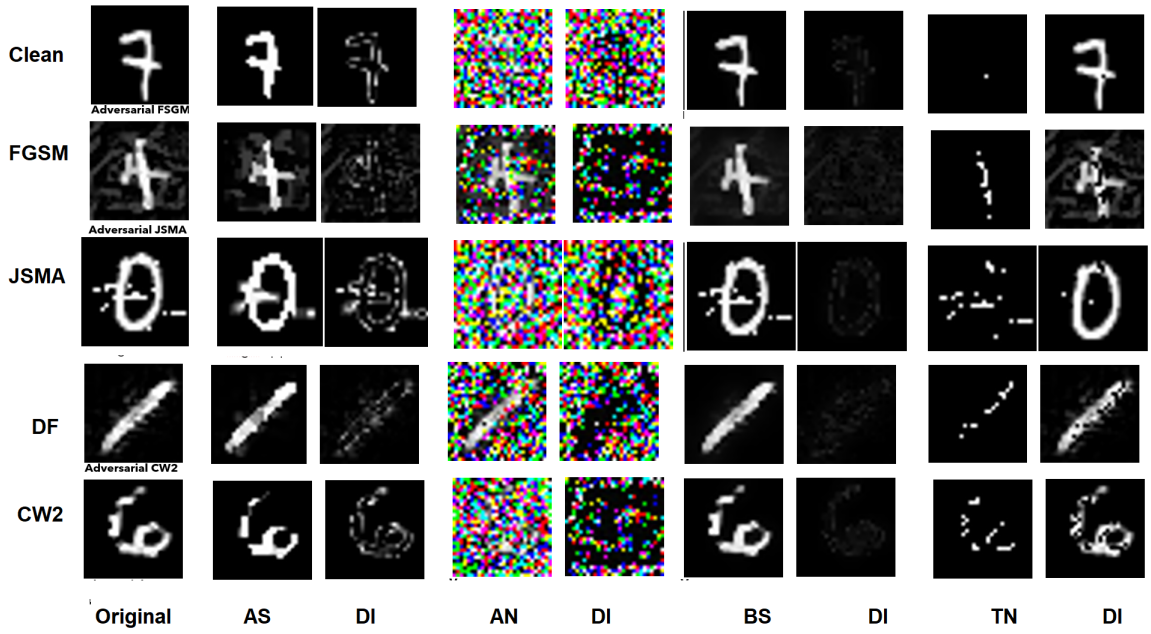


Figure 4.6: Effects of Several Filters techniques on MNIST dataset, after applying different Filters

processed images can be measured by different metrics (E.g: Histogram difference). These metrics will have significantly different values from the clean dataset's same metrics. We can differentiate between non-adversarial and adversarial images using these metrics. It could be possible that one unique sequence of edge-preserving and other Filters could make it more distinguishable than other sequences, thereby providing metrics of such differences to be used as a threshold value to classify adversarial and non-adversarial images.

4.2 Ensemble the Input Filters

There are numerous Filters that exist. Each can perform different operations in images. So, for a specific type of attack, a specific type of Filters sequence will work better. From my empirical observation, we can select a diverse set of unique Filters's such as BS, thinning [66], AN, blur [203], sharpen [67], thickening [66], and AS. These techniques and their combinations are best suited to maximize enhancement between the difference image (DI)'s of adversarial and clean input. It can be observed in Figure 4.1 that, for the MNIST dataset, FGSM method tends to add

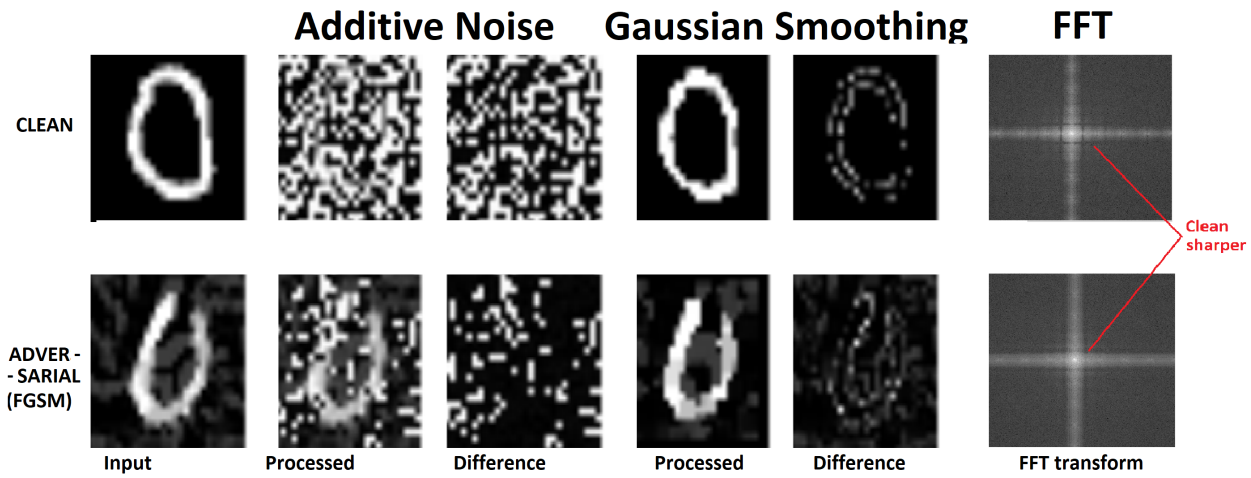


Figure 4.7: Here, the transformation of additive noise and adaptive smoothing filter and the difference between original and filtered images are illustrated. We use the histogram value from the difference image as my feature metrics. On the right side, fourier transformation effect and SNR values difference of adversarial and clean input value. We can also notice a pattern in the red arrow between adversarial(FGSM) and clean images.

pixelate noise around the object in the image, while JSMA seems to add along with the object borders, and CW tends to lose some erosion, whereas DF adds a very small amount of perturbing. Therefore, I assumed that sharpening can help to highlight JSMA attacks, performing blur will have more effect in CW, and AS will have a good effect on FGSM attacks. In figure 4.4, I can see different Filter Set can work better for different attack types. So, I selected these Filter Set, and found a combination of these Filter Set that performs better. If I consider different datasets (e.g., ImageNet, CFIAR), I have to consider different Filter Set, thus I select variety sets of Filter Set, which can perform required all basic operations.

Filter techniques can be seen as input transformation/conversion techniques, which can modify any input to another input where specific properties of the original information are modified/transformed. We used approx 50 filters in my experiments, such as adaptive smoothing (AS), additive noise (AN), bilateral smoothing (BS), Gaussian blur, sharpen operation, thickening operation, Fourier transformation, Laplace transformation, log-polar conversion, wavelet conversion, etc. In table 1, a list of filters used in my experiment for adversarial input detection are provided. Figure **??**, we presented the filtered output of the MNIST image. We can see the difference images have distinct characteristics for adversarial and nonadversarial images. In my

Filter Family	Code	Filter Name	Attack Method			
			FGSM	BIM	PGD	JSMA
ANALYTICAL	FT4	**Distance**	50	70	70	50
	FT10	**Morph**	75	70	70	60
EDGE Base	FT5	**Canny**	75	75	75	70
	FT11	**Sobel**	50	75	50	75
	FT16	**Gaussian edge**	75	70	75	75
Noise Add		**Median Blur**	65	60	65	55
		Average Blur	70	70	70	70
	FT1	**Gaussian Blur**	70	65	70	60
	FT7	**Gaussian Noise**	60	50	60	65
		Dilation	70	70	70	75
		opening	75	70	75	65
		Closing	70	50	70	75
		SaltAndPepper	75	75	75	75
	FT13	**SierraDithering**	70	70	50	**70**
Noise Reduce	FT12	**Erosion**	75	55	75	70
	FT0	**Sharpen**	70	70	75	50
	FT6	**Shrink**	50	50	55	55
Texture		**OilPainting**	75	50	75	70
		Pixellate	50	70	50	50
	FT14	**Wavelet**	70	75	70	50
	FT2	**Gabor**	50	70	50	50
	FT8	**Census**	55	55	55	70
Transform		**Top_Hat**	70	50	70	75
		BlackHat	70	50	75	55
	FT9	**Lapalce**	70	75	60	75
	FT3	**Fourier**	55	55	75	75
		Exponential	50	50	50	75
	FT15	**Log-polar**	50	55	75	65
		Mirror	55	50	75	60
		TopHat	55	50	55	75
		WaterWave	75	50	75	70

Table 4.2: List of the filters and their accuracy against different attack (250 adversarial inputs for each attack type) on MNIST dataset (ACOORD.net library used for experiment) (Note: here we only provided successful detection rate).

study, we extracted features from input data such as Signal to Noise (SNR), Peak Signal Noise Ratio(PSNR), Root Mean Square Error (RMSE), Histogram[96], Local Binary Pattern(LBP).

4.3 Generation of filter Sequence

The phenomenon which we briefly described previous sections assures us it is sufficient to detect adversarial images that have distinguishable noises because if an adversarial image doesn't have distinguishable noises they hardly exist in the physical domain. It also shows that filters from different types are better for detecting all types of attacks, and we don't need to test against all attack types. We also need a balanced sample set of training datasets as we showed that these filters are transferable. Based on these, we decided to select a set of filters in a specific sequence dynamically for each input data. There will be an optimum certainty that one of the chosen filters can detect the adversarial traits in the input data by the fastest possible feasible time and will be independent of ML or AA types and immune to adaptive attacks. In the table 4.3, we can see the shorter sequence of filter has the same accuracy but the lower computational time. In that table two sequences result over different dataset and different attack type was presented. It is also visible that the overall accuracy of detection rate and specific attack type accuracy is different. For example, in sequence 1, total accuracy is 97%, but for FGSM, it is 100%.

Sequence 1	**FT1-FT2-FT3-FT4-FT5-FT6-FT7-FT8-FT9-FT10-FT11-FT12-FT13-FT15-FT16-FT17 (time : 89s for 1000 test image)**															
Dataset	MNIST				EMNIST				FMNIST				CFIAR			
Accuracy	**98 (10 types of attack)**				**98 (10 types of attack)**				**98 (10 types of attack)**				**94 (6 types of attack)**			
Attack Types	FGSM	JSMA	CW	DF	FGSM	JSMA	CW	DF	FGSM	JSMA	CW	DF	FGSM	JSMA	CW	DF
Accuracy	100	100	90	90	100	100	94	90	100	100	90	90	98	95	80	70
Error Margin	0	0	7	5	0	0	4	9	0	1	6	5	8	5	12	10
Sequence 2	**FT1-FT4-FT6-FT8-FT9-FT13-FT15 (time :70s for 1000 test image)**															
Dataset	MNIST				EMNIST				FMNIST				CFIAR			
Accuracy	**98 (10 types of attack)**				**98 (10 types of attack)**				**98 (10 types of attack)**				**92 (6 types of attack)**			
Attack Types	FGSM	JSMA	CW	DF	FGSM	JSMA	CW	DF	FGSM	JSMA	CW	DF	FGSM	JSMA	CW	DF
Accuracy	100	100	85	90	100	99	90	90	100	99	90	90	97	95	75	68
Error Margin	0	0	7	5	1	1	6	9	1	21	6	5	9	15	12	12

Table 4.3: Smaller sequences have similar performance as long sequences, but take less computational time. Here, Sequence 1 is consists of all the filters and has the same accuracy for different AAs (here, four specific AA result from ten different filters provided) as shorter sequence (2) but with less time.

Due to the process flow, the filter sequence with the same filters has a different

computational time. As a better performing filter reduce the number of the image comes to the next filter in sequence, But it is also observed that different filter has different computation time. So it is unwise to put the best performing filter first in the sequence. There are two types of processing time applied here: file transfer/read time and filter processing time for each input. File read time is equal for all filter, but the different filter has different process time. Lets assume, File transfer/read time is T_f, Filters input processing times are $t_1, t_2...t_n$ w.r.t to Filters $FT_1, FT_2...FT_n$. So each filter total consumes time for N number of input:

$$Time_{FT_i} = N \times (T_f + t_i) \tag{4.1}$$

here, $i = 1, 23..n$

In a sequence filter can be any order, and each filters are overlap their accuracy among adversarial samples. As example, Filter A and Filter B both have 50% accuracy, but cumulatively accuracy can still be low as 50% as they are able to detect same adversarial samples. Lets assume in a sequence $FT_1 - FT_2 - FT_3... - FT_9$, adversarial detection probabilities are $P_1, P_2, P_3, ...P_n$. So approximate time for the sequence will depend on the following equation

$$Time_{approx} = T_1 + (1 - P1)T_2 + (1 - P_1)(1 - P_2)T_3 + ... \tag{4.2}$$

$$Time_{approx} = \sum T_n \prod_{i=1}^{n-1}(1 - P_i) \tag{4.3}$$

We need multiple solutions because we can not use the same sequence of filter for every input. This will create an opportunity of adaptive attack. A sequence could be any length. Search for optimal set of sequences are massive computational time, if we do exhaustive search considering multiple objectives. That's why we will employ a multi-objective GA to search for the optimal set of sequences. For search filters, we need to consider different factors besides their accuracy. Based on our objective, our filters need to be fast, that's why filters order are important because separate order of the filter will consume different amounts of time. It is preferable then

our solution is time efficient. According to PH6 and PH7, we need to make sure that there is a diverseness of filter type in our set of filters. If we have a diverse type of filter that out filter will work against the untested attack and thus reduce the zero-day vulnerability. In our solution, we have to deal with two kinds of diversity:

1. Insider Diversity (Diversity of filter family in a sequence): As illustrated in figure 4.13, if only a few filter families are represented in a sequence, we will see lack of insider diversity. It requires different filter families to be effective against a different set of attacks. It is not possible to test all of the attacks, so for the safeguard, it is better to have a sequence from different filter families. Also, filters from the same family work the same way, so multiple filters present from the same family do not increase efficiency much higher.

2. Set Diversity (Diversity of the filter in the set of sequence): We will make a dynamic selection of a sequence from a set of filter sequences. If this set of sequences were made of the same filters that the purpose of dynamic selection will be lost. An attacker can assume which filters have been in use and employ back pass differential attack [32], which can bypass filter techniques. So we need to make our sequences unique. In the illustrated figure 4.13, an example of set diversity is provided.

In summary, we need a time-efficient,to produce a reliable performance and a unique set of sequence. Our designed multi-objective GA can achieve all these criteria. The purpose of using a genetic search is to find a diverse sequence of filters detecting AAs with maximum accuracy while each filter is having different characteristics and capabilities when deployed such a sequence adaptively (interchangeably) in a ML that will be unpredictable to attackers compared to a static ensemble of well-known filters. So the GA will find not only the best filter ensemble, but also a set of diverse filter sequences in multi-objective Pareto-front. In section 4.3.2, we detailed how our designed GA is achieving this. In order to implement this integrated approach, we proposed the AEF framework (illustrated in figure 5.5)) that will select a set of filters in a specific sequence dynamically for each input data. There will be an optimum certainty that one of

the selected filters can detect the adversarial traits in input by the fastest possible time and will be independent of a ML and immune to adaptive attacks. Collection of these filters will be generated using a multi-objective variable-length diversity sensitive genetic algorithm (A top-level view of GA illustrated in the figure 4.8).

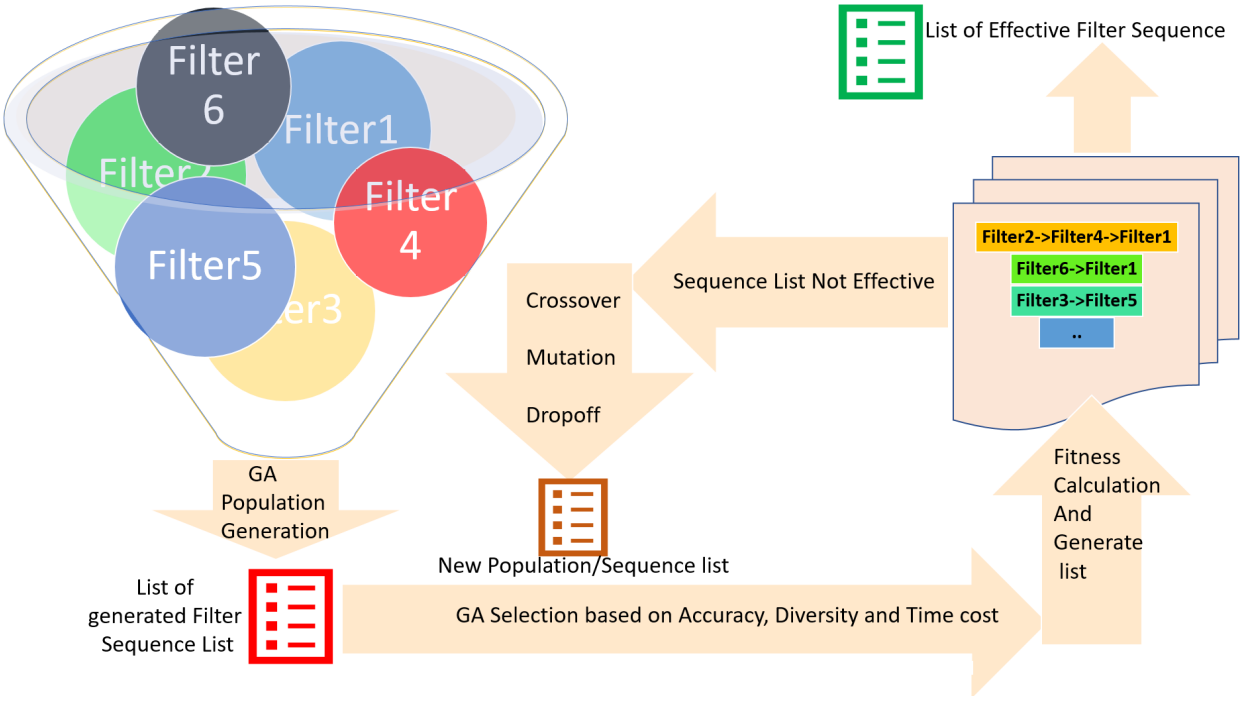


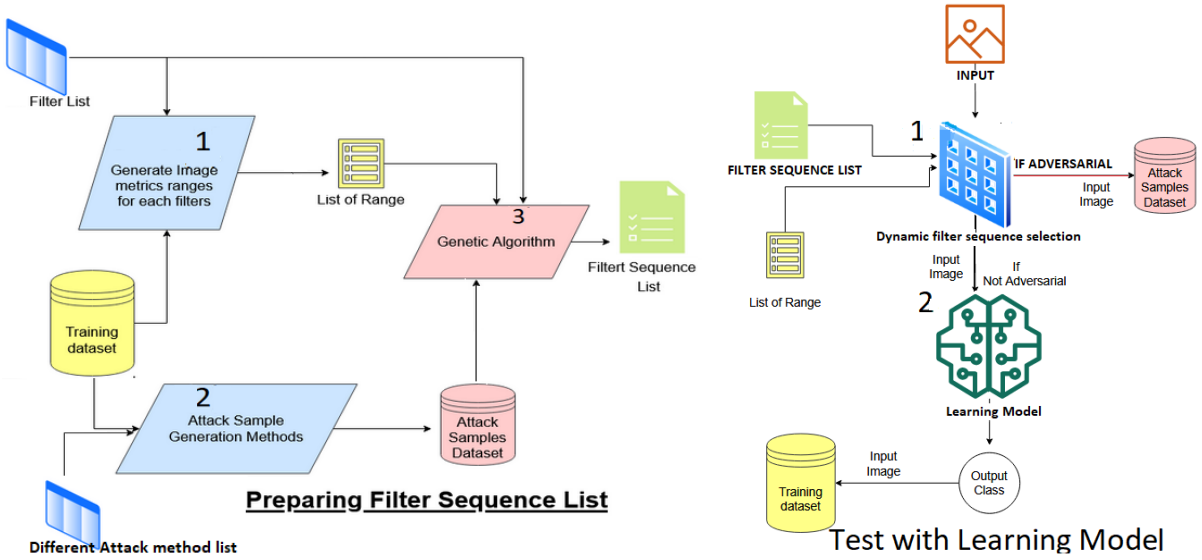
Figure 4.8: Genetic Algorithm (GA) for generating filter sequence list

If we use a random sequence selection instead of GA, we will get the low performed sequence as presented in figure 4.11.

4.3.1 Filter Sequence Search Space

If we have N number of filter, than total possible number of sequence will be our search space. We can formulate our search space by below equation:

$$Search_{space} = \sum_{k=0}^{N} \frac{N!}{N! - k!} \tag{4.4}$$



(a) The basic flow diagram for the filter sequence list generates process. It is visible that we don't need any ML information. We only need a training dataset(or a balanced part of the training dataset).

(b) Basic flow diagram when our input filters(1) running to protect a ML(2). (Note: We are not sending modified input to the ML, we are sending original input to the ML if we detected that as clean)

Figure 4.9: Flow chart in different stage of operation.

For 50 filter size of our search space has approx $8.26X10^{64}$ search items. If we don't consider time efficiency then we don't need to order in a combination of sequence (For different order a sequence accuracy remain static but time efficiency change). So search space will be less, since order in a combination is not important. For single objective (Only detection rate as F1score/accuracy) our search space will be formulated by below equation:

$$Search_{space_{Accuracy}} = \sum_{k=0}^{N} \frac{N!}{k!(N!-k!)} \tag{4.5}$$

For 50 filter size of our search space is approx $1.126X10^{15}$.

We can optimize our search space by limiting the minimum sequence length and maximum sequence length. Than our equation will be :

$$Optimized_{Search_{space}} = \sum_{k=min}^{N-max} \frac{N!}{N!-k!} \tag{4.6}$$

Here, min and max are the minimum and maximum length of a filter sequence. In our experiment, we have 17 filters and minimum length were 6, our optimized experimental search space has consist of 9.66^{14} search item.

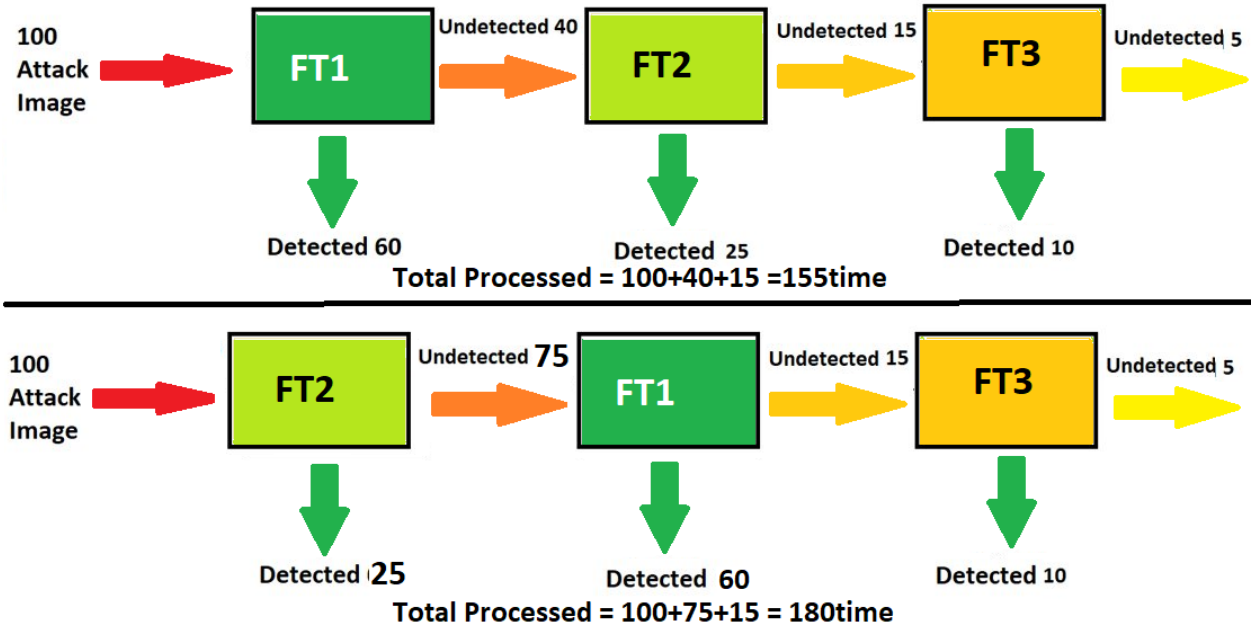


Figure 4.10: Both sequence has 95% accuracy(Not F1) and same size but top sequence has less time.Steps of adversarial input, detecting sequential by each filter is presented, we can see that reordering these sequences will change the number of undetected images for the next filter. So for different sequence processed time varies but detection accuracy remain same. (Note: here a simple example used for illustration, where FT1 and FT2 has no overlap detection)

4.3.2 GA Methodology

We have three steps in our experimentation process as shown in figure 4.9. First, we created the ranges of each filter as seen in figure 1 marked (1), then we generate attack samples for our testing purposes as marked (2). Using these test-samples we implement a GA to search for appropriate filter sets that are suitable for different attack types. In 4.9, we illustrated our basic flow diagram for the testing process. We first select a random sequence and used it as our input filters (marked 1) to detect an AA from the learning model/ML (marked 2). If our filter detects it as adversarial, we add it in our attack samples and if it is a clean image, than we send it to the ML and also add it in our training dataset.

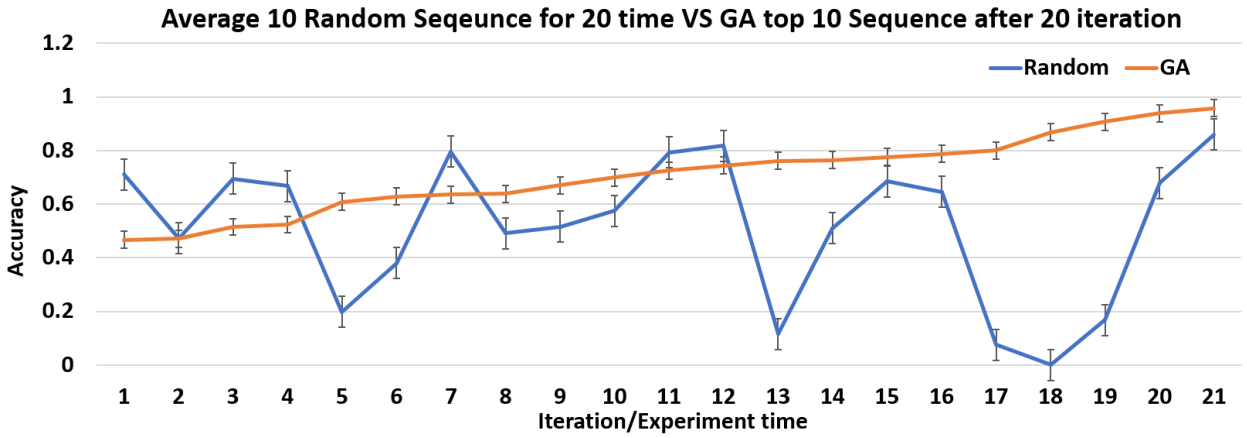


Figure 4.11: Here, X-Axis is the generated sequence, and the y-axis has the detection accuracy of the sequences. In blue, we represented the average random sequence, and in orange, we have sequences after 20 iterations of a GA for the SHAPE dataset using 12filters. We can see that randomly select a sequence has a meager chance to have good accuracy, whereas choosing a sequence from GA results will guarantee higher accuracy.

Generate metrics ranges for each filters

Algorithm 1 Generate min,max metrics for a filter (FT)

let, $min = 0$, $max = MAX_{INT}$, $STD_{dev} = 0$,$SUM = 0$,$SUM_2 = 0$

$sample_{dataset}$ has N number of $training_{data}$,

for Each $training_{data}$ td in $sample_{dataset}$ **do**

value = metrics(td) {Metrics function for filter FT}

if min<value **then**

min=value

end if

if max>value **then**

max=value

end if

$SUM = SUM + value$, $SUM^2 = SUM_2 + (value \times value)$

end for

$STD_{dev} = \sqrt{(SUM_2 - (SUM \times SUM)/N)/(N-1)}$

$min = min - STD_{dev}$, $max = max + STD_{dev}$

return min,max

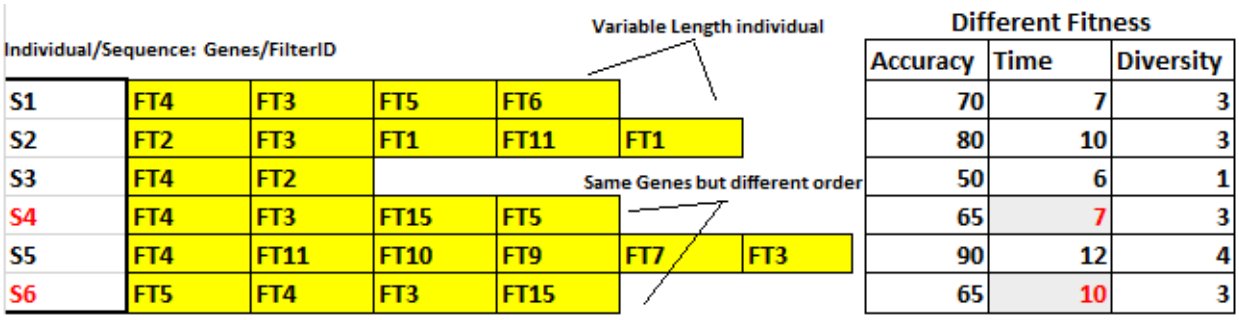


Figure 4.12: Population of GA represented, Here each individuals are variable length, In crossover and mutation duplicate occurrences were removed. Different order of sequence provide same accuracy but different time as we can see for sequence 4 and 6. Three objectives were presented we can see longer length does not guarantee accuracy or diversity

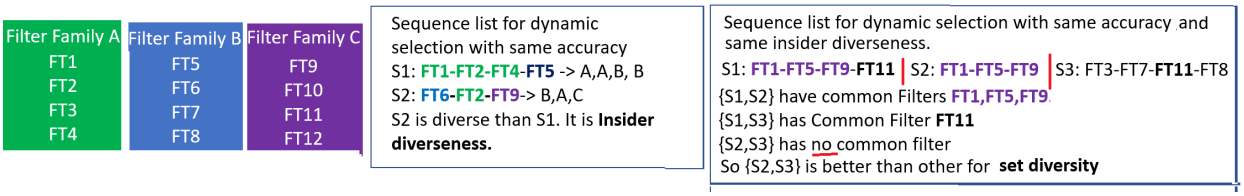


Figure 4.13: 'Insider Diversity' and 'Set Diversity' Explanation, In the Left side, we can see two sequences where one has insider diversity one has not, and left side, we can see which set has the set diversity.

First, we need to process the ML's training dataset and run all the filters and calculate their metrics value. We ran a randomly picked balanced subset of inputs from the data set and applied each of our filters and gather their Min, Max. Using algorithm 1 for each filter.

Now, if Mean is $\bar{\mu}$, Standard deviation is σ, Our lower range (L_r) and upper range(U_r) calculation denoted by equation 5.2 and 5.3

$$L_r = Min - \frac{\sigma}{\bar{\mu}} \tag{4.7}$$

$$U_r = Max + \frac{\sigma}{\bar{\mu}} \tag{4.8}$$

Using equation 5.2,5.3 for 17 filters, we generated list of upper and lowerange. In figure 5.2 we can observe that most of the attack samples SNR values are outside upper and lower range of clean samples.

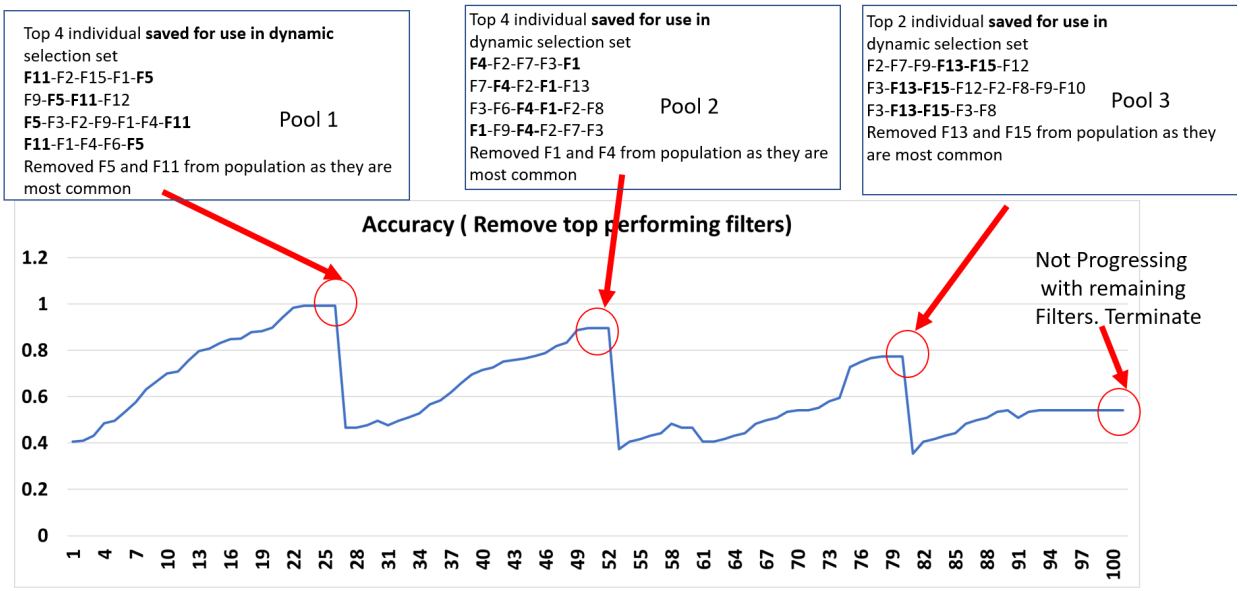


Figure 4.14: Here, X-Axis is the iteration number, and Y-axis has presented the average of $\alpha(s) \forall S$. We can see that some of the filters dominate at the top whenever we reach a local optimum. We save these sequences and remove these filters from the rest of the population. The immediate effect of the average accuracy dropped, but that picked up again. We continue to drop off dominated filters until the x-axis is stuck in a lower optimum than the threshold.

Algorithm 2 GA algorithm for Sequence Search(filter Set :FS)

1: Minimum accuracy =M_i,

2: Get metrics set MS from all training data (Clean+Adversarial) for all filter F in FS using algorithm 1

3: Generate N random sequences from FS -> population P

4: **while** MAX iteration reached or global optima reached **do**

5: Generate fitness value FV all Sequence in P using equation 5.10

6: Run suboperation for all Sequence in P using equation 4.18

7: Sort all Sequence in P based on fitness value

8: **if** Local optima reached & average fitness > M_i **then**

9: Do dropoff using algorithm 4 dropoff(P) ->SEQ_{POOL}, SEQ

10: $P = SEQ$, $SEQ_{POOL} -> Collection_{seq}$

11: goto step 5

12: **end if**

13: Remove lower half of P

14: Do crossover with first half using PMX algorithm and add the children in P

15: Do a inversion mutation

16: **end while**

We apply MOGA for searching a set of filter sequence which can detect maximum attack samples. Our GA has the following characteristics:

1. Our MOGA has a variable-length chromosome. As the optimal filter sequences could be any length, we need to make our GA multi-variable size supported.

2. We have 3 objectives to fill (accuracy, diversity and time) as mentioned in section **??** and detailed in section 4.3.2.

3. Our end collection of sequences set needs to be diverse; that's why we have a drop off operation (described in section 4.3.2) when we reach local optima.

4. To find the best time conserving order of a sequence, we have a sub-operation.(described in equation 4.18)

5. We prefer to have a simpler sequence. That's why the added penalty function prioritizes the smaller length sequence than the larger length (described in equation 5.9).

6. To faster the GA, we used adaptive weight values, prioritizing objective based on the GA average fitness. (described in equation 5.10)

Encoding and Individuals First, we encoded all filters according to table 4.2. Here, 17 algorithms were assigned sequence number $FT1, FT2...FT17$. In the leftmost column, the class type of these algorithms is mentioned. These filters are our genes. We will create our individuals/chromosome using these genes.

Using the filters encoded in table 4.2, we generated the population by random sequence generation. So each is consist of different length of filters. We remove multiple occurrences of filters in a single sequence. In figure 4.12, the encoding of filters are illustrated with different examples.

As example a random sequence $FT2FT5FT11FT12$ means **Blur -Census - Morph - Canny**. That way, we generated multiple lengths of sequences as our initial population.

Fitness Function

We have three objectives. they are Accuracy (α), time to detection(β) and Insider diverseness(γ) of filters in sequence. Accuracy is the success rate of detection by the filters.

We used F1 score as Accuracy value. We calculate F1 score using detection rates as described in below. Here, number of filter $= n$ and $S = f_i, f_i, .f_n$ is the sequence

True positive $TP = \sum_{i=0}^{n} TP$

False positive $FP = \sum_{i=0}^{n} FP$

False Negative $TP = \sum_{i=0}^{n} FN$

$$Precision = \frac{TP}{TP + FP} \tag{4.9}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.10}$$

So Detection accuracy F1 score is

$$\alpha(S) = F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{4.11}$$

For time, if each filter take t_i time, then total time to detection δ_t can be calculated by

$$\beta(S) = \sum_{i=0}^{n} t \tag{4.12}$$

For insider diverseness, for each filer $f_i \in F_i$, here F is the filter family and f is the filter, S is the sequence.

$$\gamma(S) = \frac{\sum f \in F | f \cap S| > 0}{\sum F} \tag{4.13}$$

As example,we have total 1000 adversarial samples and 1000 Non-Adversarial Sample. For each filter in the sequence, we see if the adversarial images metrics (ex:SNR) are outside the range we stored for that filters lower range (L_r) and upper range(U_r). As for $S = FT2 - FT5 - FT11 - FT12 =>$ Blur -Census - Dither -Canny. Here $n = 4$ and other metrics are

Filter: Correctly Detected (TP_i): Wrong detection(FP_i): Remaining(FN_i): Time (T_i)

$FT2 : 220 : 05 : 1000 - 200 = 780 : 0.03s$

$FT5 : 110 : 15 : 780 - 110 = 670 : 0.02s$

$FT11 : 200 : 10 : 670 - 200 = 470 : 0.015s$

$FT12 : 220 : 10 : 470 - 220 = 250 : 0.01s$

For the above example it is, $TP = 750, FP = 40, FN = 250$ so, $\alpha = 0.8380$ from equation 5.4, Time is $0.03 + 0.2 + 0.15 + 0.01 = 0.75s$.

As for Blur-Census-Dither-Canny. Here, Blur and Dither are from same class, and others are from 2 different class. So diverseness is $\frac{2+1+1=4}{6} = 0.66$. We Normalized all three objective data using equation 5.7

$$X_{sc} = \frac{X - X_{min}}{X_{max} - X_{min}}. \tag{4.14}$$

We inversed the Time data, so we have to maximize all of the objectives. Our fitness function denoted by

$$\max(f(S)) = ((\alpha_n(S)), (\beta_n(S)), (\delta_n(S))) \tag{4.15}$$

where, α_n is normalized accuracy, $\beta_n$is normalized inverse time, δ_n is diverseness factors. We have penalty function to prioritize simpler solution and weight values to speed up the GA process. Let assume, Weight Value, $W_0 = 0.90 - \frac{1}{100 - iteration_{number}}$

and Weight Value, $W_1 = 1 - W_0$

We observed that, in the beginning α is low and after a certain iteration γ gets lower. We use W_0 for α and W_1 for γ.

In the figure 4.20, we visualized the effect of weight to speedup the GA process.

For penalty functions we need below parameters

Length of Best fitted individual in previous iteration $|\max_{f(S_i)} \in \forall(S)|$

Size of current Sequence = $|S|$

Total number of filters = $\sum_{i=0}^{n} |f| \in \forall F$

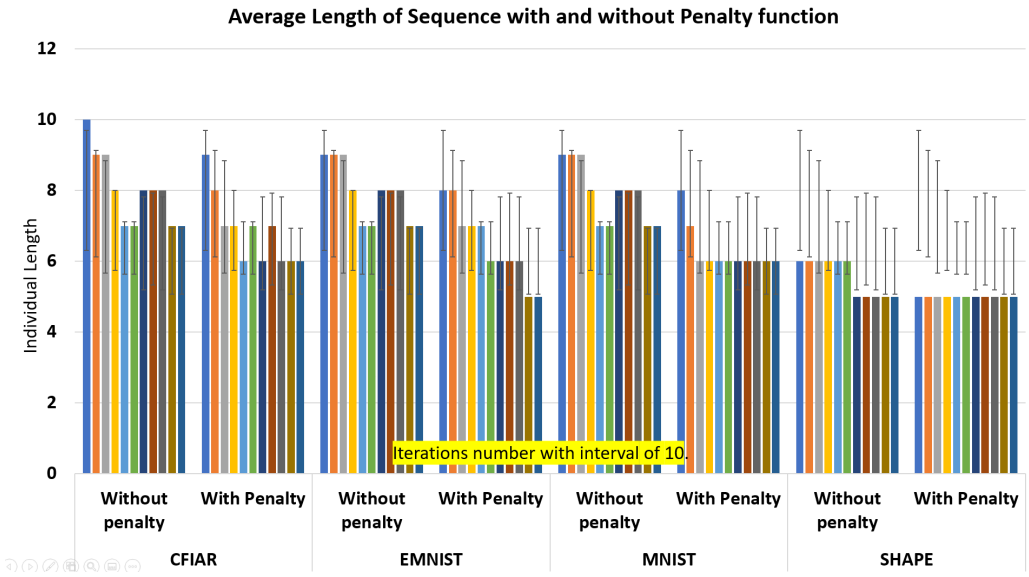


Figure 4.15: Here X-axis has the iteration number, and the Y-axis has the average length of the sequence. We can see with a penalty, the average size of the sequence tends to smaller. This experiment was done with 12 filters with 4 different datasets.

Equation for penalty function value can be denoted by

$$pf(S) = \frac{f(S)}{100} \times \frac{|S| - |\max_{f(S_i)} \in \forall(S)|}{\sum_{i=0}^{n} |f| \in \forall(F)} \tag{4.16}$$

So from equation 5.8, fitness for S is

$$f(S) = \sqrt{(\alpha_n(S)^2 \times W_0 + \beta_n(S)^2 + \delta_n(S))^2 \times W_1} - pf(S) \tag{4.17}$$

In figure 4.15, it is visible that with the penalty function, length of sequences are lower than without penalty functions. That way, our GA prioritizes to search smaller sequences. Also, a shorter time is co-related with sequence length. So $\beta(s)$ also affects having more straightforward sequences.

We created the 3D Pareto front using the three objectives from each individual. A non-dominated rank is assigned to each individual using the relative distance in 3D space. In figure 4.16 and 4.19, some individuals fitness functions are presented. For each individual, we ran another sub-operation to find the lowest time of the sequence order. This sub operation is done by

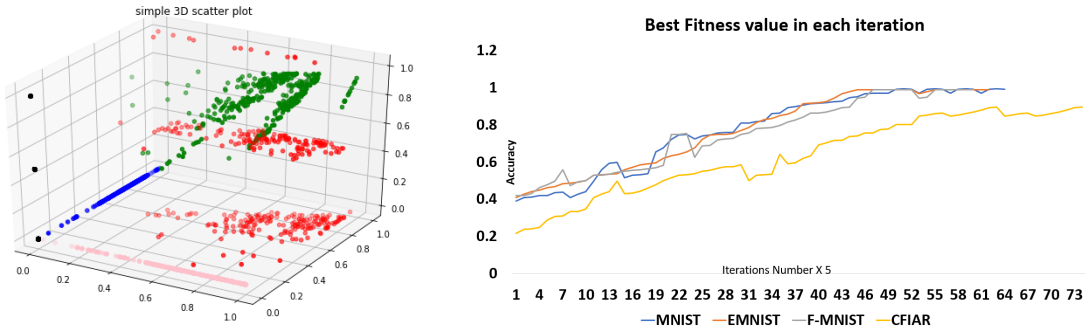
random search or genetic search for the fastest combination of filters in that individual/sequences. For a Sequence 'S', we take all filter $f \in S$ and try $\binom{f}{k}$ to create $S_{i=0..k}$ Sequences and return the sequence with best $\beta(S_i)$. We replace current S with S_i in the population. so equation for sub operation is

$$S_{new} = \min_{\beta(S_i) \in \binom{f}{k}} f(S) \tag{4.18}$$

As example, if our $S = (FT9 - FT3 - FT6 - FT2)$ we will try K number of combination using $FT9, FT3, FT6, FT2$ like

$S_1 = (FT3 - FT9 - FT2 - FT6)$,$S_1 = (FT2 - FT9 - FT3 - FT6)$

$...S_K = (FT3 - FT9 - FT6 - FT2)$ and will replace the S by the S_i with best βS_i. If the individual's length is L, then the search space for sub-operation is $L!$. If $(500 \leq L!)$, than we do a simple random search. Otherwise, we do a simple GA for finding the best time optimized filter combination.



(a) Here, Pink is time efficiency, black is length efficiency, and blues are each individual's accuracy. Red points are 3d vectors of each individual, and greens are the Pareto front where no one is dominate

(b) Best fitness value as GA iteration (per iteration= X-axis value X 5) progressed for four different data-set. We can see that for CFIAR it need more iteration to reach fitness over 90.

Figure 4.16: MOGA Pareto front and performances illustrated.

Selection, Crossover and Mutation We used elitist strategy with rank Selection. We keep the best performing filters for next generation. We did PMX crossover as we need unique values in a sequence. An example of PMX cross over is

Parent 1: FT8 FT4 FT7 ***FT3 FT6 FT2 FT5*** *FT1 FT9 FT0*

Parent 2: FT0 FT1 FT2 FT3 FT4 FT5 FT6 FT7 FT8 FT9

Child 1: FT0 FT7 FT4 ***FT3 FT6 FT2 FT5*** *FT1 FT8 FT9*

Here, from parent one we pick a random part and test of the part we took from other two parent using PMX crossover algorithm[41]. In PMX crossover, parent 1 gives a genetic element, and the identical swath from the other parent is sprinkled about in the child. After that, the surviving alleles are lifted straight from parent 2.

Algorithm 3 PMX Selection algorithm (Parent 1, Parent 2)[41]

1: Randomly select a part from parent 1 and copy them directly to the child. Note the indexes of the segment.
2: Looking in the same segment positions in parent 2, select each value that hasn't already been copied to the child.
3: **while** For each of these values: **do**
4: Note the index of this value in Parent 2. Locate the value, V, from parent 1 in this same position.
5: Locate this same value in parent 2.
6: If the index of this value in Parent 2 is part of the original swath, go to step 3. using this value.
7: If the position isn't part of the original swath, insert Step A's value into the child in this position.
8: **end while**
9: Copy any remaining positions from parent 2 to the child.
10: **return** Child

For mutation, we did a inversion mutation,where one gene of an individual replaces by another gene. In our method, we pick a random index and random filter from $\forall(\nexists f \in S)$. As example, for sequence

$S = FT2 - FT3 - \mathbf{FT5} - FT8 - FT9$,

we pick random index 2, and random filter FT10, so new sequence

$S_{new} = FT2 - FT3 - \mathbf{FT10} - FT8 - FT9$.

Drop-Off Operation

Algorithm 4 drop-off algorithm ($SEQ = Seq1, Seq2, ...Seq_n$, Remove K item, Partition P, SEQ_{POOL})

1: Sort the Sequences based on their fitness values

2: first n/P sequences -> SEQ_{top} ->SEQ_{POOL}

3: $Item_{number} = 0$

4: **while** $Item_{number} \neq K$ **do**

5: most common item T in SEQ_{top}

6: remove T from every Sequence in SEQ_{top} & SEQ

7: $Item_{number} = Item_{number} + 1$

8: **end while**

9: **return** SEQ and SEQ_{POOL}

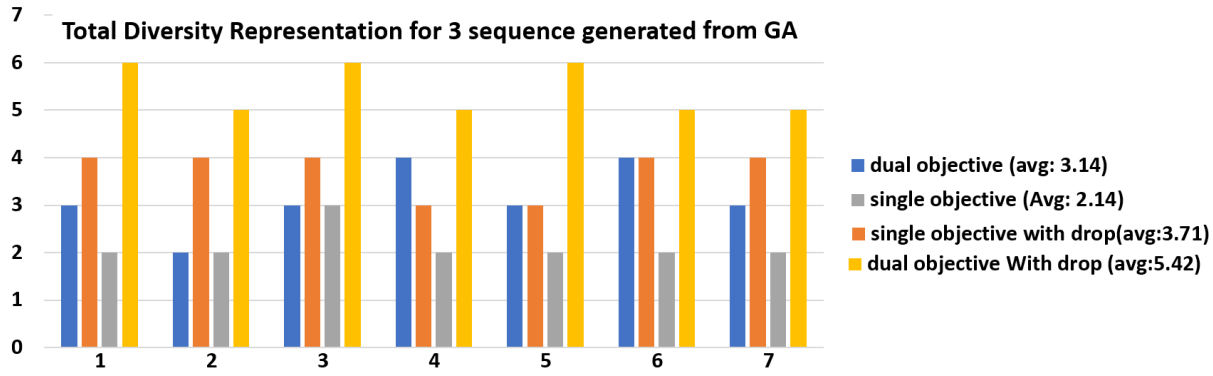


Figure 4.17: Y-axis presents the diversity value, and the X-axis represents the total number of experiments. The difference in diversity values from sequences generated by different GA method is illustrated. After GA generated sequence, we randomly picked three sequences seven times and showed their total diversity value. We can see with drop off functionality, we reach maximum diversity. Without drop off and diversity objectives, it failed to cover half of the family.This experiment was done in MNIST dataset

To maintain the diversity, landscape aware method introduces by [122] or a noise-based approach introduces by [11] can be used. Landscape conscious process consumes more computational complexity, and the noise-based system would be hard to encode in our problem, thats why we introduce a simple concept of dropoff to keep set diversity. We observed that only a handful of filters are dominated the GA. It makes the dynamic selection pool vulnerable as all of the sequences are made with common filters. An attacker can guess these filters in a white box

setup and use the BPDA attack to bypass this defense technique. That way, whenever we reached an optimum, we save the best performing Sequence as a distinguished pool and remove the most common filters in these sequences. We provided an algorithm 4 which move the good sequence as a pool of sequence for dynamic section and remove the dominated filter from the current population. In figure 4.14 this is illustrated, here we can see we get three pools of sequences. In the dynamic selection, we randomly select one sequences from each of these three pools to test an attack is adversarial or not. We have a threshold for dropoff number (D_n), if top K number of Sequence ($S_{i=0..k}$) has more than (D_n) common filter, we save $S_{i=0..k}$ in a pool, and remove common filters from all the Sequence in current population. Remove filters can be extracted from below equation

$$\exists f_{i=0..D_n} = S_1 \cap S_2 \cap ..... \cap S_k \tag{4.19}$$

In the figure 4.17, the importance of dropoff is presented. In that figure, we provided a diversity of 3 sequences each time. It seems in a single objective without dropoff, all of the sequences are from 2-3 classes. But with dropoff and with diversity objective, the sequences generated from GA are from 5-6 different classes.

4.3.3 Experimental Results and analysis

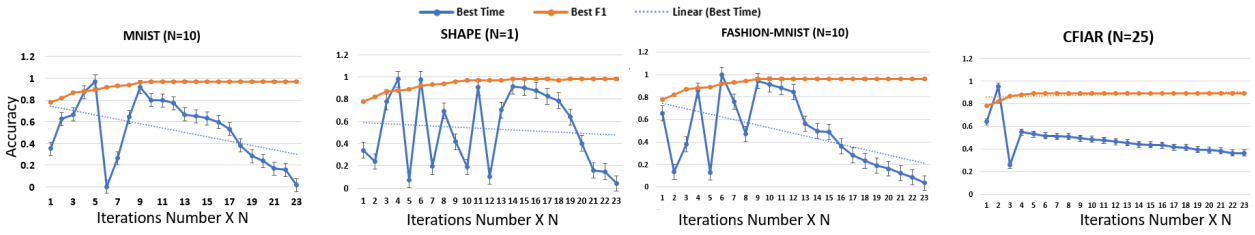


Figure 4.18: For N iterations best accuracy and best time-cost were plotted. Here we can see time and accuracy were not equally progressed. But after we get a good F1 score individuals time started to improved.

In table 4.4, we presented the first 15 individuals and their accuracy and time. The first column has a sequence of filters. We can see that different sequence has the same accuracy, but they have additional time cost. For example, the third row has sequences 11 and 5 with accuracy

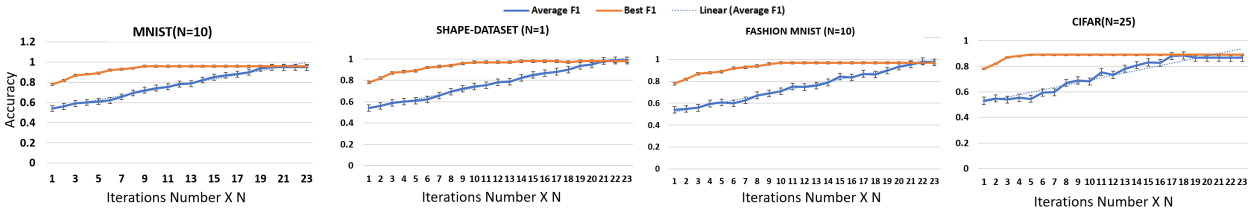


Figure 4.19: For N iterations, average accuracy and best accuracy was plotted. We can see that in the initial state when the sequence was utterly random, accuracy was below 50%, but after several iterations, it started to go higher

Figure 4.20: Here X-axis has the iteration number, and the Y-axis has fitness. We can see for MNIST without accuracy, GA requires 63 iterations to reach a fitness over 98. Whereas with the weight, we can get the same fitness by 40 iterations. This experiment was done with 12 filters with MNIST, EMNIST, F-MNIST, Shape Dataset, and CFIAR dataset.

98% detection rate by taking 0.45S. Whereas 7th row 15,8,3,7 has 77% accuracy and take 0.86s as time cost. In figure 4.16, we illustrated a chart of different dataset GA performances. Where we can see MNIST, EMNIST and F-MNIST are faster than CFIAR. While running the GA, we did 100 iterations, and we plotted average accuracy and best accuracy in figure 4.18. We can see blue as the best accuracy reached about 95% after 15 iterations, but the average accuracy came above 95% after 35 iterations. We need a set of sequences. So we need to run the iteration until at least 40 individuals have above standard accuracy. Because if we have 40 sequences for our ML, then the attacker probability to guess the sequence correctly will be $\frac{1}{40} = 0.025$. So run a successful adaptive attack will be computationally impossible. In figure 4.19, accuracy and time cost are plotted for all 40iterations. It is visible that the best accuracy and time cost are not co-related. We used multi-objective functions, so lower time-cost gets prioritized in the iteration. However, due to very small differences in the time cost between the individuals, it doesn't significantly impact iterations.

In table 4.6, we presented the comparison between the F1 score in test time when we use a

individual	F1	Time	Diversity
8, 14, 9, 13, 6, 15	0.97880597	0.741013	5
5, 7, 15, 11, 12, 14	0.97880597	0.451521	5
11, 5	0.97880597	0.363813	2
11, 7, 6, 4, 15	0.97880597	0.481844	4
11, 5, 14, 4, 8	0.97880597	0.415009	4
3, 8, 9, 15, 11, 5	0.96880597	0.866414	5
15, 8, 3, 7	0.790522388	0.686515	2
9, 8, 13	0.768656716	0.668462	2
15, 11, 12	0.768656716	0.514135	4
10, 14, 9, 12	0.751865672	0.694747	3
7, 3, 11, 2, 9	0.751865672	0.839804	3
8, 2, 9, 12	0.748134328	0.593654	3

Table 4.4: Accuracy,Time, and diversity After applied different Series of filter for MNIST dataset

	Acc	Testing Time
Naive Bayes	65	0.3
Random Forest	99	0.1
Neural Net 3	78	0.3
Logistic regression	65	0.2
GA+range(Our Technique)	98	0.04-0.09
Decision tree	99	0.09

Table 4.5: Metrics of Adversarial samples and clean samples from each filter used to train different ML and their accuracy

single objective and dual objective. Here, the sequences generated from the GA has experimented with a random set of adversarial and non-adversarial images. The test accuracy was dropped in both scenarios. But sequences in dual objectives (which were more diverse) have less drop than the single objective (which were less diverse). Here, we experimented multiple times F1, and provided the F1 mean of the results, and we also offered the standard deviation to evaluate the confidence value of these results.

Instead of using GA, if we use a random search or brute force that will also find good filter sequences, it would require higher search coverage. In figure 4.21, we illustrated the difference between search coverage between GA and random search. Here we only considered the single objective, so equation 4.5 is applied here. In our experiment, we needed a more extensive search coverage due to other objectives.

Dataset	Sequence with single objective			Sequence with Dual objective		
	F1 in GA	Testing F1	std dev	F1 in GA	Testing F1	Std dev
MNIST	99	87	1	97	93	1
EMNIST	98	85	1	97	92	1
FASHION-MNIST	95	89	1	92	90	1
CFIAR	90	72	1.5	88	83	1.25

Table 4.6: A comparison between single vs. dual objective (diversity and accuracy) GA sequences is presented. Single objective GA sequences have a larger drop in mean accuracy rate (F1-score) than the dual objective.

Detection Method	MNIST				CIFAR				Avg
	FGSM	JSMA	DF	CW	FGSM	JSMA	DF	CW	
RF based adversarial training[81]	0.96	0.84	0.98	0.66	0.64	0.63	0.60	0.72	0.77
KNN based learning [81]	0.98	0.80	0.98	0.6	0.56	0.52	0.52	0.69	0.73
SVM based learning [81]	0.98	0.89	0.98	-	0.69	0.69	0.64	0.77	0.81
Feature Squeezing[215]	**1.00**	**1.00**		-	0.20	0.88	0.77	-	0.77
Ensemble [15]	0.99	-	0.45	-	0.99	-	0.42	-	0.71
Decision Mismatch[141]	0.93	0.93	0.91	-	0.93	**0.97**	0.91	-	0.93
Image quality features [6]	**1.00**	0.90	**1.00**	-	0.72	0.70	0.68	-	0.83
AEF(Our Framework)	1.00	1.00	1.00	**1.00**	0.94	0.95	**0.96**	**0.94**	**0.99**

Table 4.7: Here, we provided a comparison with other adversarial input detection techniques based on Accuracy. On average, we outperform other methods. As examples, our methods work with 96% accuracy in the CFIAR data-set where the feature squeezing technique has 0.88% accuracy.

We also explored local search methods such as Hill climbing search. However, our preliminary experiments on Hill climbing show evidence that it will be stuck in local maxima after some iterations. It is possible that this problem could be solved using stochastic hill-climbing, random walks, and simulated annealing. But implementation of those is complex than Multi-objective GA. In GA, we have a better way to secure population diversity, which is hard to implement in local search. Some prior literature study, as example work of [97, 213] shows result comparison of GA vs. local search for a similar problem like our search problem, and GA offers better performance than local search. That is the reason we opt for GA instead of local search algorithms.

It could be argued that instead of doing a range check and GA, we can use another ML's to detect adversarial samples using the image metrics we generated in the first phase of our experiments. We tested with the 5000 adversarial samples and 500 clean samples metrics dataset (MNIST). In table 4.5, we showed the result by running different ML's and compare with our technique. It is visible that three-layer neural net and naive bayes perform poorly, but random

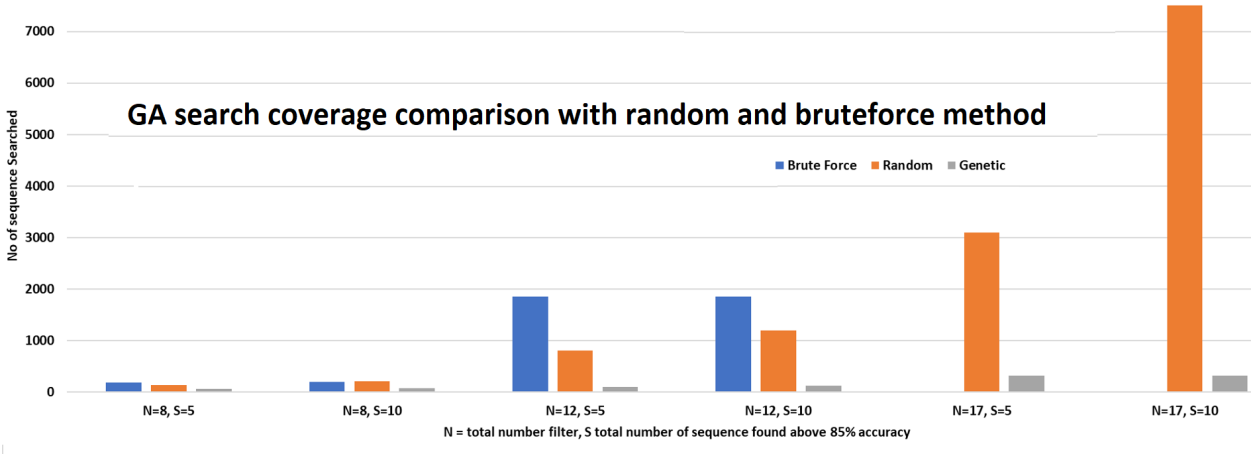


Figure 4.21: Here, We are searching S number of sequences with accuracy over 85% only using brute force and random forest search. For filter size 10, we weren't able to complete the brute force search. It is visible using GA higher accuracy filters can be found by less search from the bars. The random search may be suitable for a lower number of filters, but the random search takes a large coverage area than the GA search when the filter size is large.

forest and decision tree outperform us. But these techniques require more testing time, and also this technique accuracy depends on the adversarial datasets. Another important fact is they are also ML, so they are all vulnerable to standard AAs.

4.4 Output Filters

4.4.1 Negative Selection Based Filtering technique

We proposed a V-detector based NSA outlier detection strategy for each class label. NSA does not require adversarial examples to classify between adversarial knowledge and clean samples. Only a set of clean data are enough to generate a set of detectors for adversarial samples and based on the analysis given in section 2A, which shows NSA methods are more suitable for non-linear and limited positive samples OCC.

In the figure **??**, we illustrated the process of negative detector generation. A feature extraction technique (example : DNN based [**ono2018lf**] or binary feature extractions [**calonder2010brief**]) will mapping feature data for each label. These features will be used as self

data for the NSA algorithm. We will use different class labels features and each class will have unique representation spaces as their features are different. These feature types were selected based on which feature performs best to distinguish between one class label to other class labels. Our experiment used a simple conv net to extract features for each class.

In the figure **??**, we visualized how our NSA will interact with the ML model. ML model will provide the output class. Using that class and the input data, the NSA will use the same feature mapping technique for that class and try to match that class's negative detectors.

For overall framework concept, we illustrated our proposed workflow in the figure **??**. At first, as there is no data in the clean dataset, we have to assume the first few inputs are clean input. Based on these clean inputs and their class label, our NSA will generate the detectors for each class; when we have enough detectors, the ML model will provide the output class for that to the NSA when a new input comes. NSA will take the input and output class label and check that input for that class NSA detectors set. If the input is detected as an outlier, the process will stop. Otherwise, the information will send to a clean dataset to regenerate the detector set, and the output class will be given as the final class output. We presented the algorithm flow chart in the figure **??**, here NSA detector re generated when a clean input is detected. This way, we implemented relearning as new attack detectors are optimized with the new inputs and can perform better.

One of our proposed framework's possible limitations is that at the start, the clean data-set is empty; thus, adequate negative detectors are not likely to generate until sufficient clean inputs are collected. We can proceed with a sample set of clean data at the beginning to counter this problem; otherwise, we have to assume at first system will have only clean inputs.

Attack type	**Step1**	**Step2**	**Step3**	**Step4**
FGSM	0.86	0.92	0.902	0.93
BIM	0.89.0	90.0	90.0	0.90
PGD	0.92	0.94	0.95	0.95

Table 4.8: Detection of adversarial inputs which classified as MNIST class label '0' and with clean input of class label '0' , after sample size increased 100 in each step.

Attack type	NSA	OCSVM	IF	VAE	SOGAL	MOGAL
FGSM	0.93	0.99	0.93	0.65	0.5	0.5
BIM	0.90	0.98	0.91	0.66	0.5	0.5
PGD	0.95	0.99	0.92	0.50	0.5	0.5
MBIM	0.91	0.98	0.94	0.46	0.5	0.5
HSJ	0.88	0.55	0.41	0.65		
JSMA	0.9	0.56	0.8	0.83		
CW	0.96	0.42	0.66	0.52		
DF	0.91	0.45	0.76	0.55		

Table 4.9: detection of adversarial inputs which classified as MNIST class label '0' and with clean input of class label '0'

Attack type	MNIST Class '0'	MNIST '1'	MNIST Class '2'	MNIST Class '3'	CIFAR 'A
FGSM	0.93	0.92	0.93	0.92	0.8
Basic Iterative Method	0.90	0.90	0.91	0.91	0.8
PGD(random start)	0.95	0.92	0.92	0.90	0.7
MBIM	0.91	0.90	0.94	0.86	0.7
hopskipjump	0.84	0.65	0.80	0.65	
JSMA	0.9	0.96	0.8	0.93	0.6
CW	0.96	0.89	0.86	0.62	
DF	0.91	0.89	0.96	0.73	

Table 4.10: adversarial attacks on MNIST class label '0-3' and CIFAR 'dog' and 'airplane' class detection rate (each class has 200 positive and 200 adversarial samples which classifies as that class by a CNN)

4.4.2 Feature Selection

Negative Detector generation

We experimented with MNIST digit 0,1,2,3 and CIFAR dataset 'airplane' and 'dog' class. For example, We took the class label 'dog' as a positive class. All other classes and adversarial samples for the class dog are considered a negative class. We trained with 1000 positive data for each class. We used a simple convolution neural network to generate a feature set for each class. These features are used to create negative detectors for specific that class. To avoid the cold start problem, we put 1000 positive data for each class in the clean dataset. We used the V-detector algorithm for generating detectors.

Detection Method	MNIST				CIFAR			
	FGSM	**JSMA**	**DF**	**CW**	**FGSM**	**JSMA**	**DF**	**CW**
Random Forest based adversarial training[81]	0.96	0.84	0.98	0.66	0.64	0.63	0.60	0.72
KNN based adversarial training [81]	0.98	0.80	0.98	0.6	0.56	0.52	0.52	0.69
SVM based adversarial training [81]	0.98	0.89	0.98	-	0.69	0.69	0.64	0.77
Feature Squeezing[215]	**1.00**	**1.00**		-	0.20	0.88	0.77	-
Ensemble technique[15]	0.99	-	0.45	-	0.99	-	0.42	-
Decision Mismatch[141]	0.93	0.93	0.91	-	0.93	**0.97**	0.91	-
Image quality features [6]	**1.00**	0.90	**1.00**	-	0.72	0.70	0.68	-
Our Proposed Method	0.92	0.90	0.96	**0.82**	0.94	0.85		

Table 4.11: Comparison with other adversarial input detection technique based on accuracy (in our method, for MNIST average of class 0-3 was experimented and for CIFAR only tested with Airplane and dog class)

Defense Strategy	**Attack Sample Generation not needed**	**ML Model Modification not require**	**No Accuracy Reduction of ML**	**Not Vul to adap**
Adversarial Training	N	Y	N	N
Ensemble Method	N	Y	N	N
Pre-Processing Defense	N	Y	Y	N
Architecture Alteration	Y	N	N	N
Our proposed Method	**Y**	**Y**	**Y**	**Y**

Table 4.12: Advantages of our proposed method than other methods in terms of applicability

Experiments

We experimented with 1000 positive inputs and 250 attack inputs from each attack type. The V-detectors were initialize with 1000 clean input before. We used F1 score as Accuracy value. We calculate F1 score using detection rates as described in below. Adversarial samples detected as adversarial (True positive) TP, Clean samples detected as adversarial (False positive) FP, Adversarial samples detected as clean samples (False Negative) FN, We calculated

$$Precision = \frac{TP}{TP + FP} \tag{4.20}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.21}$$

So, Detection accuracy F1 score is:

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{4.22}$$

We also used other OCC techniques for comparison using pyod library[229]. We regenerate the V-detectors after every 100 clean inputs and using the randomized set for the clean and adversarial dataset in a batch of 200 to collect the results and provided the average F1 scores as the final result.

4.4.3 Result Analysis

In the table 5.2, we experimented the attack sample accuracy rate with V-detector generated using different number of clean samples. It is observed that after each iteration V-detector performance slightly increased. In the table 5.10, we presented results against different attacks types. We can see our defense works with different types of attack, as an example (FGSM, BIM) but result sharply decline for the CIFAR data set. We concluded that was due to not having a better feature mapping. It is observed that for different classes, different performance from V-detector as for JSMA MNIST class as 80% accuracy for class label 0 but same attack type class label '1' has 85% accuracy.

In the table 5.3, we compare V-detector NSA results with other techniques; we can see that OCSVM and IF performs better than NSA for gradient-based attacks for low noise attacks NSA outperform both of them. Variable Autoencoder didn't perform well due to a low number of samples. SOGAL and MOGAL based techniques were failed to work with low models. NSA works faster than them based on the results from table **??**.

In an adaptive attack, such as BPDA [199], attackers bypass well known pre-process techniques by applying a differential approximation. This differential approximation process will not work as the detector's position and radius change after each successful non-adversarial input. So, the defensive filters can be changing continuously, and the attacker will not have a static

defense to bypass. However, query-based attacks need to be detected as those can bias the defensive filters.

In table 5.11, we compared our results with other well-known defense strategies. Our defense technique has exhibited similar performance as other techniques but our is more effective in detecting advanced low-noise attacks such as CW and Deepfool. Moreover, existing defense techniques have many limitations which is evident in our comparative results shown in table 4.12. As mentioned earlier, negative filtering strategy can work without any attack sample generation and remain robust against current state of the adaptive attack. [171] introduced a mechanism to leverage 'Generative Adversarial Networks (GAN)' capability to reduce adversarial perturbations' performance. The GAN effectiveness depends on the GAN training, which is computationally complex and needs proper understanding of the dataset. In contrast, our approach doesn't need a complex training method and computationally faster than GAN based defense technique.

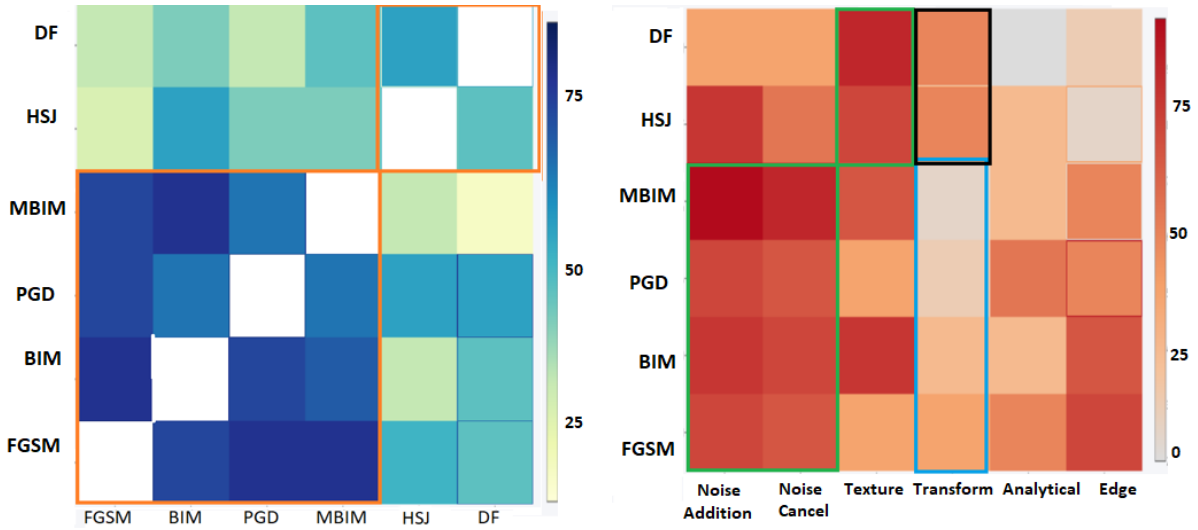
Chapter 5

Integrated Filtering- End-to-End

5.1 Research Findings

From our extensive literature review and empirical examination, we observed seven adversarial phenomena. These are:

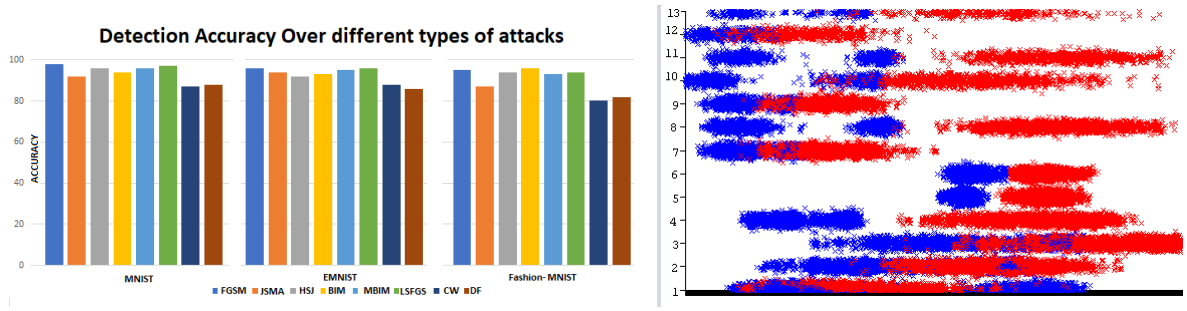
1. Advanced AAs are ineffective in the physical environment.
2. AA noises have a distinct nature that is related to the attack method.
3. Clean and adversarial inputs have identifiable noise difference.
4. Same filtering technique will work for all ML model for a specific dataset.
5. Different filters have different effectiveness to detect AA.
6. Attack methods from the same family possess weakness to the same detection filters.
7. If data-sets are similar, filter-based defenses are transferable.
8. Outlier detection methods can detect AAs as outliers.



(a) The best performing filter from each attack method was applied to the other attacks, and their accuracy was presented in the heatmap (MNIST Dataset).

(b) Different filters have different effectiveness to detect attack types. Here, each filter family average performance was presented against different attacks. In the green area, texture-based filters perform well in boundary-based attack and Noise filters works better for gradient-based attacks

Figure 5.1: Two different phenomenon PH5 and PH6 observed.



(a) Different attack type accuracy over all 3 different dataset against using the filters range generated from Fashion-MNIST. This proved filter based defense are transferable.

(b) All images SNR value plotted in the x-axis(0-1 normalized) w.r.t filter-id. In the y-axis, blue are clean inputs, and reds are adversarial images SNR values, we can observe that most of the adversarial inputs SNR are outside the SNR values of clean inputs. Some of the filters there is overlap, but there are distinct ranges present for adversarial and clean images.

Figure 5.2: Two different phenomenon 7 and 3 observed.

Advanced AAs are ineffective in the physical environment.

In 2017, [112] showed that in the digital version and printed version success of adversarial methods decline. They tried to justify their argument with FGSM, BIM, and other iterative methods.[130, 148] experimented with FGSM, BIM, and LBFGS methods and showed the destruction rate up of 100% based on distances invalidating these attacks.

AA noises have a distinct nature that is related to the attack method.

Different attack methods try to attack an ML model differently; for example, in the JSMA attack method, the most significant part of the input noise is added; in the FGSM attack, a gradient loss noises are added. In score based attack, the most significant pixel value searched and changed. In the work of [77], these behaviors have been briefly discussed.

Clean and adversarial inputs have identifiable noise difference.

Researchers ([158][6][77]) demonstrate adversarial and clean images have a comparable differences in their noise value which are identifiable for attacks such as FGSM, BIM etc. In figure 4.7, we illustrated the noise differences between clean and adversarial images SNR metrics. In figure 4.7 and 4.7, it was also illustrated that normal filtering technique highlighted the noise part after pixel difference method[77], and these noises could be detected using other metrics such as the histogram average and the local binary pattern average. In the figure 5.2 we illustrated the adversarial example metrics value normalized in 0-1 in x-axis and presented for the different 15 filters from table 4.2. We can see that clean and adversarial images overlap, but in cumulatively, they are distinguishable.

Same filtering technique will work for all ML model for a specific dataset

From PH2 and PH3, we can see that filters can detect AAs in data preprocess stage[158] [6][77]. That means this technique will work for the black-box model, which means defense is not required to access or modify the ML. So if the ML changes, for example, from Resnet to Vgg or

SVM to a Random Forest, the defense technique needs no changes. It will be completely independent of the ML changes.

Different filters have different effectiveness to detect AAs

We have experimented with different filters, as presented in table 4.2. From this result, using filter classification, we generated the heat-map in figure 5.1. Here, we can see that noise addition and canceling filters are working better in the gradient-based attack, and texture-based filters are working better for boundary-based attack types. For example, FGSM and BIM are both gradient-based attacks, and we find out blur works against both of these attacks. This result is expected as the second phenomenon established that AA noises have a distinct nature related to the attack method. This phenomenon proves that picking one filter from each filter family will have more effectiveness than selecting all the filters from the same filter family class.

Attack methods from the same family possess weakness to the same detection filters

We plotted observed average accuracy for each filter family on each attack method as illustrated in the heatmap figure 5.1 b. It is visible that if filter works for one attack type, it will also work for other attack types. The best performing filter for FGSM performs well in other attack-type of the same family. That assures us that we do not need to evaluate all the attacks, but at least one from each attack family is sufficient to assess defenses' efficiency.

If data-sets are similar, filter-based defenses are transferable.

In figure 5.2, it is illustrated that the same filters, whose ranges are calculated using Fashion-MNIST, is working with satisfactory results. MNIST, EMNIST, and Fashion-MNIST are the same type of dataset.This phenomenon assures us that the AEF filter does not need the ranges from the full dataset. A subsample min-max range will provide an efficient range to detect the AA.

Abbr	Algorithm	Accuracy
OCSVM [39]	One-Class SVM	99
LMDD[12]	Deviation-based	98
LOF[22]	Local Outlier Factor	98
COF[191]	Connectivity-Based	91
CBLOF[83]	Clustering-Based	92
HBOS[62]	Histogram-based	91
kNN[161]	k Nearest Neighbors	91
ABOD [108]	Angle-Based	62
COPOD[121]	Copula-Based	75
SOS[90]	Stochastic Selection	66
IF[195]	Isolation Forest	99
FB [114]	Feature Bagging	99
XGBOD [228]	Extreme Boosting Based	26
AutoEncoder[2]	Fully connected AutoEncoder	43
VAE[105]	Variational AutoEncoder	41
SO_GAAL [127]	Single-Objective GAN	40
MO_GAAL[127]	Multiple-Objective GAN	35
Vdetector[234]	Variable Size NSA	99
RNSA[50]	Random real value NSA	75

Table 5.1: List of outlier detection algorithm and their accuracy to detect adversarial (FGSM) input of class label 'O'.

Outlier detection methods can detect AAs as outliers

The work of ruff et al.[167] shows that outlier detection methods can classify class label from outlier samples. In multi-class classification, each class separately generate their own latent space and outlier detected there as negative class and inliers are detected as positive class and able to achieve 95%+ accuracy for MNIST class classification. Similar approach we experimented with adversarial samples for single class classification. We took class label '0' as positive class or inlier, all other 9 classes and adversarial samples for class 0 are considered negative class or outlier. We trained with 1000 positive class. We used [229] developed outlier library in our experimentation. We tested with 500 positive data, and 500 adversarial sample (FGSM samples generated using [63] and [149]) of class label 0. The accuracy was presented in the table 5.1. We can see that one class support vector machine and V-detector based negative selection algorithm does better than other.

Static defenses can by-passed by adaptive attacks

Carlini el al[32] exhibited an adaptive attack where the attacker can bypass the known defenses. So, if the defense is not changes or remain static it will be vulnerable to adaptive attacks. Also, more recent works showed that dynamic defense mechanism which claims effectiveness against adaptive attacks fails against gradient based adaptive attack[198].

5.2 Overall Architecture

To build a robust ML/AI-based system against malicious adversaries, we designed a dual-filtering (i.e., commutative filtering) scheme, which employs two filtering mechanisms: one at the input stage (before samples are fed to the core learning model) and other at the output of ML (before the decision module). These two filters can function independently as well as dependently (i.e., in a commutative fashion). Specifically, the input filter's main aim is to filter misleading and out of distribution inputs (e.g., image of animal but not human face in a face recognition system). The output filter's goal is handling larger variations and restricting misclassification rates in order to improve overall accuracy of the system. The proposed dual-filtering strategy can be used both in training and testing phases of ML. For instance, the independent input filter may be used to detect and deter the poising attacks in a supervised ML. Likewise, dual-commutative filters may help addressing adversaries both in supervised and unsupervised ML. A machine learning framework usually consists of four main modules: feature extraction, feature selection (optional), classification/clustering, and decision. As depicted in Figure 5.3, the input filters are placed after pre-processing of data stream/feature selection to feed to core learning model and the output filters are placed after classification/clustering/raw decision module, respectively.

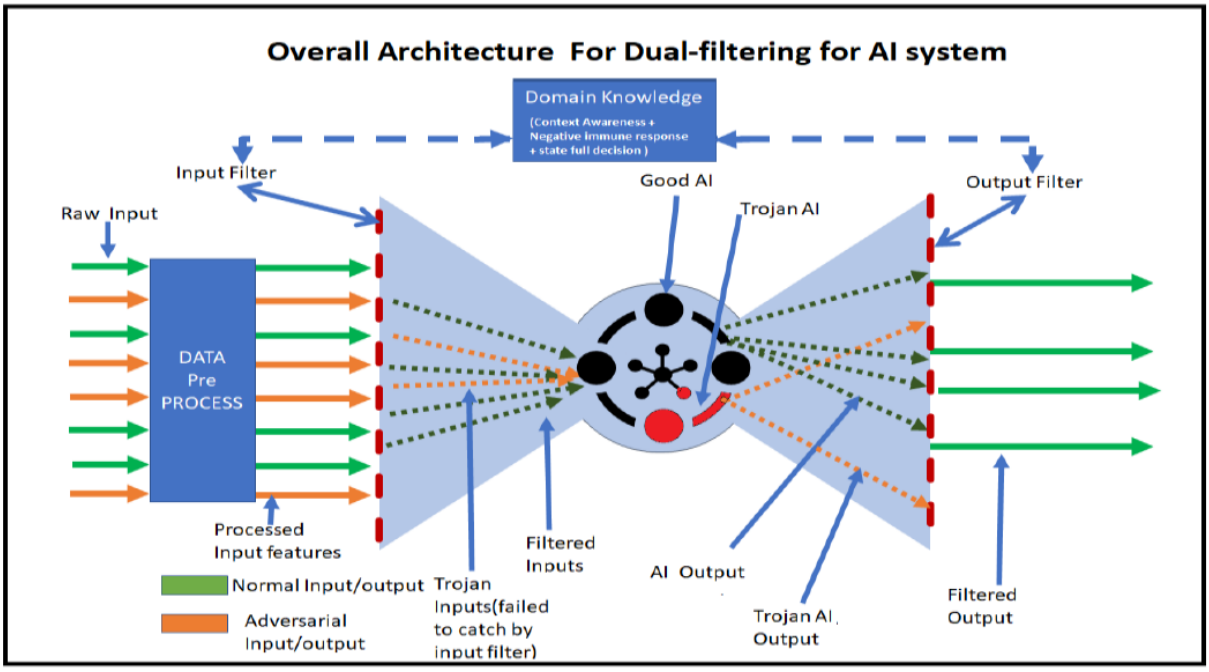


Figure 5.3: Schematic of the proposed Dual-Filtering (DF) framework.

As can been seen in Figure 5.3, the raw input sample is first pre-processed and then fed to the input filter to determine if the received feature/sample is benign or attack and reject accordingly. The outcome (i.e., raw decision) by ML system is given to the output filter for further scrutiny. The output filter uses context-information and/or communicates with the input filter to make the correct final decision. An ensemble of different noise removal or detection filters was applied to detect AAs in a recent work [47]. Other techniques focused mostly on adding extra layer on a ML module by adversarial sample training or modification of deep learning models. These defense methods have some constraints, and exposed ML models to new vulnerabilities [76].

In 2019, some works reported launching adaptive attacks where they could bypass known defenses [32]. To alleviate the situation, we consider a non-deterministic (white-box) approach where the attackers cannot perceive our defenses to launch adaptive attacks. Accordingly, we investigated an active learning[175] based dual-validation scheme which work as an extra security (filtering) layer and improve the learning model's trustworthiness.

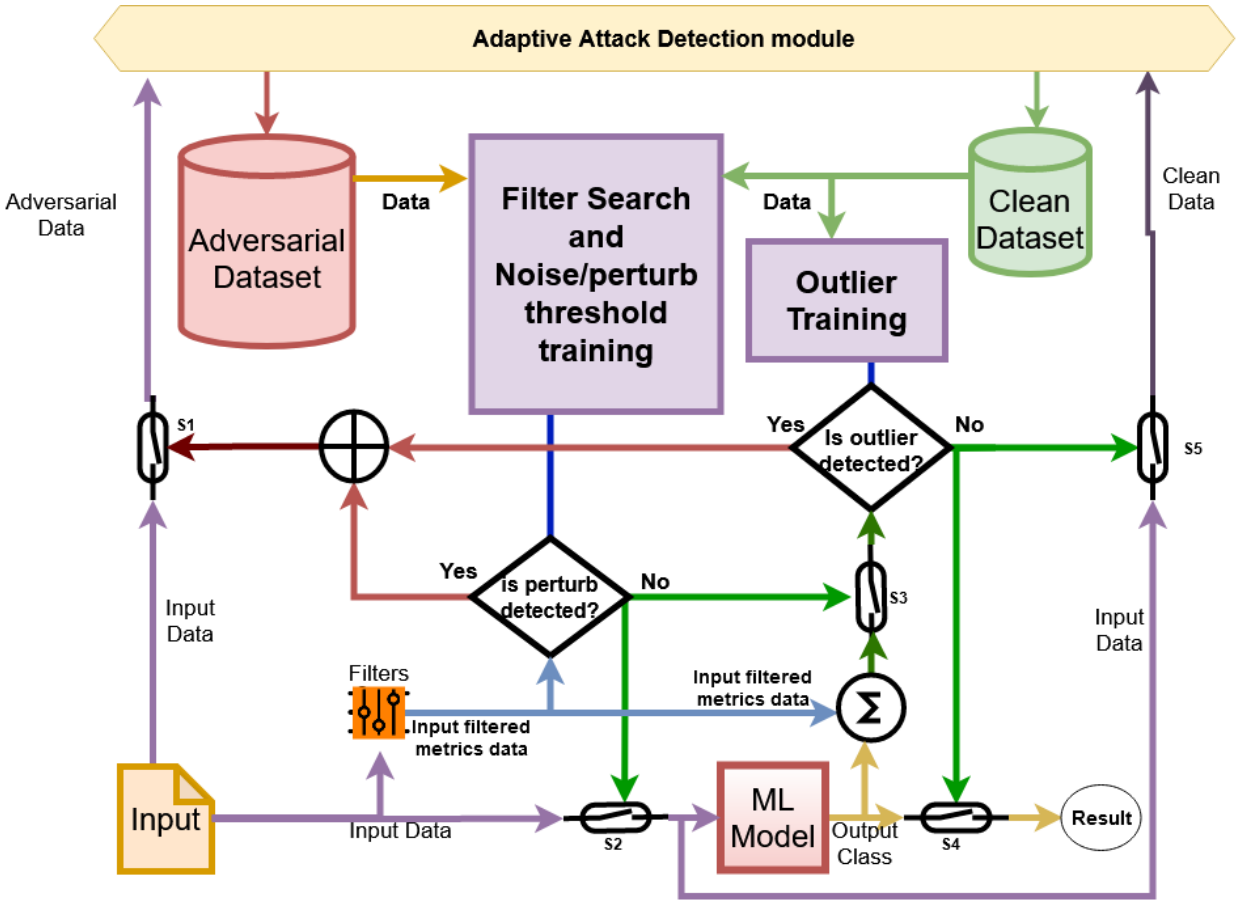


Figure 5.4: Illustration of basic flow concept for proposed Dual Inspection framework. If the input is not adversarial, the original input (not the processed) be sent to the learning model/ML and after ML produce class label, that labels latent space will be used in outlier method. The outlier decision boundary and the threshold of noise will change as the dataset of adversarial and clean data set are updated by each input.

In the figure 5.4, we illustrated the basic concept of our proposed solution. As we know that it is possible to detect adversarial input noise using different filters, we will apply filters to detect noise. We need to know which filter we need and the difference between the clean and adversarial noise threshold. That's why we first use the information from the ML model to determine the input is an outlier for the class label the ML model is classified or not. If it is an outlier, we will send it to the adversarial dataset. If not, we will send it to the clean dataset and update the outlier methods decision boundary and determine the required filters and the noise thresholds. Before update/retrain the output and input learning model, we will inspect the data for adaptive attack patterns in the adaptive attack detection module. The figure 5.5 demonstrated our

proposed dual inspection strategy. It is shown that the inspection before and after ML are independent and can be deployed as a plugin. As in active learning, when the clean dataset has some data, it will train the outlier detection techniques, and the 'inspection after ML' module will start to work. After the outlier finds some adversarial examples, the adversarial dataset receives some data. When the adversarial dataset has sufficient data, our multi-objective Genetic algorithm started the genetic search for filter sequences that are effective against the adversarial noises and the differentiating noise thresholds for these sequences. As time progresses, MOGA will detect more adversarial samples, and the knowledge of the outlier detection technique will transfer to noise detection techniques. This way ML model has to process fewer adversarial examples. We will select different filter sequences for each input and different outlier detection methods for each input to make the defense dynamic. After each (or a specific amount of input), outlier methods will retrain, and it will update the outlier detection decision boundary. Similarly, MOGA will update the filters library subsequently. This way, both outlier and filter-based defense technique will keep themselves updated as time progress. As this method can be vulnerable by adaptive attack, we will store the data and inspect for adaptive attack pattern before update the filters and outlier detection methods.

The basic workflows from the figure 5.4:

1. Input will be sent for filters to extract different metrics (SNR, Histogram etc). There will be a dynamic selection of the filter set from the filter library.

2. Extracted filter metrics value will check for perturb, if it is above certain threshold switch S1 will open or other wise switch s2 and s3 will open.

3. S1 open:

    - input will be sent to adversarial dataset and the process will terminate.
    - Adversarial dataset will retrain the filter sequence search for noise detection and change the threshold value.

4. S3 and S2 open:

    - If S3 open, extracted filter metrics value will be sent to outlier detection system.
    - If S2 open, input data will be sent to ML model and Switch S5.

5. ML model will deliver the output class to S4 and outlier detection system.

6. Outlier detection system will randomly pick one outlier detection method. If it detected as outlier witch s1 will open, otherwise S4 and S5 will open.

7. S1 open:

    - input will be sent to adversarial dataset and the process will terminate.
    - Adversarial dataset will retrain the filter sequence search for noise detection and change the threshold value.

8. S4 and S5 open:

    - S4 will provide the final output class and S5 will send the input to clean dataset which will trigger the retrain of outlier methods and change the outlier decision boundary.

5.3 Workflow

5.3.1 Multi-objective Genetic Search for filters

We need multiple filter sequences because we cannot use the same sequence of filter for every input. A sequence could be any length. Search for optimal set of sequences require significant computational time, if we do exhaustive search, considering multiple objectives. That's why we will employ a multi-objective GA to search for the optimal set of sequences as pareto-front solutions. For search filters, we need to consider different factors besides their accuracy. Based on our objective, our filters need to be fast, that's why order of filters are important because different

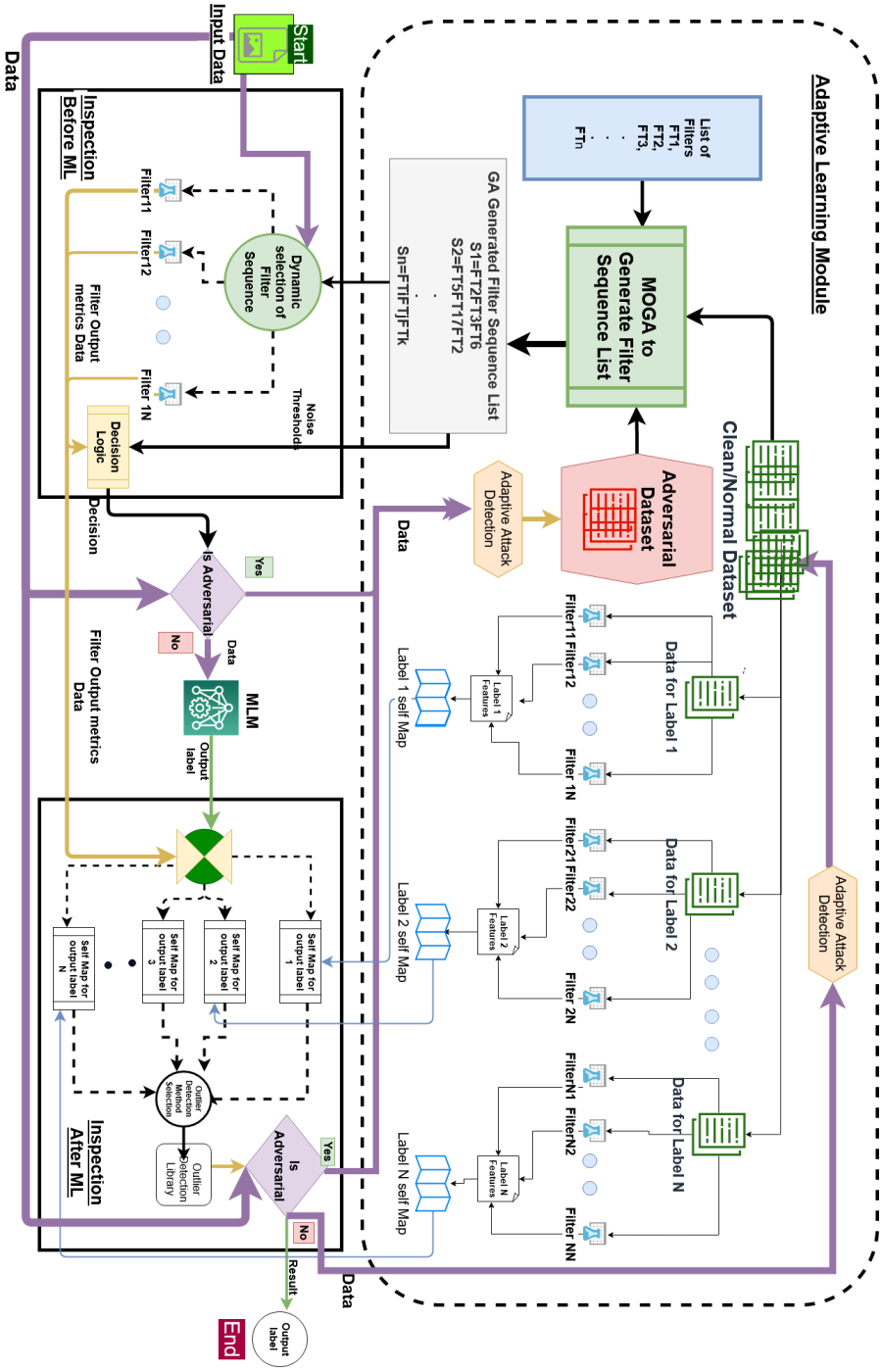


Figure 5.5: Illustration of proposed Dual Inspection framework. If the input is not adversarial, the original input (not the processed) be sent to the learning model/ML and after ML produce class label, that labels latent space will be used in outlier method. Selection of outlier and filer sequence will be dynamic.

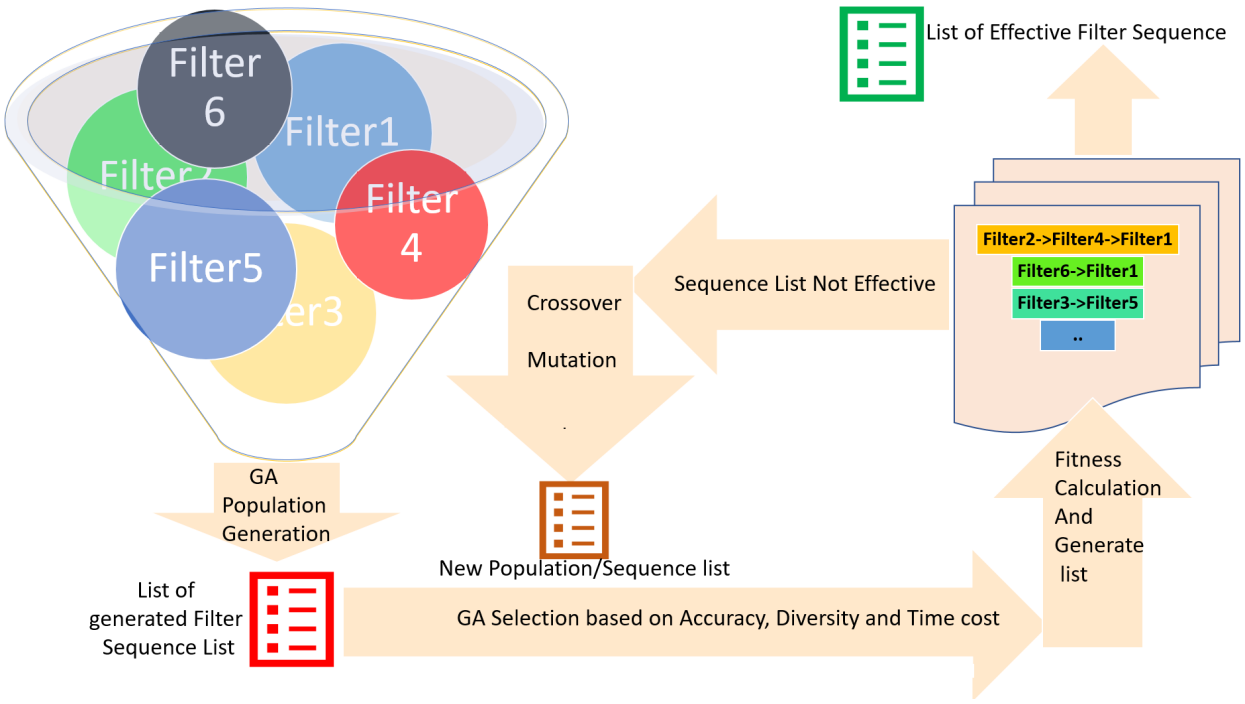


Figure 5.6: GA to search appropriate filters.

order of filters require different amounts of processing times. It is preferable to have our solutions time efficient. If we have N number of filters, then total possible number of sequences will be our search space. If we do not consider time efficiency, then we do not need to order in a combination of sequence (For different order a sequence accuracy remain static but time efficiency change). We can optimize our search space by limiting the minimum sequence length and maximum sequence length. So, our search space equation will be :

$$Optimized_{Search_{space}} = \sum_{k=min}^{N-max} \frac{N!}{N! - k!} \tag{5.1}$$

Here, min and max are the minimum and maximum length of a filter sequence. Suppose, we have 17 filters and minimum length were 6, then our optimized experimental search space has consists of 9.66^{14} search item. It justifies the necessity of using heuristics search method like GA. In summary, we need a time-efficient, to produce a reliable performance and a unique set of sequence. Our designed multi-objective GA can achieve all these criteria. The purpose of using a genetic search is to find a diverse sequence of filters detecting AAs with maximum accuracy

while each filter is having distinctive characteristics and capabilities when deployed such a sequence adaptively (interchangeably) in a ML that will be unpredictable to attackers compared to a static ensemble of well-known filters. So, the GA will find not only the best filter ensemble, but also a set of diverse filter sequences in multi-objective Pareto-front.

Perturb Range/threshold Determination of filters

First, we need to process the clean dataset and run all the filters and calculate their metrics value and gather their Mean,Std Dev, Max. Using algorithm for each filter. Now, if Mean is $\bar{\mu}$, Standard deviation is σ, Our lower range (L_r) and upper range(U_r) calculation denoted by equation 5.2 and 5.3

$$L_r = Min - \frac{\sigma}{\bar{\mu}} \tag{5.2}$$

$$U_r = Max + \frac{\sigma}{\bar{\mu}} \tag{5.3}$$

Using equation 5.2,5.3 for 17 filters, we generated list of upper and lowerange

Encoding

First, we encoded all filters according to table 4.2. Here, 17 algorithms were assigned sequence number $FT1, FT2...FT17$. These filters are our genes. We will create our individuals/chromosome using these genes. We generated the population by random sequence generation using the genes. So, each sequence is consists of different length of filters. We remove multiple occurrences of filters in a single sequence. As example a random sequence $FT2FT5FT11FT12$ means **Blur -Census - Morph - Canny**. That way, we generated multiple lengths of sequences as our initial population.

Fitness function

We have three objectives. they are Accuracy (α), time to detection(β) and Insider diverseness(γ) of filters in sequence. Accuracy is the success rate of detection by the filters. The filter sequence

takes adversarial and clean samples from the dataset. Based on the filter sequence's metrics value range, we check how many adversarial samples we can detect and how many we falsely detected. We used F1 score as Accuracy value.

$$\alpha(S) = F1 \tag{5.4}$$

For time, if each filter take t_i time, then total time to detection δ_t can be calculated by

$$\beta(S) = \sum_{i=0}^{n} t \tag{5.5}$$

For insider diverseness, for each filer $f_i \in F_i$, here F is the filter family and f is the filter, S is the sequence.

$$\gamma(S) = \frac{\sum f \in F | f \cap S| > 0}{\sum F} \tag{5.6}$$

We normalized all three objective data using equation 5.7

$$X_{sc} = \frac{X - X_{min}}{X_{max} - X_{min}}. \tag{5.7}$$

We inversed the Time data, so, we have to maximize all of the objectives. Our fitness function denoted by

$$\max(f(S)) = ((\alpha_n(S)), (\beta_n(S)), (\delta_n(S))) \tag{5.8}$$

where, α_n is normalized accuracy, $\beta_n$is normalized inverse time, δ_n is diverseness factors.

We have penalty function to prioritize simpler solution and weight values to speed up the GA process. We observed that, in the beginning α is low and after a certain iteration γ gets lower. We use W_0 for α and W_1 for γ as weigh value.

For penalty functions we need below parameters

Length of Best fitted individual in previous iteration $|\max_{f(S_i)} \in \forall(S)|$

Size of current Sequence = $|S|$

Total number of filters = $\sum_{i=0}^{n} |f| \in \forall F$

Equation for penalty function value can be denoted by

$$\textit{pf(S)} = \frac{\textit{f(S)}}{100} \times \frac{|S| - |\max_{f(S_i)} \in \forall(S)|}{\sum_{i=0}^{n} |f| \in \forall(F)} \tag{5.9}$$

So, from equation 5.8, fitness for S is

$$\textit{f}(S) = \sqrt{(\alpha_n(S)^2 \times W_0 + \beta_n(S)^2 + \delta_n(S))^2 \times W_1} - \textit{pf(S)} \tag{5.10}$$

Crossover, mutation and selection

We used an elitist strategy with rank Selection [46] and kept the best performing filters for the next generation in steady-state genetic search. We used PMX crossover as the order of the filter sequence are important optimization criteria in a sequence[200]. In the figure 5.6, We illustrated the genetic search of near-optimal filter sequences where search terminated after specific iterations or if the fitness values do not improve for a long period i.e. threshold number of iterations.

5.3.2 One class classifications Outlier method

There will be different latent spaces for each class to detect that class's outlier. In the figure 5.7, we can see that MNIST digits have their clusters for each class label, and these are well separable. In the figure 5.8, we can see filter-based metrics can very easily differentiate between adversarial and clean sample. We suggest using an ensemble of different outlier detection methods—for example, a combination of One-class SVM, Isolation forest, and Negative selection algorithm. Our experimental results shows that negative selection algorithm is random nonlinear learning system, which is applicable for adversarial detection and it randomness made is easy to make the system adaptive by regular updating the learning model.

In the table 5.2, we experimented the attack sample accuracy rate with v-detector generated using different number of clean samples. It is observed that after each iteration

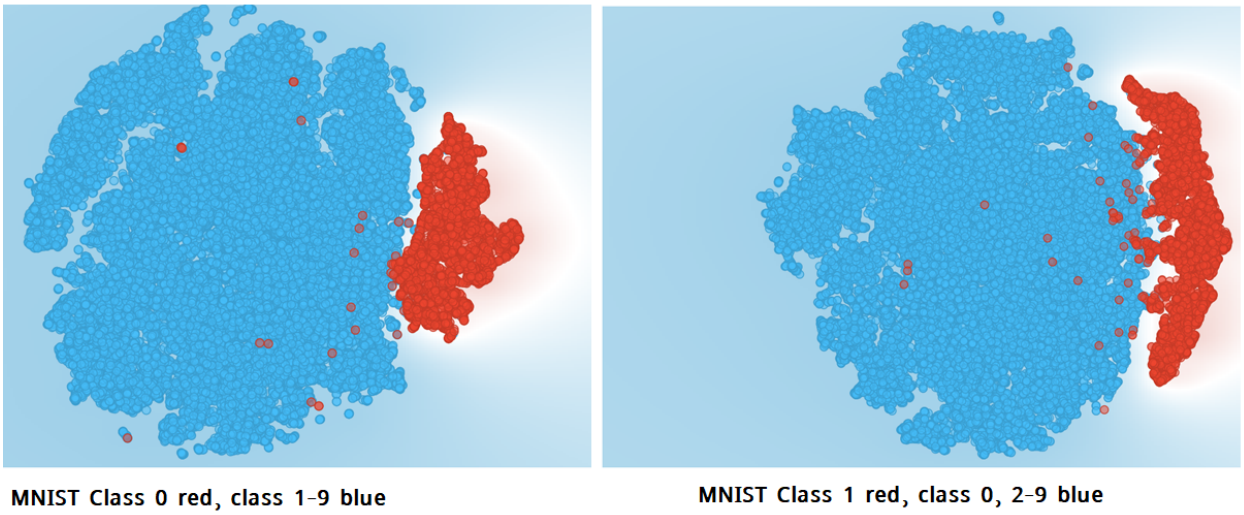


Figure 5.7: PCA based clustering for class label 0, and 1 from MNIST dataset using each class own latent space.

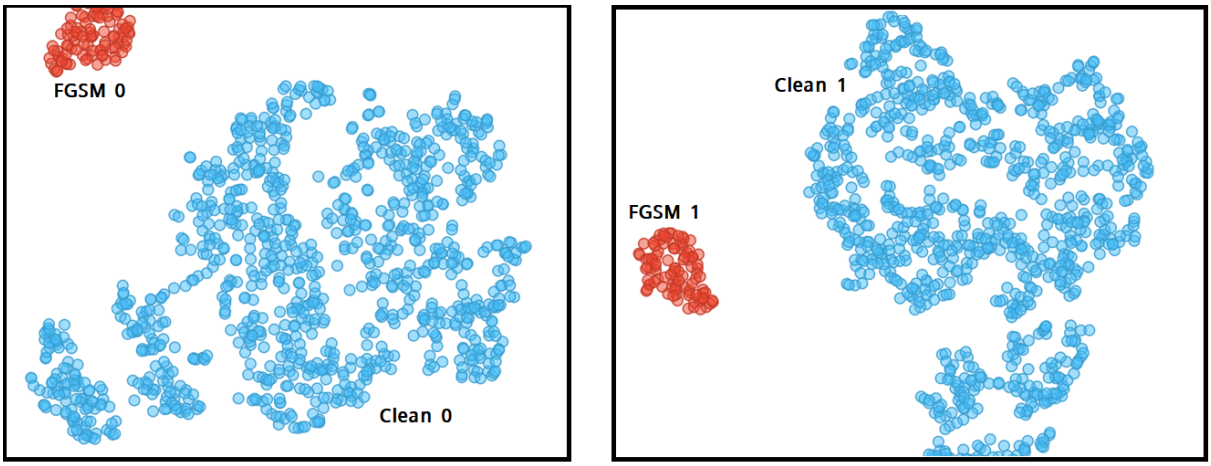


Figure 5.8: FGSM based adversarial input differs from their target class using filtered metrics presented using PCA for dimensional reduction.

Attack type	Step1	Step2	Step3	Step4
FGSM	0.86	0.92	0.902	0.93
BIM	0.89.0	90.0	90.0	0.90
PGD	0.92	0.94	0.95	0.95

Table 5.2: Detection of adversarial inputs which classified as MNIST class label '0' and with clean input of class label '0' , after sample size increased 100 in each step.

Attack type	NSA	OCSVM	IF	VAE	SOGAL	MOGAL
FGSM	0.93	0.99	0.93	0.65	0.5	0.5
BIM	0.90	0.98	0.91	0.66	0.5	0.5
PGD	0.95	0.99	0.92	0.50	0.5	0.5
MBIM	0.91	0.98	0.94	0.46	0.5	0.5
HSJ	0.88	0.55	0.41	0.65		
JSMA	0.9	0.56	0.8	0.83		
CW	0.96	0.42	0.66	0.52		
DF	0.91	0.45	0.76	0.55		

Table 5.3: detection of adversarial inputs which classified as MNIST class label '0' and with clean input of class label '0'.

v-detector performance slightly increased. In the table 5.3, we compare v-detector NSA results with other techniques; we can see that OCSVM and IF performs better than NSA for gradient-based attacks for low noise attacks NSA outperforms both of them. Variable Autoencoder did not perform well due to a low number of samples. SOGAL and MOGAL based techniques were also failed to work with low models.

5.3.3 Adaptiveness and dynamic selection

We randomly choose different filter sequence and other outlier methods to keep the system dynamic for each input. After each input, the outlier detection modules are updated by changing their decision boundaries, making the detection filters adaptive. However, the filter sequence can change, and the noise threshold value gets updated after each MOGA run. This makes common adaptive attacks ineffective as each input continuously updates the defense strategy. An adaptive attacker will first send random clean input. And started to add some noise in these inputs and send repeatedly until the classification result changes. That way adaptive attackers will know the decision boundary of the learning model. Then adaptive attacker will start creating input that is close to the decision boundary in the representation space.

In our method, attackers have to bypass our dynamic and changing adversarial detection method, which decision boundary is not affected by the actual learning model. If our filter set or outlier detector method was static this method would work, but dynamic selection of filterset and

outlier detection method will make it hard to formulate the adaptive attack. Additionally, after a certain set of inputs, we will regenerate negative detector sets by considering these new inputs as self data. So, entire outliers decision parameter would change and the adaptive attacker will not able to establish a fixed decision boundary line for adversarial and nonadversarial input data as adaptive attacker is looking for class classification boundary not adversarial and non-adversarial decision boundary. This update method can also be vulnerable to adaptive attacks which aim to bias the method accuracy, we added a adaptive attack detection module before update/retrain our adversarial detection techniques.

In the adaptive attack detection module, we will analyze distributions of last certain number of inputs are align with total distributions of inputs. We will use the Kolmogorov-Smirnov Goodness of Fit Test (K-S test) compares data with a known distribution and lets us know if they have the same distribution. This test is nonparametric as it doesn't assume any particular underlying distribution[16]. The Kolmogorov-Smirnov test determines a null hypothesis, H_0, that the two samples originate from the same distribution. Then we explore for evidence that this hypothesis should be rejected and formulate this as probability ρ. If the prospect of the samples being from different distributions tops a confidence level we reject the original hypothesis and accept hypothesis H_1, which stated that the two samples are from different distributions. Based on the KS distribution table, if $\rho < \frac{1.22}{\sqrt{n}}$ (where n= number of stored input) than the stored input has inputs from adaptive attack. We disregarded those samples as these may create data bias in our defense learning system.

In summary, our adaptive defense mechanism consists of the following properties

- Dynamic selection of filter set sequence which will make it harder to formulate adaptive attack based on known filter knowledge.
- Dynamic selection of outlier detection method, it will make the adaptive attack to consider all outlier detection method when developing attack input that will make generating input computationally expensive.

- Defense is always learning which will continue changing the filter sequences and decision boundary of outlier detection models. It will make an adaptive attack difficult to search decision boundary.

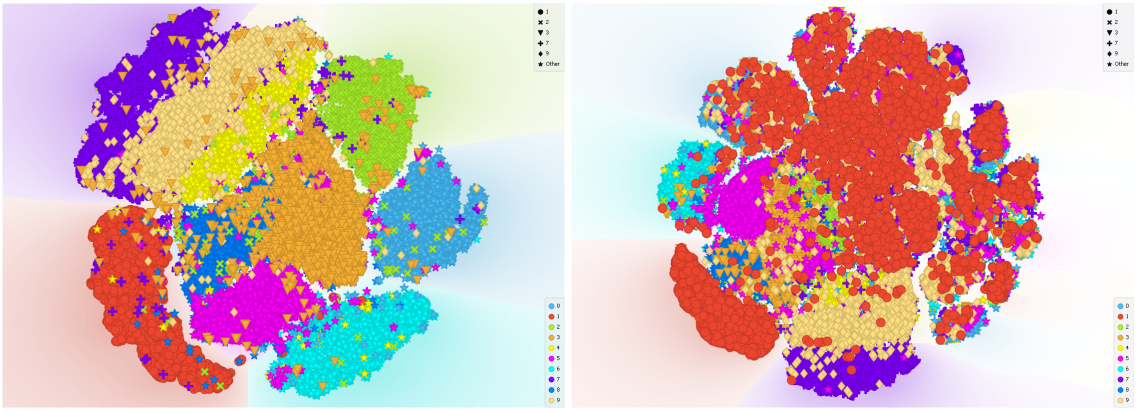
- To protect against continuous query-based attacks, we will monitor and analyze input trends using the K-S test. The number of inputs considered for the K-S test will be dynamic. Formulate a query-based attack on the defense system will be hard due to the randomness of the K-S test sample number. Our input trend detection system can effectively monitor adaptive attacks and able to take countermeasure.

Our defense properties will make the state of the art adaptive attack ineffective and it will make computationally harder to formulate new adaptive attacks. Our proposed approach has a cold start problem as in the beginning we have empty adversarial and clean dataset. That's why outlier detection or filter based inspection does not start working until significant samples in the clean and adversarial dataset. Also, we assume that the first set of samples are clean, otherwise outlier detection gets train with noisy data. For the MNIST dataset, we observed 250 random samples required for each class to detect outliers using OCSVM and Isolation forest.

5.4 Experiments

5.4.1 Dataset Generation

We did a comprehensive experiment with MNIST and CFIAR-10. We did extensive testing with the MNIST dataset for all the 10 classes and with the full dataset. We did CFIAR testing with two classes. After that, we evaluated our method using EMNIST, Fashion-MNIST, and IMAGENET data-set which re-validated our methodology. We generated FGSM, JSMA, and CW samples to test the results. We generated 100,000 FGSM samples using LENET-5. LeNet-5 LeNet-5 CNN architecture is made up of 7 layers. The layer composition consists of 3 convolutional layers, 2 subsampling layers and 2 fully connected layers. For JSMA we generated 100,000 JSMA



((a)) All clean data projected using t-sne visualization with class labels.

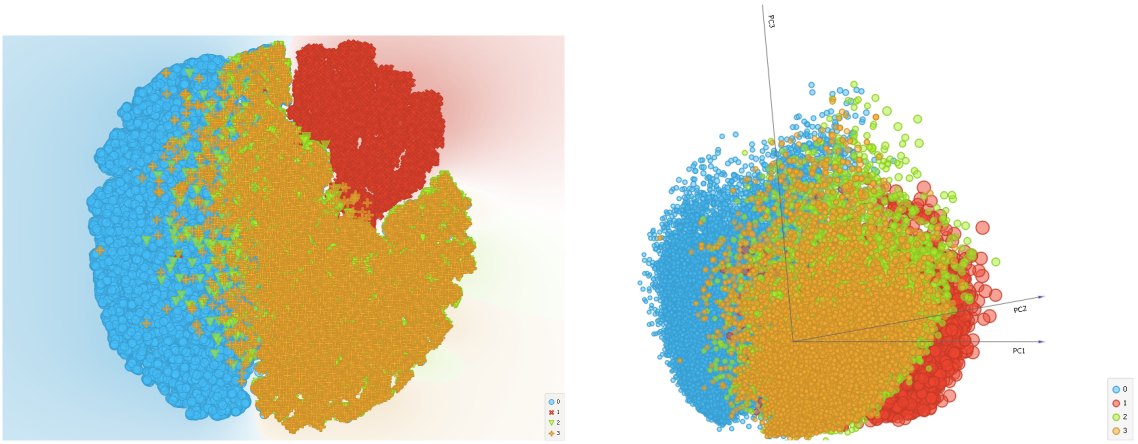
((b)) All clean and adversarial data projected t-sne visualization with class labels.

Figure 5.9: Experimental data representation space for each class of MNIST digits.

samples using a CNN. CNN architecture is made up of 5 layers. The layer composition consists of 3 convolutional layers, 1 flatten and 1 dense layers. All of the activation functions are using RELU. and last we generated 100,000 CW samples using VGG-16 neural net.

To establish the ground truth for our research we used 30000 clean image samples, 10000 FGSM, 10000 JSMA and 10000 CW attack samples on MNIST dataset. For filtering operation we picked 14 filters using python opencv library. They are medianblur, GaussianBlur, AverageBlur, Bilateral blur, AdditivePoissonNoise, AdditiveGaussianNoise ,Erode TopHat ,Blackhat ,Morphology gradient ,Opening ,Closing, Dialte filter. We apply the filer in the image and than extracted difference between original image and the filtered image. After that we measure the average and standard deviation of white color histogram for the extracted image and horizontal and vertical signal to noise ration for the extracted image.

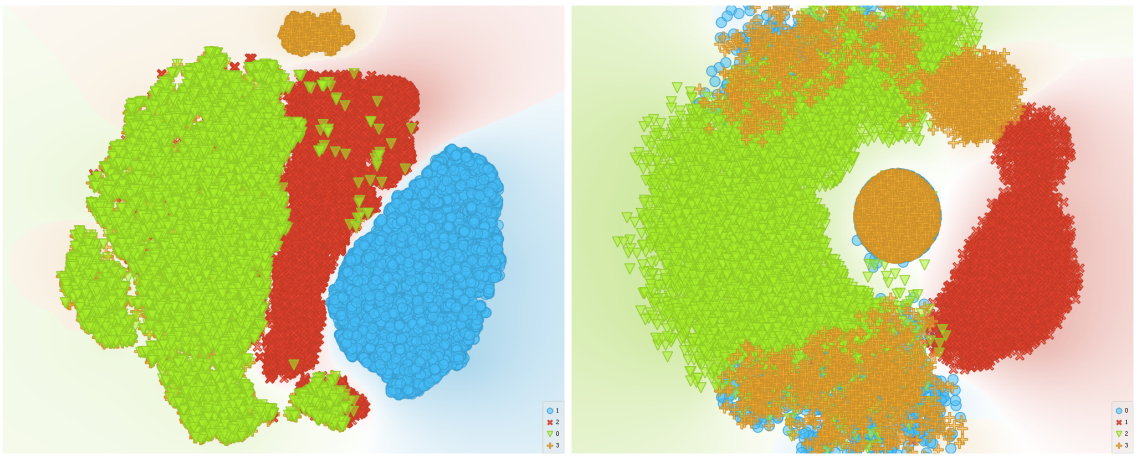
In the figure 5.9 'b' we visualized how adversarial (FGSM+JSMA+CW) inputs of one class label overlap with other class label compared with 'a' where only clean inputs where presented. This shows that adversarial samples are hard to distinguish between class labels. In the figure 5.10, we represented all inputs with their adversarial attack type along side the clean input. Here blues are the clean one. We can see here the FGSM which is visualized with red are not overlapping with clean one or other attack type much. But JSMA and CW are highly overlapping



((a)) All clean and adversarial data projected using PCA reduction and t-sne conversion.

((b)) All clean and adversarial data projected using 3 PCA components.

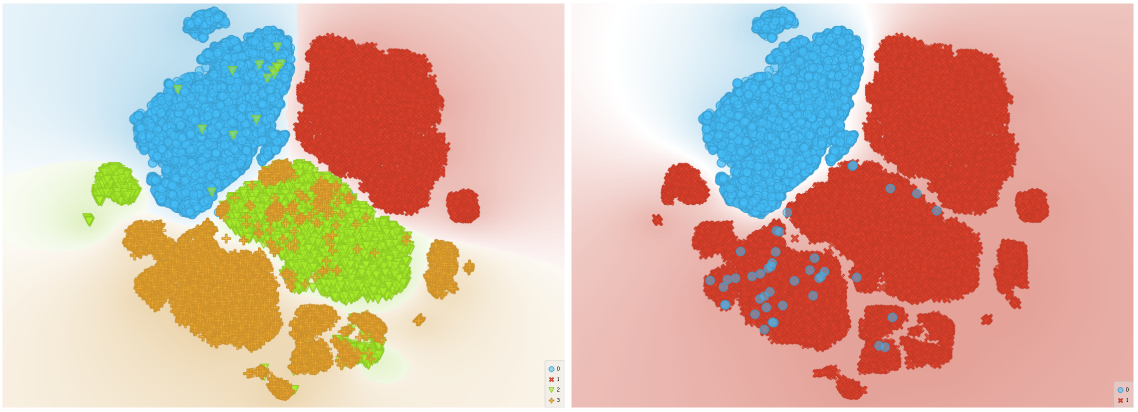
Figure 5.10: Experimental data representation space(Here clean is green, red is FGSM, blue is JSMA and yellow is CW).



((a)) All clean and adversarial data projected after applied 14 filters SNR values.

((b)) All clean and adversarial data projected after applied 14 filters average histogram values.

Figure 5.11: Experimental data representation space after filter applied with one metrics(Here clean is green, red is FGSM, blue is JSMA and yellow is CW).



((a)) All clean and adversarial data projected after applied 14 filters SNR and Histogram average valuesapplied 14 filters SNR and Histogram average values (Here clean is blue, red is FGSM, green is JSMA and yellow is CW).

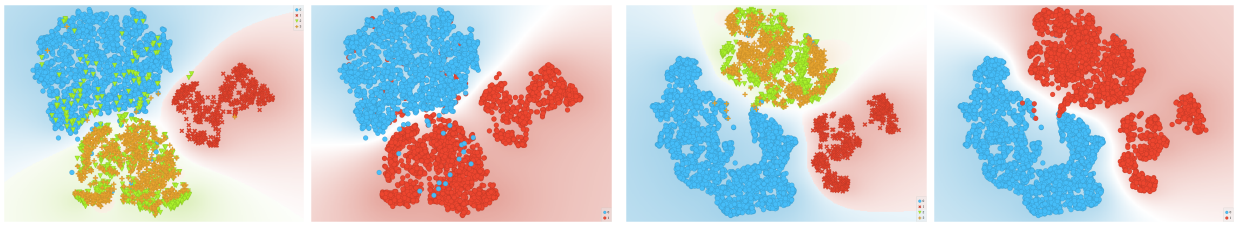
((b)) All clean and adversarial data projected after applied 14 filters SNR and Histogram average values (here blue is clean and red is fgsm,jsma and cw.

Figure 5.12: Experimental data representation space after filter applied with two metrics.

with each one and also partially with clean one. CW inputs are more overlapping with clean samples.

In the figure 5.11, we represented the adversarial and clean samples after applying 14 filters. in the figure 'a' of 5.11, we represented the SNR values of the images and it showed FGSM (blue) samples are very easily separable but JSMA (red) and CW(yellow) are hard to separate using SNR values only. However, JSMA are more separable but CW and clean samples are completely overlapping. in the figure 'b' of 5.11, we illustrated using histogram value and it made CW more separable than the clean ones. In the figure 5.12, we applied both SNR and histogram metrics together, and it visible that adversarial and clean samples are now more easily separable. In the figure 'b' of 5.12 blues are the clean samples and re are the adversarial samples. We can see some clean samples are overlapping with adversarial samples but other way is rare. In the figure 'a' we presented the adversarial attack type two and we see some CW samples are also overlapping with clean samples but it is negligible.

In the figure 5.13, we presented each class label adversarial and clean data both without adversarial classification and with adversarial classifications. This visual presentation shows that, when we each class as inlier and all other as outlier, than adversarial samples were more easily

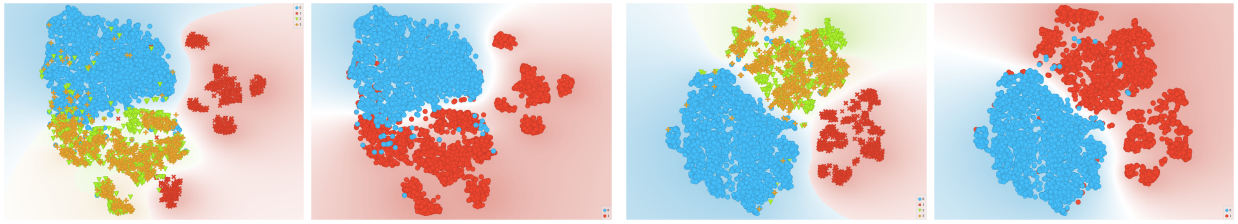


((a)) MNIST '0' with FGSM, JSMA and CW sep-arately

((b)) MNIST '0' with FGSM, JSMA and CW to-gether

((c)) MNIST '1' with FGSM, JSMA and CW sep-arately

((d)) MNIST '1' with FGSM, JSMA and CW to-gether

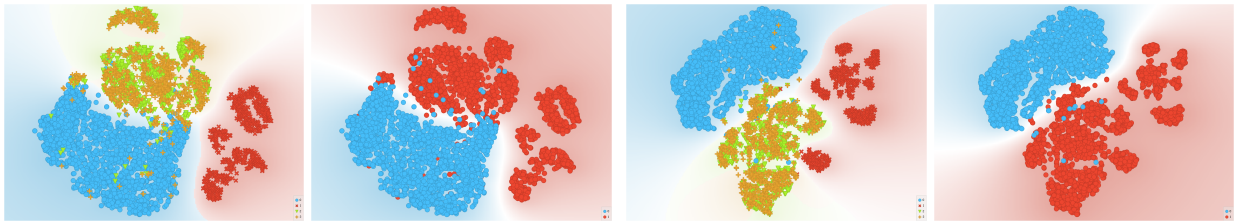


((e)) MNIST '2' with FGSM, JSMA and CW sep-arately

((f)) MNIST '2' with FGSM, JSMA and CW to-gether

((g)) MNIST '3' with FGSM, JSMA and CW sep-arately

((h)) MNIST '3' with FGSM, JSMA and CW to-gether

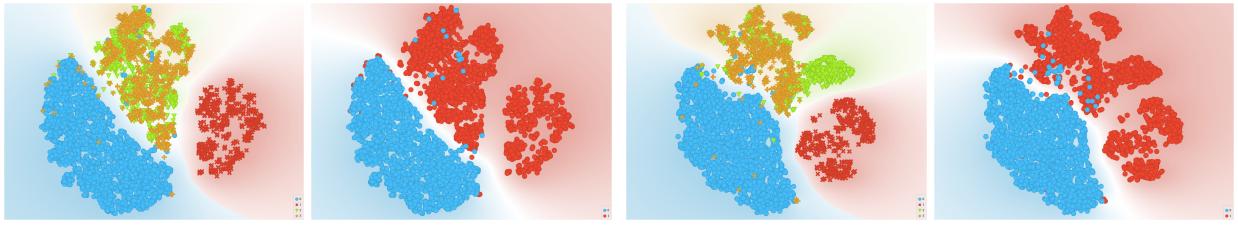


((i)) MNIST '4' with FGSM, JSMA and CW sep-arately

((j)) MNIST '4' with FGSM, JSMA and CW to-gether

((k)) MNIST '5' with FGSM, JSMA and CW sep-arately

((l)) MNIST '5' with FGSM, JSMA and CW to-gether

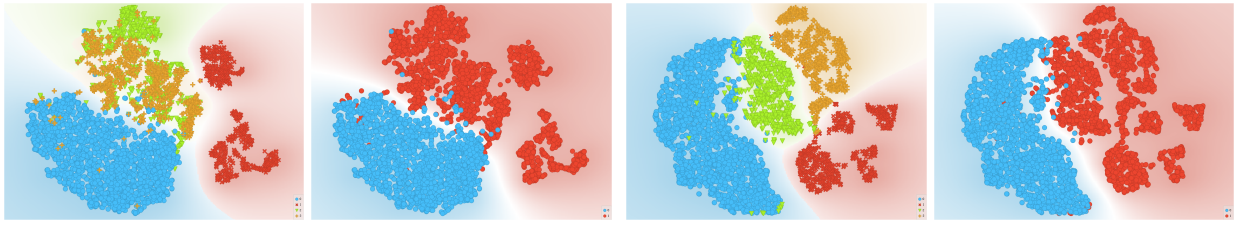


((m)) MNIST '6' with FGSM, JSMA and CW sep-arately

((n)) MNIST '6' with FGSM, JSMA and CW to-gether

((o)) MNIST '7' with FGSM, JSMA and CW sep-arately

((p)) MNIST '7' with FGSM, JSMA and CW to-gether



((q)) MNIST '8' with FGSM, JSMA and CW sep-arately

((r)) MNIST '8' with FGSM, JSMA and CW to-gether

((s)) MNIST '9' with FGSM, JSMA and CW sep-arately

((t)) MNIST '9' with FGSM, JSMA and CW to-gether

Figure 5.13: Experimental data representation space for each class of MNIST digits with adversarial attack Here clean is blue, red is fgsma, green is JSMA and yellow is CW).

detectable.

Model	AUC	CA	F1	Precision	Recall	LogLoss	Specificity
Random Forest	0.973	0.845	0.844	0.844	0.845	0.412	0.926
kNN	0.870	0.643	0.624	0.626	0.643	0.753	0.810
Naive Bayes	0.794	0.562	0.444	0.367	0.562	0.947	0.691
Neural Network	0.815	0.573	0.501	0.629	0.573	0.919	0.763
SVM	0.527	0.523	0.399	0.434	0.523	2.073	0.606
Logistic Regression	0.813	0.566	0.489	0.442	0.566	0.952	0.724

Table 5.4: Adversarial type classification for MNIST dataset for Clean, FGSM, JSMA, and CW.

Model	AUC	CA	F1	Precision	Recall	LogLoss	Specificity
Random Forest	0.998	0.970	0.970	0.970	0.970	0.154	0.985
kNN	0.966	0.844	0.840	0.837	0.844	0.332	0.928
Naive Bayes	0.914	0.737	0.740	0.749	0.737	0.613	0.916
Neural Network	0.951	0.816	0.810	0.807	0.816	0.420	0.919
SVM	0.860	0.302	0.208	0.681	0.302	1.598	0.853
Logistic Regression	0.937	0.790	0.783	0.778	0.790	0.473	0.910

Table 5.5: Binary classification for MNIST dataset for Clean and Adversarial (FGSM, JSMA, and CW).

Model	AUC	CA	F1	Precision	Recall	LogLoss	Specificity
Random Forest	1.000	0.999	0.999	0.999	0.999	0.007	1.000
kNN	0.999	0.984	0.984	0.984	0.984	0.038	0.995
Naive Bayes	0.999	0.983	0.983	0.984	0.983	0.203	0.994
Neural Network	1.000	0.998	0.998	0.998	0.998	0.008	0.999
SVM	0.896	0.590	0.531	0.815	0.590	1.317	0.864
Logistic Regression	0.999	0.983	0.983	0.983	0.983	0.056	0.994

Table 5.6: Adversarial type classification for MNIST dataset for Clean, FGSM, JSMA, and CW after applied histogram and SNR based features.

Model	AUC	CA	F1	Precision	Recall	LogLoss	Specificity
Random Forest	1.000	0.999	0.999	0.999	0.999	0.002	0.998
kNN	1.000	0.998	0.998	0.998	0.998	0.005	0.995
Naive Bayes	0.998	0.999	0.999	0.999	0.999	0.042	0.997
Neural Network	1.000	0.999	0.999	0.999	0.999	0.005	0.997
SVM	0.996	0.753	0.652	0.814	0.753	0.563	0.265
Logistic Regression	0.999	0.997	0.997	0.997	0.997	0.017	0.992

Table 5.7: Binary classification for MNIST dataset for Clean and Adversarial (FGSM, JSMA, and CW) after apply SNR and Histogram features.

In the table 5.4, we used 6 different learning method to differentiate between clean and adversarial attack type using image pixel information, it is evident that random forest performs better than others and in the 5.5, we converted in as binary problem where only clean and adversarial input was classified. SVM method performed very poorly as the representation spaces was not linear. But random forest performs well that other methods. But when we applied SNR and Histogram feature based classification all other method except SVM started to performs well and neural network started to outperforms other methods as presented in table 5.6 and 5.7.

In the table 5.8, we presented the identification of different class labels correctly using

		Predicted										
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	
Actual	**0**	96.7 %	0.2 %	0.4 %	0.2 %	0.5 %	0.2 %	0.3 %	0.7 %	0.6 %	0.2 %	**32**
	1	0.2 %	97.5 %	0.3 %	0.3 %	0.3 %	0.4 %	0.2 %	0.4 %	0.4 %	0.0 %	**33**
	2	0.4 %	0.3 %	95.4 %	0.6 %	0.8 %	0.8 %	0.7 %	0.4 %	0.5 %	0.1 %	**32**
	3	0.6 %	0.3 %	0.5 %	95.4 %	0.7 %	0.6 %	0.4 %	0.5 %	0.7 %	0.2 %	**32**
	4	0.5 %	0.5 %	0.5 %	0.5 %	96.4 %	0.3 %	0.5 %	0.1 %	0.5 %	0.3 %	**32**
	5	0.6 %	0.5 %	0.7 %	0.6 %	0.7 %	95.1 %	0.6 %	0.6 %	0.6 %	0.2 %	**32**
	6	0.5 %	0.4 %	0.5 %	0.3 %	0.5 %	0.5 %	96.5 %	0.2 %	0.4 %	0.1 %	**32**
	7	0.3 %	0.5 %	0.3 %	0.3 %	0.6 %	0.4 %	0.3 %	96.8 %	0.3 %	0.2 %	**32**
	8	0.5 %	0.3 %	0.4 %	0.1 %	0.3 %	0.3 %	0.6 %	0.5 %	96.5 %	0.4 %	**33**
	9	0.1 %	0.2 %	0.2 %	0.3 %	0.2 %	0.1 %	0.2 %	0.2 %	0.2 %	98.5 %	**32**
		3275	**3334**	**3224**	**3225**	**3295**	**3227**	**3271**	**3307**	**3331**	**3266**	**327**

Table 5.8: Confusion matrix of MNIST adversarial input detections using SNR and histogram value.

SNR and histogram value-based checking. We used Random-forest learning. It is seen that some classes are harder to identify than other class labels. As an example, class 2 and 3 is harder than identify adversarial class for input label 9.

5.4.2 Result Analysis

Models Used	MNIST			CFIAR		
	FGSM	**JSMA**	**CW**	**FGSM**	**JSMA**	**CW**
MCD	0.9846	0.99	0.9101	0.8616	0.864	0.7871
OCSVM	0.6851	0.697	0.5421	0.8731	0.535	0.5417
LMDD	0.6673	0.601	0.553	0.5752	0.561	0.5965
LOF	0.997	0.912	0.93	0.8963	0.832	0.8096
COF	0.3991	0.37	0.3568			
CBLOF	0.9866	0.959	0.9			
HBOS	0.9865	0.915	0.9	0.8354	0.859	0.0016
KNN	0.9993	0.909	0.9628	0.9957	0.925	0.0682
SOD	0.3842	0.461	0.3831			
ABOD	0.9994	0.999	0.9776	0.9982	0.922	0.8881
COPD	0.9273	0.996	0.8105	0.8255	0.803	0.7099
SOS	0.4551	0.37				
FB	0.9942	0.99	0.9692	0.8863	0.839	0.7716
IF	0.9933	0.97	0.89	0.8444	0.834	0.6339
LSCP	0.9992	0.9	0.9832	0.8982	0.827	0.78
XGBOD	0.5			0.5	0.59	
LODA	0.9703	0.99	0.91	0.7766	0.661	0.6286
AE	0.6738	0.73	0.62			
VAE	0.8833	0.78	0.7			
SOGAL	0.4	0.3	0.3			
MOGAL	0.2	0.374	0.34			
V-Detector	0.98	0.99	0.94	0.99	0.86	0.78

Table 5.9: Comparison of results with different outlier detection models to compare V-detector NSA performance with other OCC methods.

In the table 5.9, we compared v-detector performance on MNIST digits (0-9) as illustrated in figure 5.13 and 4 class's of CFIAR-10 dataset. Our result shows that v-detector outperforms other out-lire detector consistently for all attack type and dataset.

Attack	CFIAR 'CAT'	CFIAR 'Truck'	CFIAR 'DOG'	CFIAR 'Ship'	Imagenet 'gorilla'	Image
FGSM	0.93	0.92	0.93	0.92	0.68	
BIM	0.90	0.90	0.91	0.71	0.83	
PGD	0.95	0.92	0.92	0.90	0.73	
MBIM	0.91	0.90	0.94	0.96	0.73	
HSJ	0.84	0.65	0.80	0.65		
JSMA	0.7	0.76	0.7	0.73	0.63	
CW	0.76	0.67	0.66	0.62		

Table 5.10: adversarial attacks on CFIAR and Imagenet detection rate (each class has 200 positive and 200 adversarial samples which classifies as that class by a Alexnet for imagenet and VGG-16 for CFIAR).

In the table 5.10, we presented results using similar experiments we used for ground truth experiments. Our performance of CFIAR and IMAGENET is very good compare to the state-of-the-art attack. Also, a good portion of false positives was failed adversarial examples due to perturbation loss while converting physical form. This result verifies that the same filters and histogram, SNR-based methods are applicable for all datasets of the same domain. We also tried to formulate BPDA attack against our defense but failed to formulate the attack.

When we were evaluating our defense against an advanced attack (with very low noises/perturbs) we observed that as all adversarial attack types aim to reduce the perturbation in advanced attack types, the magnitude of perturbation gets so small that they get vanished in rounded values when converting to visual form. Kurkin and Yan Goodfellow in their paper describe this phenomenon of destruction rate by the below equation [112]. Our results in imagenet dataset also effected by this phenomenon.

5.5 Comparison

In table 5.11, we compared our results with other techniques; it is exhibited that our defense's performance is similar to other defense techniques, but our defense technique has some advantages over those like our model does not modify the ML model, it is impossible to have an adaptive attack on our defense. ML model efficiency does not reduce; instead, results get re-verified thus improve trustworthiness. However, the efficiency of our approach largely depends on the individual accuracy of outlier detection methods and noise detection filter sequences.

AML Detection Method	**MNIST**				**CIFAR**				**Avg**
	FGSM	**JSMA**	**HSJ**	**CW**	**FGSM**	**JSMA**	**HSJ**	**CW**	
RF[81]	0.96	0.84	0.98	0.66	0.64	0.63	0.60	0.72	0.77
KNN [81]	0.98	0.80	0.98	0.6	0.56	0.52	0.52	0.69	0.73
SVM [81]	0.98	0.89	0.98	-	0.69	0.69	0.64	0.77	0.81
Feature Squeezing[215]	**1.00**	**1.00**		-	0.20	0.88	0.77	-	0.77
Ensemble [15]	0.99	-	0.45	-	0.99	-	0.42	-	0.71
Decision Mismatch[141]	0.93	0.93	0.91	-	0.93	**0.97**	0.91	-	0.93
Image quality features [6]	**1.00**	0.90	**1.00**	-	0.72	0.70	0.68	-	0.83
(Our Framework)	1.00	1.00	1.00	**1.00**	0.98	0.98	**0.99**	**0.94**	**0.99**

Table 5.11: Here, we provided a comparison with other adversarial input detection techniques based on Accuracy. On average, we outperforms other methods. As examples, our methods work with 99% accuracy in the CFIAR data-set where the feature squeezing technique has 0.88% accuracy.

Adversarial training diminishes the ML model's accuracy and can make the ML model more exposed to generalization [160]. Another disadvantage of Adversarial training based defense techniques is that we need to retrain the model whenever some new attack samples are discovered. It will be hard to update all deployed ML models. Our strategy does not require any dataset not it changes ML anyway, thus no effect on ML model performance. Most pre-processing techniques reduce the adversarial effect before sending it to the ML model. The major drawback of these techniques is that their processing techniques are static; they do not evolve alongside the attack. Our strategy updates itself, it is not vulnerable to this type of adaptive attack. We also have a detection technique module which can detect adaptive attack query.

Distillation techniques work by combining the double model, and the second model uses the first model knowledge to improve accuracy. The black-box attack's recent improvement makes this out-of-date defense [33]. The strong transfer-potential of adversarial samples across neural network models [151] is the main reason for this method's collapse. It is not robust as simplistic variation in a neural network can make the system exposed to attacks [30]. The advantage of our approach over defense distillation is we do not need to modify the neural network. Our proposed approach does not need to know or change any ML model layer. So, our model remains the same for both black box and white box attack methods. [82] concluded that combining/ensemble weak defenses does not automatically improve a system's robustness. Also, the ensemble technique remains static and vulnerable to a new attack. Our proposed solution selects defense technique (filer method and outlier detection method) dynamically, thus it is robust and auto-updating decision boundaries also defend against query-based attacks. Feature squeezing [215] method reduces the data, and it reduces the accuracy of the ML model. There is no such reduction in actual model accuracy in our proposed solution. [171] proposed a mechanism to leverage the power of Generative Adversarial Networks to decrease adversarial perturbations' efficiency. The GAN efficiency depends on the GAN training, which is computationally complicated and needs proper datasets, whereas our system does not need a complicated training method.

5.6 Robustness Evaluation

Many adversarial defenses are often empirically shown to be robust against the existing attacks, however new stronger attacks are later found to break such defense. For instance, defensive distillation and adversarial training against the FGSM) were shown to be useless against more potent attacks, for example, Carlini-wagner attack. So, we need to develop successful defenses against all attacks within a particular class. But compute this is computationally expensive due to the nature of model complexity and the existence of vastly different types of machine learning models.

According to the literature[178], We can define Robustness Verification as a process that certifies the lower bound of a learning model strength against adversarial attacks in varying constrained conditions. An well-known example is L∞-bounded attack[47]. There are two primary categories of robustness verification methods. These are deterministic and probabilistic verification. Deterministic verification approaches able to identify strength of input against adversarial attacks. But the probabilistic verification methods only work with a specific probability (e.g. 99.9%).

Li et al.[120] stated that "An algorithm A is certified as a robustness verification approach, if for any (x_0, y_0), as long as there exists $x \in B_p(x_0)$ with $F_\theta(x) \neq y_0$ (adversarial example), $A(f_\theta, x_0, y_0, \epsilon) = false$(deterministic verification) or $Pr[A(f_\theta, x_0, y_0, \epsilon) = false] \geq 1 - \alpha$ (probabilistic verification), where α is a pre-defined small threshold. By definition, a robustness verification approach A provides a sufficient condition for model robustness. If A also provides a necessary condition for model robustness, A is said to be complete, otherwise incomplete."

The robust training approaches are usually extended from adversarial training. Adversarial training[47] is a powerful defense which approximately solves the min-max problem where we need reduce the regular loss function such as cross entropy. In ML, we have an estimator δ that is used to estimate a $\theta \in \Theta$ We also assume a risk function $R(\theta, \delta)$, usually specified as the integral of a loss function. In this framework, $\tilde{\delta}$ is called minmax problem if it satisfies

$_\theta R(\theta, \tilde{\delta}) = \infty_{\delta_\theta} R(\theta, \delta)$[]

The inner maximization of Minmax problem is hard to solve due to nonconvexity, and is usually approximated by empirical attacks such as basic iterative attack methods. In the figure 5.14 we describes several methods to determine robustness of adversarial training process however as these are only work for whitebox models. Our defense model works on only input data and and we consider learning models as a blackbox which are not comparable to existing defense model robustness lower bound. In the table 5.12, different probabilistic method was presented which used to evaulate robustness in adversarial training. As our method is not adversarial training we can not use these either.

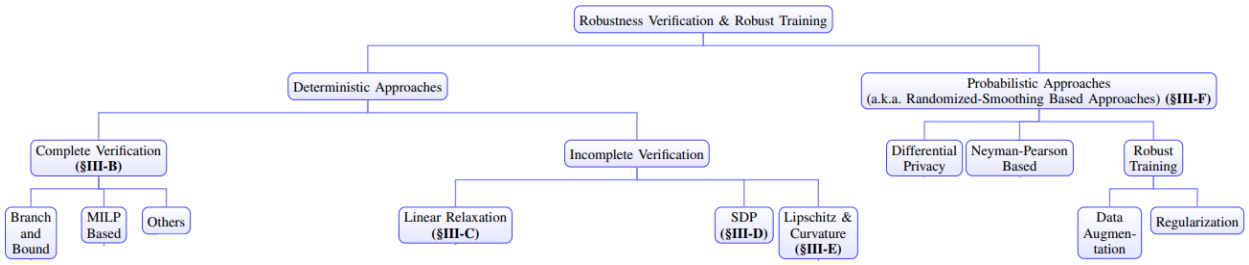


Figure 5.14: Different robustness verification techniques [120]

Robustness Verification Approaches		Adversaries in Evaluation
Verification	Differential Privacy Based [116]	L1,L2
	Neyman-Pearson Based [43]	L1, L2, L∞
	f -Divergence [40]	L1
	Re´nyi divergence Based [118]	L2
Robust Training	Data Augmentation[163]	L1, L2, L∞
	Adversarial Training citesilva2020opportunities,li2020sok	L2
	Adversarial + Pretraining[178, 120]	L2
	MACER [223]	L2
	ADRE [57]	L2

Table 5.12: Different probabilistic method for adversarial robustness

Carlini mentioned [**carlinini2019evaulating**], evaluating adversarial defense robustness should be measure by following a collection of recommendations that they identified as common flaws in adversarial example defense evaluations. In our machine learning adversarial threat model we consider all of these flaws.

Further study is possible to explore measurement of robustness lower bound for input inspection based strategy, We hope this would be a research opportunities for future researchers in this field.

5.7 Summary

In summary, any commercial product that is using advanced machine/deep/reinforcement learning can benefit from our innovative DF technique.

- Use of commutative dual filtering technique in any AI/ML–based utility applications.

- Use of negative filtering will prevent Trojan AI to change decision resulting in robust AI/ML systems.
- Easy to incorporate in existing and future ML systems will increase adoption and deploy ability.
- Enhanced performance/accuracy and robustness of ML products and online services will increase in diverse applications.
- Improved security will result in quality of experience of users.

Chapter 6

Real-World Application

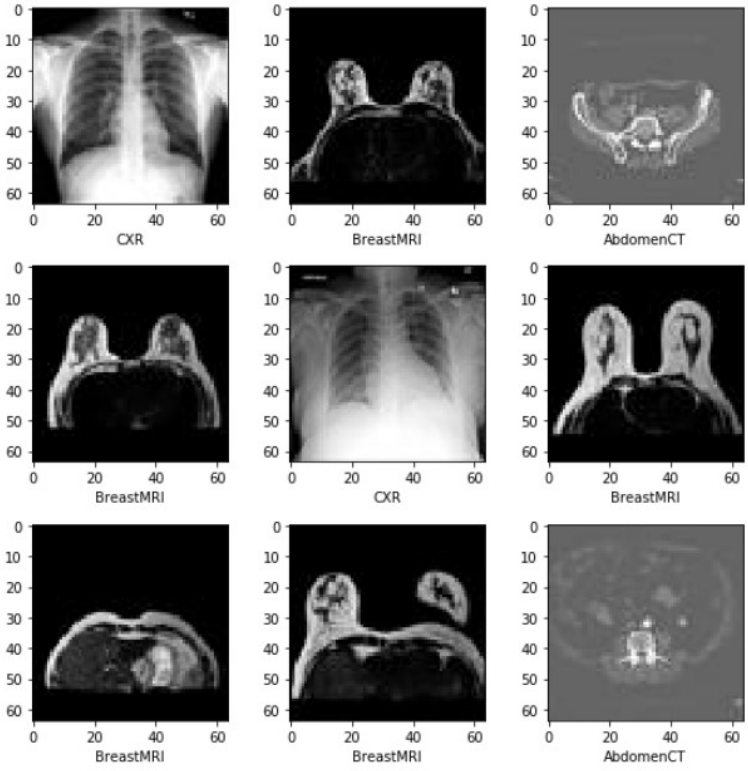


Figure 6.1: Medical Imaging Dataset

6.1 Proof-of-the-Concept Prototype

I developed a Django web service which can evaluate medical image if they are adversarial or not. This dataset is a simple MNIST-style medical images in 64x64 dimension; There were originally taken from other datasets and processed into such style. There are 58954 medical images belonging to 6 classes. They are:

- AbdomenCT
- BreastMRI
- CXR
- ChestCT
- Hand
- HeadCT

6.2 Experiments

We experiment with FGSM, CW and JSMA dataset, and the accuracy of detection rate is 98% by input filters and 100% by the output filter.

6.3 Scope and Limitations

This service provide a Rest API which can be use by any medical ML model to check the input and the output. The work of output filter is limited here as output filters work only based on medical image class not the diagnosis result class.

6.4 Lessons Learned

I have designed an adaptive filtering methodology that does not require any modification to the ML model or information inside the ML model. Our strategy can implement in any ML-based system without costly pre-training. It is to be noted that current adaptive attacks are ineffective in our defense. As our method verify the input and the ML model output with non-obvious diverse inspection and secondary (outlier) detection, the results exhibited that it could increase the trustworthiness of the ML model applications. I focused our experiments on the computer vision domain, but our techniques are suitable for other domains (audio, text, time series). I am planning to expand our test in different domains and enrich our filter collection for better performance. I plan to deploy this as a library so any ML model developer can use our framework as an extension for his ML for security purposes. Our technique can be tuned between speed and accuracy; also, as it is independent of the ML, making the framework suitable for privacy.

References

[1] George A Adam et al. “Stochastic Combinatorial Ensembles for Defending Against Adversarial Examples.” In: *arXiv preprint arXiv:1808.06645* (2018), pp. 1–15.

[2] Charu C Aggarwal. “Outlier analysis.” In: *Data mining*. Springer. 2015, pp. 237–263.

[3] Md Manjurul Ahsan et al. “Applications and Evaluations of Bio-Inspired approaches in Cloud Security: A Review.” In: *IEEE Access* (2020).

[4] Jonathan Aigrain and Marcin Detyniecki. “Detecting Adversarial Examples and Other Misclassifications in Neural Networks by Introspection.” In: *arXiv preprint arXiv:1905.09186* (2019).

[5] Z. Akhtar, J. Monteiro, and T. H. Falk. “Adversarial Examples Detection Using No-Reference Image Quality Features.” In: *2018 International Carnahan Conference on Security Technology (ICCST)*. 2018, pp. 1–5.

[6] Zahid Akhtar, João Monteiro, and Tiago H Falk. “Adversarial Examples Detection Using No-Reference Image Quality Features.” In: *2018 International Carnahan Conference on Security Technology (ICCST)*. IEEE. 2018, pp. 1–5.

[7] Richard D Alexander and Srinivas Panguluri. “Cybersecurity terminology and frameworks.” In: *Cyber-Physical Security*. Springer, 2017, pp. 19–47.

[8] Moustafa Alzantot et al. “Generating natural language adversarial examples.” In: *arXiv preprint arXiv:1804.07998* (2018).

[9] Anelia Angelova, Yaser Abu-Mostafam, and Pietro Perona. “Pruning training sets for learning of object categories.” In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*. Vol. 1. IEEE. 2005, pp. 494–501.

[10] Marc Antonini et al. “Image coding using wavelet transform.” In: *IEEE Transactions on image processing* 1.2 (1992), pp. 205–220.

[11] Jarosław Arabas and Karol Opara. “Population diversity of non-elitist evolutionary algorithms in the exploration phase.” In: *IEEE Transactions on Evolutionary Computation* (2019).

[12] Andreas Arning, Rakesh Agrawal, and Prabhakar Raghavan. “A Linear Method for Deviation Detection in Large Databases.” In: *KDD*. Vol. 1141. 50. 1996, pp. 972–981.

[13] Anish Athalye and Nicholas Carlini. “On the robustness of the cvpr 2018 white-box adversarial example defenses.” In: *arXiv preprint arXiv:1804.03286* (2018).

[14] Anish Athalye, Nicholas Carlini, and David Wagner. “Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples.” In: *arXiv preprint arXiv:1802.00420* (2018).

[15] Alexander Bagnall, Razvan Bunescu, and Gordon Stewart. “Training ensembles to detect adversarial examples.” In: *arXiv preprint arXiv:1712.04006* (2017).

[16] Donald R Barr and Teddy Davidson. “A Kolmogorov-Smirnov test for censored samples.” In: *Technometrics* 15.4 (1973), pp. 739–757.

[17] David M Blei, Michael I Jordan, et al. “Variational inference for Dirichlet process mixtures.” In: *Bayesian analysis* 1.1 (2006), pp. 121–143.

[18] John Bradshaw, Alexander G de G Matthews, and Zoubin Ghahramani. “Adversarial examples, uncertainty, and transfer testing robustness in gaussian process hybrid deep networks.” In: *arXiv preprint arXiv:1707.02476* (2017), pp. 1–33.

[19] John Bradshaw, Alexander G de G Matthews, and Zoubin Ghahramani. “Adversarial examples, uncertainty, and transfer testing robustness in Gaussian process hybrid deep networks.” In: *arXiv preprint arXiv:1707.02476* (2017).

[20] Wieland Brendel, Jonas Rauber, and Matthias Bethge. “Decision-based adversarial attacks: Reliable attacks against black-box machine learning models.” In: *arXiv preprint arXiv:1712.04248* (2017).

[21] Heinz Breu et al. “Linear time Euclidean distance transform algorithms.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17.5 (1995), pp. 529–533.

[22] Markus M Breunig et al. “LOF: identifying density-based local outliers.” In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 2000, pp. 93–104.

[23] Carla E Brodley and Mark A Friedl. “Identifying mislabeled training data.” In: *Journal of artificial intelligence research* 11 (1999), pp. 131–167.

[24] TB Brown et al. “Adversarial patch. arxiv e-prints (dec. 2017).” In: *arXiv preprint cs.CV/1712.09665* 1.2 (2017), p. 4.

[25] Michael Brückner and Tobias Scheffer. “Stackelberg games for adversarial prediction problems.” In: *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2011, pp. 547–555.

[26] Wilhelm Burger and Mark J Burge. *Digital image processing: an algorithmic introduction using Java*. Springer, 2016.

[27] Nicholas Carlini. “Lessons Learned from Evaluating the Robustness of Defenses to Adversarial Examples.” In: (2019).

[28] Nicholas Carlini and David Wagner. “Adversarial examples are not easily detected: Bypassing ten detection methods.” In: *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. 2017, pp. 3–14.

[29] Nicholas Carlini and David Wagner. “Audio adversarial examples: Targeted attacks on speech-to-text.” In: *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE. 2018, pp. 1–7.

[30] Nicholas Carlini and David Wagner. “Defensive distillation is not robust to adversarial examples.” In: *arXiv preprint arXiv:1607.04311* (2016).

[31] Nicholas Carlini and David Wagner. “Magnet and" efficient defenses against adversarial attacks" are not robust to adversarial examples.” In: *arXiv preprint arXiv:1711.08478* (2017).

[32] Nicholas Carlini et al. “On evaluating adversarial robustness.” In: *arXiv preprint arXiv:1902.06705* (2019).

[33] Anirban Chakraborty et al. “Adversarial attacks and defences: A survey.” In: *arXiv preprint arXiv:1810.00069* (2018).

[34] Olivier Chapelle, Patrick Haffner, and Vladimir N Vapnik. “Support vector machines for histogram-based image classification.” In: *IEEE transactions on Neural Networks* 10.5 (1999), pp. 1055–1064.

[35] Jianbo Chen, Michael I Jordan, and Martin J Wainwright. “Hopskipjumpattack: A query-efficient decision-based attack.” In: *arXiv preprint arXiv:1904.02144* (2019).

[36] Jiefeng Chen et al. “ReabsNet: Detecting and Revising Adversarial Examples.” In: *arXiv preprint arXiv:1712.08250* (2017).

[37] Pin-Yu Chen et al. “Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models.” In: *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. 2017, pp. 15–26.

[38] Steven Chen, Nicholas Carlini, and David Wagner. “Stateful Detection of Black-Box Adversarial Attacks.” In: *arXiv preprint arXiv:1907.05587* (2019).

[39] Yunqiang Chen, Xiang Sean Zhou, and Thomas S Huang. “One-class SVM for learning in image retrieval.” In: *Proceedings 2001 International Conference on Image Processing (Cat. No. 01CH37205)*. Vol. 1. IEEE. 2001, pp. 34–37.

[40] Hao Cheng et al. “Posterior differential regularization with f-divergence for improving model robustness.” In: *arXiv preprint arXiv:2010.12638* (2020).

[41] Runwei Cheng and Mitsuo Gen. “Crossover on intensive search and traveling salesman problem.” In: *Computers & Industrial Engineering* 27.1-4 (1994), pp. 485–488.

[42] Ping-Yeh Chiang et al. “Certified defenses for adversarial patches.” In: *arXiv preprint arXiv:2003.06693* (2020).

[43] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. “Certified adversarial robustness via randomized smoothing.” In: *International Conference on Machine Learning*. PMLR. 2019, pp. 1310–1320.

[44] Francesco Crecchi, Davide Bacciu, and Battista Biggio. “Detecting Adversarial Examples through Nonlinear Dimensionality Reduction.” In: *arXiv preprint arXiv:1904.13094* (2019).

[45] Dipankar Dasgupta. “Advances in artificial immune systems.” In: *IEEE computational intelligence magazine* 1.4 (2006), pp. 40–49.

[46] Dipankar Dasgupta. “Handling deceptive problems using a different genetic search.” In: *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*. IEEE. 1994, pp. 807–811.

[47] Dipankar Dasgupta, Zahid Akhtar, and Sen Sajib. “Machine learning in cybersecurity: a comprehensive survey.” In: (2020).

[48] Dipankar Dasgupta, Zhou Ji, and Fabio Gonzalez. “Artificial immune system (AIS) research in the last five years.” In: *The 2003 Congress on Evolutionary Computation, 2003. CEC’03.* Vol. 1. IEEE. 2003, pp. 123–130.

[49] Dipankar Dasgupta and Fernando Nino. *Immunological computation: theory and applications*. CRC press, 2008.

[50] Dipankar Dasgupta et al. “Negative selection algorithm for aircraft fault detection.” In: *International Conference on Artificial Immune Systems*. Springer. 2004, pp. 1–13.

[51] Sumanth Dathathri et al. “Detecting Adversarial Examples via Neural Fingerprinting.” In: *arXiv preprint arXiv:1803.03870* (2018).

[52] Thomas G Dietterich and Ghulum Bakiri. “Solving multiclass learning problems via error-correcting output codes.” In: *Journal of artificial intelligence research* 2 (1994), pp. 263–286.

[53] Thomas G Dietterich, Hermann Hild, and Ghulum Bakiri. “A comparison of ID3 and backpropagation for English text-to-speech mapping.” In: *Machine Learning* 18.1 (1995), pp. 51–80.

[54] Gavin Weiguang Ding, Luyu Wang, and Xiaomeng Jin. “AdverTorch v0.1: An Adversarial Robustness Toolbox based on PyTorch.” In: *arXiv preprint arXiv:1902.07623* (2019).

[55] Yinpeng Dong et al. “Discovering adversarial examples with momentum.” In: *arXiv preprint arXiv:1710.06081* (2017).

[56] Kevin Eykholt et al. “Robust physical-world attacks on deep learning visual classification.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 1625–1634.

[57] Huijie Feng et al. “Regularized training and tight certification for randomized smoothed classifier with provable robustness.” In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04. 2020, pp. 3858–3865.

[58] Jeff Forcier, Paul Bissex, and Wesley J Chun. *Python web development with Django*. Addison-Wesley Professional, 2008.

[59] Stephanie Forrest et al. “Self-nonself discrimination in a computer.” In: *Proceedings of 1994 IEEE computer society symposium on research in security and privacy*. Ieee. 1994, pp. 202–212.

[60] Bernhard Froba and Andreas Ernst. “Face detection with the modified census transform.” In: *Sixth IEEE International Conference on Automatic Face and Gesture Recognition, 2004. Proceedings.* IEEE. 2004, pp. 91–96.

[61] Dragan Gamberger, Nada Lavrač, and Sašo Džeroski. “Noise elimination in inductive concept learning: A case study in medical diagnosis.” In: *International Workshop on Algorithmic Learning Theory*. Springer. 1996, pp. 199–212.

[62] Markus Goldstein and Andreas Dengel. “Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm.” In: *KI-2012: Poster and Demo Track* (2012), pp. 59–63.

[63] *gongzhitaao/tensorflowadversarial*.

[64] Fabio Gonzalez, Dipankar Dasgupta, and Luis Fernando Niño. “A randomized real-valued negative selection algorithm.” In: *International Conference on Artificial Immune Systems*. Springer. 2003, pp. 261–272.

[65] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and Harnessing Adversarial Examples*. 2014. arXiv: `1412.6572 [stat.ML]`.

[66] Megha Goyal. “Morphological image processing.” In: *IJCST* 2.4 (2011).

[67] Harry N Gross and John R Schott. “Application of spectral mixture analysis and image fusion techniques for image sharpening.” In: *Remote Sensing of Environment* 63.2 (1998), pp. 85–94.

[68] Kathrin Grosse et al. “On the (statistical) detection of adversarial examples.” In: *arXiv preprint arXiv:1702.06280* (2017).

[69] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. “Badnets: Identifying vulnerabilities in the machine learning model supply chain.” In: *arXiv preprint arXiv:1708.06733* (2017).

[70] Zhenhua Guo, Lei Zhang, and David Zhang. “A completed modeling of local binary pattern operator for texture classification.” In: *IEEE transactions on image processing* 19.6 (2010), pp. 1657–1663.

[71] KD Gupta and S Sen. “A Genetic Algorithm Approach to Regenerate Image from a Reduce Scaled Image Using Bit Data Count.” In: *BRAIN. Broad Research in Artificial Intelligence and Neuroscience* 9.2 (2018), pp. 34–44.

[72] Kishor Datta Gupta and Dipankar Dasgupta. “Negative Selection Algorithm Research and Applications in the last decade: A Review.” In: *arXiv preprint arXiv:2105.06109* (2021).

[73] Kishor Datta Gupta and Dipankar Dasgupta. “Using Negative Detectors for Identifying Adversarial Data Manipulation in Machine Learning.” In: ().

[74] Kishor Datta Gupta and Dipankar Dasgupta. “Who is Responsible for Adversarial Defense?” In: *arXiv preprint arXiv:2106.14152* (2021).

[75] Kishor Datta Gupta, Dipankar Dasgupta, and Zahid Akhtar. “Adversarial Input Detection Using Image Processing Techniques (IPT).” In: *2020 11th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*. IEEE. 2020, pp. 0309–0315.

[76] Kishor Datta Gupta, Dipankar Dasgupta, and Zahid Akhtar. “Applicability issues of evasion-based adversarial attacks and mitigation techniques.” In: *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. 2020, pp. 1506–1515.

[77] Kishor Datta Gupta, Dipankar Dasgupta, and Zahid Akhtar. “Determining Sequence of Image Processing Technique (IPT) to Detect Adversarial Attacks.” In: *arXiv preprint arXiv:2007.00337* (2020).

[78] Kishor Datta Gupta, Dipankar Dasgupta, and Sajib Sen. “Smart Crowdsourcing Based Content Review System (SCCRS): An Approach to Improve Trustworthiness of Online Contents.” In: *International Conference on Computational Social Networks*. Springer. 2018, pp. 523–535.

[79] Lars Kai Hansen and Peter Salamon. “Neural network ensembles.” In: *IEEE Transactions on Pattern Analysis & Machine Intelligence* 10 (1990), pp. 993–1001.

[80] Johanna Hardin and David M Rocke. “Outlier detection in the multiple cluster setting using the minimum covariance determinant estimator.” In: *Computational Statistics & Data Analysis* 44.4 (2004), pp. 625–638.

[81] Jamie Hayes and George Danezis. “Machine learning as an adversarial service: Learning black-box adversarial examples.” In: *arXiv preprint arXiv:1708.05207* 2 (2017).

[82] Warren He et al. “Adversarial example defense: Ensembles of weak defenses are not strong.” In: *11th {USENIX} Workshop on Offensive Technologies ({WOOT} 17)*. 2017.

[83] Zengyou He, Xiaofei Xu, and Shengchun Deng. “Discovering cluster-based local outliers.” In: *Pattern Recognition Letters* 24.9-10 (2003), pp. 1641–1650.

[84] Geoffrey E Hinton and Ruslan R Salakhutdinov. “Reducing the dimensionality of data with neural networks.” In: *science* 313.5786 (2006), pp. 504–507.

[85] Mia Hubert, Michiel Debruyne, and Peter J Rousseeuw. “Minimum covariance determinant and extensions.” In: *Wiley Interdisciplinary Reviews: Computational Statistics* 10.3 (2018), e1421.

[86] Olakunle Ibitoye et al. “The Threat of Adversarial Attacks on Machine Learning in Network Security–A Survey.” In: *arXiv preprint arXiv:1911.02621* (2019).

[87] Andrew Ilyas et al. “Adversarial examples are not bugs, they are features.” In: *Advances in Neural Information Processing Systems*. 2019, pp. 125–136.

[88] Andrew Ilyas et al. “Black-box adversarial attacks with limited queries and information.” In: *arXiv preprint arXiv:1804.08598* (2018).

[89] Anil K Jain, Jianchang Mao, and K Moidin Mohiuddin. “Artificial neural networks: A tutorial.” In: *Computer* 29.3 (1996), pp. 31–44.

[90] JHM Janssens et al. “Stochastic outlier selection.” In: *tech. rep.* (2012).

[91] Zhou Ji and Dipankar Dasgupta. “Estimating the detector coverage in a negative selection algorithm.” In: *Proceedings of the 7th annual conference on Genetic and evolutionary computation*. 2005, pp. 281–288.

[92] Zhou Ji and Dipankar Dasgupta. “Real-valued negative selection algorithm with variable-sized detectors.” In: *Genetic and Evolutionary Computation Conference*. Springer. 2004, pp. 287–298.

[93] Zhou Ji and Dipankar Dasgupta. “Revisiting negative selection algorithms.” In: *Evolutionary Computation* 15.2 (2007), pp. 223–251.

[94] Zhou Ji and Dipankar Dasgupta. “V-detector: An efficient negative selection algorithm with “probably adequate” detector coverage.” In: *Information sciences* 179.10 (2009), pp. 1390–1406.

[95] Zhou Ji et al. “A boundary-aware negative selection algorithm.” In: *Proceedings of the 9th International Conference on Artificial Intelligence and Soft Computing, ACTA Press*. 2005.

[96] Juho Kannala and Esa Rahtu. “Bsif: Binarized statistical image features.” In: *Proceedings of the 21st international conference on pattern recognition (ICPR2012)*. IEEE. 2012, pp. 1363–1366.

[97] Joanna Karbowska-Chilińska and Paweł Zabielski. “A genetic algorithm vs. local search methods for solving the orienteering problem in large networks.” In: *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*. Springer. 2012, pp. 11–20.

[98] Raimi Karim. *Illustrated: 10 CNN Architectures*. 2019. URL: https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d.

[99] Danny Karmon, Daniel Zoran, and Yoav Goldberg. "Lavan: Localized and visible adversarial noise." In: *arXiv preprint arXiv:1801.02608* (2018).

[100] Guy Katz et al. "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks." In: *Computer Aided Verification* (2017). Ed. by Rupak Majumdar and Viktor Kunčak, pp. 97–117.

[101] Ziv Katzir and Yuval Elovici. "Detecting Adversarial Perturbations Through Spatial Behavior in Activation Spaces." In: *arXiv preprint arXiv:1811.09043* (2018).

[102] Nikhil Ketkar. "Introduction to pytorch." In: *Deep learning with python*. Springer, 2017, pp. 195–208.

[103] McClaning Kevin. *Radio Receiver Design*. 2000.

[104] JooSeuk Kim and Clayton D Scott. "Robust kernel density estimation." In: *The Journal of Machine Learning Research* 13.1 (2012), pp. 2529–2565.

[105] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes." In: *arXiv preprint arXiv:1312.6114* (2013).

[106] Utku Kose. "Techniques for Adversarial Examples Threatening the Safety of Artificial Intelligence Based Systems." In: *arXiv preprint arXiv:1910.06907* (2019).

[107] Vassili Kovalev, Alexander Kalinovsky, and Sergey Kovalev. "Deep learning with theano, torch, caffe, tensorflow, and deeplearning4j: Which one is the best in speed and accuracy?" In: (2016).

[108] Hans-Peter Kriegel, Matthias Schubert, and Arthur Zimek. "Angle-based outlier detection in high-dimensional data." In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2008, pp. 444–452.

[109] Hans-Peter Kriegel et al. “Outlier detection in axis-parallel subspaces of high dimensional data.” In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer. 2009, pp. 831–838.

[110] Volodymyr Kuleshov et al. “Adversarial examples for natural language classification problems.” In: (2018).

[111] GNC Kumar. “Artificial intelligence: Definition, types, examples, technologies.” In: *Accessed March* 1 (2018), p. 2019.

[112] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. “Adversarial examples in the physical world.” In: *arXiv preprint arXiv:1607.02533* (2016).

[113] Siddique Latif, Rajib Rana, and Junaid Qadir. “Adversarial machine learning and speech emotion recognition: Utilizing generative adversarial networks for robustness.” In: *arXiv preprint arXiv:1811.11402* (2018).

[114] Aleksandar Lazarevic and Vipin Kumar. “Feature bagging for outlier detection.” In: *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. 2005, pp. 157–166.

[115] J Le. “The 10 Deep Learning Methods AI Practitioners Need to Apply.” In: *Cracking the data science interview* (2017).

[116] Mathias Lecuyer et al. “Certified robustness to adversarial examples with differential privacy.” In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2019, pp. 656–672.

[117] Yunseong Lee et al. “From the Edge to the Cloud: Model Serving in ML. NET.” In: *IEEE Data Eng. Bull.* 41.4 (2018), pp. 46–53.

[118] Bai Li et al. “Certified adversarial robustness with additive noise.” In: *arXiv preprint arXiv:1809.03113* (2018).

[119] Dan Li et al. “Anomaly detection with generative adversarial networks for multivariate time series.” In: *arXiv preprint arXiv:1809.04758* (2018).

[120] Linyi Li et al. “Sok: Certified robustness for deep neural networks.” In: *arXiv preprint arXiv:2009.04131* (2020).

[121] Zheng Li et al. “COPOD: copula-based outlier detection.” In: *arXiv preprint arXiv:2009.09463* (2020).

[122] Arnaud Liefooghe et al. “Landscape-aware performance prediction for evolutionary multi-objective optimization.” In: *IEEE Transactions on Evolutionary Computation* (2019).

[123] Jae S Lim. “Two-dimensional signal and image processing.” In: *ph* (1990).

[124] Jiayang Liu et al. “Detection Based Defense Against Adversarial Examples From the Steganalysis Point of View.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4825–4834.

[125] Xin Liu et al. “Dpatch: An adversarial patch attack on object detectors.” In: *arXiv preprint arXiv:1806.02299* (2018).

[126] Yanpei Liu et al. “Delving into transferable adversarial examples and black-box attacks.” In: *arXiv preprint arXiv:1611.02770* (2016).

[127] Yezheng Liu et al. “Generative adversarial active learning for unsupervised outlier detection.” In: *IEEE Transactions on Knowledge and Data Engineering* (2019).

[128] Yingqi Liu et al. “Trojaning attack on neural networks.” In: (2017).

[129] Jiajun Lu, Theerasit Issaranon, and David Forsyth. “Safetynet: Detecting and rejecting adversarial examples robustly.” In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 446–454.

[130] Jiajun Lu et al. “No need to worry about adversarial examples in object detection in autonomous vehicles.” In: *arXiv preprint arXiv:1707.03501* (2017).

[131] Chen Ma et al. “MetaAdvDet: Towards Robust Detection of Evolving Adversarial Attacks.” In: *arXiv preprint arXiv:1908.02199* (2019).

[132] Aleksander Madry et al. “Towards deep learning models resistant to adversarial attacks.” In: *arXiv preprint arXiv:1706.06083* (2017).

[133] Stephen Marsland, Jonathan Shapiro, and Ulrich Nehmzow. “A self-organising network that grows when required.” In: *Neural networks* 15.8-9 (2002), pp. 1041–1058.

[134] Henrik H Martens. “Two notes on machine “Learning”.” In: *Information and Control* 2.4 (1959), pp. 364–379.

[135] JJ McCarthy, ML Minsky, and N Rochester. *Artificial intelligence*. Tech. rep. Research Laboratory of Electronics (RLE) at the Massachusetts Institute of . . ., 1959.

[136] Dongyu Meng and Hao Chen. “Magnet: a two-pronged defense against adversarial examples.” In: *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 2017, pp. 135–147.

[137] Jan Hendrik Metzen et al. “On detecting adversarial perturbations.” In: *arXiv preprint arXiv:1702.04267* (2017).

[138] T. Miyato et al. “Virtual Adversarial Training: A Regularization Method for Supervised and Semi-Supervised Learning.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41.8 (2019), pp. 1979–1993.

[139] Takeru Miyato, Andrew M Dai, and Ian Goodfellow. “Adversarial training methods for semi-supervised text classification.” In: *arXiv preprint arXiv:1605.07725* (2016).

[140] Takeru Miyato et al. “Virtual adversarial training: a regularization method for supervised and semi-supervised learning.” In: *IEEE transactions on pattern analysis and machine intelligence* 41.8 (2018), pp. 1979–1993.

[141] João Monteiro, Zahid Akhtar, and Tiago H Falk. “Generalizable adversarial examples detection based on bi-model decision mismatch.” In: *arXiv preprint arXiv:1802.07770* (2018).

[142] Young-Il Moon, Balaji Rajagopalan, and Upmanu Lall. “Estimation of mutual information using kernel density estimators.” In: *Physical Review E* 52.3 (1995), p. 2318.

[143] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. “Deepfool: a simple and accurate method to fool deep neural networks.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2574–2582.

[144] Seyed-Mohsen Moosavi-Dezfooli et al. “Universal adversarial perturbations.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1765–1773.

[145] Nina Narodytska and Shiva Prasad Kasiviswanathan. “Simple black-box adversarial perturbations for deep networks.” In: *arXiv preprint arXiv:1612.06299* (2016).

[146] Saakshi Narula, Arushi Jain, et al. “Cloud computing security: amazon web service.” In: *2015 Fifth International Conference on Advanced Computing & Communication Technologies*. ieee. 2015, pp. 501–505.

[147] Anh Nguyen, Jason Yosinski, and Jeff Clune. “Deep neural networks are easily fooled: High confidence predictions for unrecognizable images.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 427–436.

[148] Huy H Nguyen et al. “Detecting and Correcting Adversarial Images Using Image Processing Operations and Convolutional Neural Networks.” In: *arXiv preprint arXiv:1912.05391* (2019).

[149] Maria-Irina Nicolae et al. “Adversarial Robustness Toolbox v1.1.1.” In: *CoRR* 1807.01069 (2018). URL: `https://arxiv.org/pdf/1807.01069`.

[150] Tianyu Pang et al. “Towards robust detection of adversarial examples.” In: *Advances in Neural Information Processing Systems*. 2018, pp. 4579–4589.

[151] Nicolas Papernot et al. “Distillation as a defense to adversarial perturbations against deep neural networks.” In: *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2016, pp. 582–597.

[152] Nicolas Papernot et al. “Distillation as a defense to adversarial perturbations against deep neural networks.” In: *IEEE Symposium on Security and Privacy (SP)* (2016), pp. 582–597.

[153] Nicolas Papernot et al. “Technical Report on the CleverHans v2.1.0 Adversarial Examples Library.” In: *arXiv preprint arXiv:1610.00768* (2018).

[154] Nicolas Papernot et al. “The limitations of deep learning in adversarial settings.” In: *2016 IEEE European symposium on security and privacy (EuroS&P)*. IEEE. 2016, pp. 372–387.

[155] Andrea Paudice et al. “Detection of adversarial training examples in poisoning attacks through anomaly detection.” In: *arXiv preprint arXiv:1802.03041* (2018).

[156] Antonio F Pérez-Rendón and Rafael Robles. “The convolution theorem for the continuous wavelet tranform.” In: *Signal processing* 84.1 (2004), pp. 55–67.

[157] Fabio Pierazzi et al. “Intriguing properties of adversarial ML attacks in the problem space.” In: *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2020, pp. 1332–1349.

[158] Aaditya Prakash et al. “Deflecting adversarial attacks with pixel deflection.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8571–8580.

[159] C Rader and NJIToA Brenner. “A new principle for fast Fourier transformation.” In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 24.3 (1976), pp. 264–266.

[160] Aditi Raghunathan et al. “Adversarial Training Can Hurt Generalization.” In: *arXiv preprint arXiv:1906.06032* (2019).

[161] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. “Efficient algorithms for mining outliers from large data sets.” In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 2000, pp. 427–438.

[162] Jonas Rauber, Wieland Brendel, and Matthias Bethge. “Foolbox: A python toolbox to benchmark the robustness of machine learning models.” In: *arXiv preprint arXiv:1707.04131* (2017).

[163] Sylvestre-Alvise Rebuffi et al. “Fixing data augmentation to improve adversarial robustness.” In: *arXiv preprint arXiv:2103.01946* (2021).

[164] Shuhuai Ren et al. “Generating natural language adversarial examples through probability weighted word saliency.” In: *Proceedings of the 57th annual meeting of the association for computational linguistics*. 2019, pp. 1085–1097.

[165] Douglas A Reynolds. “Gaussian Mixture Models.” In: *Encyclopedia of biometrics* 741 (2009).

[166] Edwina L Rissland and David B Skalak. “CABARET: rule interpretation in a hybrid architecture.” In: *International journal of man-machine studies* 34.6 (1991), pp. 839–887.

[167] Lukas Ruff et al. “Deep one-class classification.” In: *International conference on machine learning*. 2018, pp. 4393–4402.

[168] Sara Sabour et al. “Adversarial manipulation of deep representations.” In: *arXiv preprint arXiv:1511.05122* (2015).

[169] Nafiz Sadman et al. “Detect Review Manipulation by Leveraging Reviewer Historical Stylometrics in Amazon, Yelp, Facebook and Google Reviews.” In: *Proceedings of the 2020 The 6th International Conference on E-Business and Applications*. 2020, pp. 42–47.

[170] Sumit Saha. *A comprehensive guide to convolutional neural networks–the ELI5 way*. 2018.

[171] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. “Defense-gan: Protecting classifiers against adversarial attacks using generative models.” In: *arXiv preprint arXiv:1805.06605* (2018).

[172] Spyridon Samonas and David Coss. “THE CIA STRIKES BACK: REDEFINING CONFIDENTIALITY, INTEGRITY AND AVAILABILITY IN SECURITY.” In: *Journal of Information System Security* 10.3 (2014).

[173] Thomas Schlegl et al. “Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery.” In: *Information Processing in Medical Imaging*. Ed. by Marc Niethammer et al. Cham: Springer International Publishing, 2017, pp. 146–157. ISBN: 978-3-319-59050-9.

[174] Sajib Sen et al. “AGenetic ALGORITHM APPROACH TO OPTIMIZE DISPATCHING FOR A MICROGRID ENERGY SYSTEM WITH RENEWABLE ENERGY SOURCES.” In: ().

[175] Burr Settles. “Active learning literature survey.” In: (2009).

[176] Uri Shaham et al. “Defending against adversarial images using basis functions transformations.” In: *arXiv preprint arXiv:1803.10840* (2018).

[177] Mahmood Sharif et al. “Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition.” In: *Proceedings of the 2016 acm sigsac conference on computer and communications security*. 2016, pp. 1528–1540.

[178] Samuel Henrique Silva and Peyman Najafirad. “Opportunities and challenges in deep learning adversarial robustness: A survey.” In: *arXiv preprint arXiv:2007.00753* (2020).

[179] Igor Simonovski and Miha Boltežar. “The norms and variances of the Gabor, Morlet and general harmonic wavelet functions.” In: *Journal of Sound and Vibration* 264.3 (2003), pp. 545–557.

[180] Marcus Soll et al. “Evaluating defensive distillation for defending text processing neural networks against adversarial examples.” In: *International Conference on Artificial Neural Networks*. Springer. 2019, pp. 685–696.

[181] Jakub Špaňhel et al. “Holistic recognition of low quality license plates by CNN using track annotated data.” In: *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. IEEE. 2017, pp. 1–6. ISBN: 978-1-5386-2939-0. DOI: 10.1109/AVSS.2017.8078501.

[182] D Spina, C Valente, and GR Tomlinson. “A new procedure for detecting nonlinearity from transient data using the Gabor transform.” In: *Nonlinear Dynamics* 11.3 (1996), pp. 235–254.

[183] Ashwin Srinivasan, Stephen Muggleton, and Michael Bain. “Distinguishing exceptions from noise in non-monotonic learning.” In: *Proceedings of the 2nd International Workshop on Inductive Logic Programming*. Citeseer. 1992, pp. 97–107.

[184] Fridtjof Stein. “Efficient computation of optical flow using the census transform.” In: *Joint Pattern Recognition Symposium*. Springer. 2004, pp. 79–86.

[185] Thilo Strauss et al. “Ensemble methods as a defense to adversarial perturbations against deep neural networks.” In: *arXiv preprint arXiv:1709.03423* (2017).

[186] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. “One pixel attack for fooling deep neural networks.” In: *IEEE Transactions on Evolutionary Computation* 23.5 (2019), pp. 828–841.

[187] Kashif Sultan, Hazrat Ali, and Zhongshan Zhang. “Big data perspective and challenges in next generation networks.” In: *Future Internet* 10.7 (2018), p. 56.

[188] Christian Szegedy et al. *Intriguing properties of neural networks*. 2013. arXiv: 1312.6199 [cs.CV].

[189] Elham Tabassi et al. *A Taxonomy and Terminology of Adversarial Machine Learning*. 2019.

[190] Thomas Tanay and Lewis Griffin. “A boundary tilting persepective on the phenomenon of adversarial examples.” In: *arXiv preprint arXiv:1608.07690* (2016).

[191] Jian Tang et al. “Enhancing effectiveness of outlier detections for low density patterns.” In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer. 2002, pp. 535–548.

[192] Rohan Taori et al. “Targeted adversarial examples for black box audio systems.” In: *2019 IEEE Security and Privacy Workshops (SPW)*. IEEE. 2019, pp. 15–20.

[193] A Tch. *The mostly complete chart of neural networks, explained*. 2019.

[194] Shixin Tian, Guolei Yang, and Ying Cai. “Detecting adversarial examples through image transformation.” In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[195] Maurras Ulbricht Togbe et al. “Anomaly Detection for Data Streams Based on Isolation Forest Using Scikit-Multiflow.” In: *International Conference on Computational Science and Its Applications*. Springer. 2020, pp. 15–30.

[196] Florian Tramèr et al. “Ensemble adversarial training: Attacks and defenses.” In: *arXiv preprint arXiv:1705.07204* (2017), pp. 1–20.

[197] Florian Tramèr et al. “Ensemble adversarial training: Attacks and defenses.” In: *arXiv preprint arXiv:1705.07204* (2017).

[198] Florian Tramer et al. “On Adaptive Attacks to Adversarial Example Defenses.” In: *arXiv preprint arXiv:2002.08347* (2020).

[199] Jonathan Uesato et al. “Adversarial risk and the dangers of evaluating against weak attacks.” In: *arXiv preprint arXiv:1802.05666* (2018).

[200] Anant J Umbarkar and Pranali D Sheth. “Crossover operators in genetic algorithms: a review.” In: *ICTACT journal on soft computing* 6.1 (2015).

[201] B Vanajakshi, B Sujatha, and K Sri Rama Krishna. “An Analysis of Thinning & Skeletonization for Shape Representation.” In: *International Journal of Computer Communication and Information System (IJCCIS)* 2.1 (2010), pp. 250–253.

[202] Harald Von Boehmer and Pawel Kisielow. “Self-nonself discrimination by T cells.” In: *Science* 248.4961 (1990), pp. 1369–1373.

[203] Frederick M Waltz and John WV Miller. “Efficient algorithm for Gaussian blur using finite-state machines.” In: *Machine Vision Systems for Inspection and Metrology VII*. Vol. 3521. International Society for Optics and Photonics. 1998, pp. 334–341.

[204] Jingyi Wang et al. “Detecting adversarial samples for deep neural networks through mutation testing.” In: *arXiv preprint arXiv:1805.05010* (2018).

[205] Xiaosen Wang, Hao Jin, and Kun He. “Natural language adversarial attacks and defenses in word level.” In: *arXiv preprint arXiv:1909.06723* (2019).

[206] D Randall Wilson and Tony R Martinez. “Reduction techniques for instance-based learning algorithms.” In: *Machine learning* 38.3 (2000), pp. 257–286.

[207] Victor J Wilson and Leslie P Felpel. “Specificity of semicircular canal input to neurons in the pigeon vestibular nuclei.” In: *Journal of neurophysiology* 35.2 (1972), pp. 253–264.

[208] Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. “A Survey on Neural Architecture Search.” In: *arXiv preprint arXiv:1905.01392* (2019).

[209] Rey Wiyatno and Anqi Xu. “Maximal jacobian-based saliency map attack.” In: *arXiv preprint arXiv:1808.07945* (2018).

[210] George Wolberg and Siavash Zokai. “Robust image registration using log-polar transform.” In: *Proceedings 2000 International Conference on Image Processing (Cat. No. 00CH37101)*. Vol. 1. IEEE. 2000, pp. 493–496.

[211] Eric Wong and J Zico Kolter. “Provable defenses against adversarial examples via the convex outer adversarial polytope.” In: *arXiv preprint arXiv:1711.00851* (2017).

[212] Xi Wu et al. “Reinforcing Adversarial Robustness using Model Confidence Induced by Adversarial Training.” In: *International Conference on Machine Learning*. 2018, pp. 5330–5338.

[213] Simon Wun and Andrew Horner. “A comparison between local search and genetic algorithm methods for wavetable matching.” In: *Journal of the Audio Engineering Society* 53.4 (2005), pp. 314–325.

[214] Cihang Xie et al. “Mitigating adversarial effects through randomization.” In: *arXiv preprint arXiv:1711.01991* (2017).

[215] Weilin Xu, David Evans, and Yanjun Qi. “Feature squeezing: Detecting adversarial examples in deep neural networks.” In: *arXiv preprint arXiv:1704.01155* (2017).

[216] Xiaojun Xu et al. “Detecting AI Trojans Using Meta Neural Analysis.” In: *arXiv preprint arXiv:1910.03137* (2019).

[217] David XD Yang and Abbas El Gamal. “Comparative analysis of SNR for image sensors with enhanced dynamic range.” In: *Sensors, cameras, and systems for scientific/industrial applications*. Vol. 3649. International Society for Optics and Photonics. 1999, pp. 197–211.

[218] Tao Yang et al. “GF-NSA: A Negative Selection Algorithm Based on Self Grid File.” In: *Applied Mechanics and Materials* 44-47 (Dec. 2010), pp. 3200–3203. DOI: 10.4028/www.scientific.net/AMM.44-47.3200.

[219] Dong Yin et al. “A fourier perspective on model robustness in computer vision.” In: *Advances in Neural Information Processing Systems*. 2019, pp. 13255–13265.

[220] Xiaoyong Yuan et al. “Adversarial examples: Attacks and defenses for deep learning.” In: *IEEE transactions on neural networks and learning systems* 30.9 (2019), pp. 2805–2824.

[221] Q. Zeng et al. “A Multiversion Programming Inspired Approach to Detecting Audio Adversarial Examples.” In: *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2019, pp. 39–51.

[222] Xiaohui Zeng et al. “Adversarial attacks beyond the image space.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 4302–4311.

[223] Runtian Zhai et al. “Macer: Attack-free and scalable robust training via maximizing certified radius.” In: *arXiv preprint arXiv:2001.02378* (2020).

[224] Chiliang Zhang et al. “Detecting adversarial perturbations with saliency.” In: *2018 IEEE 3rd International Conference on Signal and Image Processing (ICSIP)*. IEEE. 2018, pp. 271–275.

[225] Min-Ling Zhang and Zhi-Hua Zhou. “ML-KNN: A lazy learning approach to multi-label learning.” In: *Pattern recognition* 40.7 (2007), pp. 2038–2048.

[226] Pinlong Zhao et al. “Detecting adversarial examples via key-based network.” In: *arXiv preprint arXiv:1806.00580* (2018).

[227] Qi Zhao and Toyoaki Nishida. “Using qualitative hypotheses to identify inaccurate data.” In: *Journal of Artificial Intelligence Research* 3 (1995), pp. 119–145.

[228] Yue Zhao and Maciej K Hryniewicki. “XGBOD: improving supervised outlier detection with unsupervised representation learning.” In: *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2018, pp. 1–8.

[229] Yue Zhao, Zain Nasrullah, and Zheng Li. “PyOD: A Python Toolbox for Scalable Outlier Detection.” In: *Journal of Machine Learning Research* 20.96 (2019), pp. 1–7. URL: `http://jmlr.org/papers/v20/19-011.html`.

[230] Yue Zhao et al. “LSCP: Locally selective combination in parallel outlier ensembles.” In: *Proceedings of the 2019 SIAM International Conference on Data Mining*. SIAM. 2019, pp. 585–593.

[231] Zhengli Zhao, Dheeru Dua, and Sameer Singh. “Generating natural adversarial examples.” In: *arXiv preprint arXiv:1710.11342* (2017).

[232] Zhaoxiang Yi et al. “A Matrix Negative Selection Algorithm for Anomaly Detection.” In: *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. 2008, pp. 978–983. DOI: 10.1109/CEC.2008.4630915.

[233] Qin Zhiyuan et al. “A robust adaptive image smoothing algorithm.” In: *ALGORITHMS* 1.5 (2005).

[234] G. Zhou et al. *Body Sensor Networks*. Cambridge, MA: MIT Press, 2008.