Security and Robustness of Neural Networks

Master's Thesis

Siddique Reza Khan

khansidd@b-tu.de

Chair of Computer Engineering
BTU Cottbus-Senftenberg

Supervisors:
Prof. Dr. -Ing. habil. Michael Hübner
Dr. Marcelo Brandalero

December 5, 2020

Declaration of Authorship

The author declares that he has written the thesis at hand independently, without outside help and the use of any other but the listed sources. Thoughts taken directly or indirectly from external sources (including electronic sources) are marked accordingly without exception. Sources used verbatim and contentual were quoted according to the recognized rules for scientific work. This thesis has not been submitted in the same or similar form, not even partially, within the scope of a different examination. Thus far, it also has not been publicized yet. The author herewith agrees that the thesis will be examined for plagiarism with the help of a plagiarism-detection service.

Cottbus,December 5, 2020	Siddique Reza Khan
Place, Date	Signature of the author

Acknowledgements

Undertaking this master's thesis has been a truly exciting and learning experience for me. It would not have been possible to do without all-mighty Allah's help and the support and guidance that I received from many people.

Firstly, I would like to express my sincere gratitude to my advisor Prof. Dr.-Ing. habil. Michael Hübner for his continuous support of my master's study and related research, for his patience, motivation, and immense knowledge. I would like to express my special gratitude to Dr. Marcelo Brandalero for his continuous support of my project. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my master's study. Furthermore, I would like to thank Dr. Sadegh Sadeghipour, Dr.-Ing. Tobias Koal, Mr. Hubertus von Manstein, and Mr. Hans-Werner Wiesbrock for their valuable help.

Nothing, though, would be possible without the support and belief of my family and friends. Thank you for always being there and especially when needed the most. I would like to give an especial thank to my best companion, my father (late Liaquat Reza Khan), my mother (Suraya Begum), my younger brother (Dr. Sadeque Reza Khan), my wife (Saraban Tohura), my daughter (Samaira Siddika Khan), and my classmates (Shaik Fathima and Sam Abubakar Tareque) for their continuous motivation.

The author of this master's thesis acknowledges the financial support of the BTU Cottbus-Senftenberg, IHP GmbH, and Philotech GmbH through the student assistantship position. Funding from the ITPower Solutions GmbH and BAB GmbH as student assistantship is also gratefully acknowledged.

Abstract

In the modern era, machine learning algorithms are using in several areas to process big data and information. However, the deep learning algorithm comes with security vulnerabilities in itself. Further, it is still an emerging research area to find a proper countermeasure against the exploitation of neural networks. Though Tesla has made progress enormously in a self-driving car, afterward, a research team of Tencent Keen Security Lab has detected security vulnerabilities in the Tesla system design of self-driving cars. In this project, it is built a robust neural network algorithm to defend an adversary for real-time self-driving cars to identify accurately road traffic signs and then executed white-box and black-box attack methods on the same algorithm using FGSM for untargeted and I-FGSM for targeted misclassification in output label. Then it is analyzed and evaluated with several testing how the model behaves against the attack and put an argument to establish a well-defined security assessment standard on machine learning model for decision systems in the testing phase of ML for evasion attack.

Contents

Abstract iii

List of Abbreviations vi

List of Figures x

List of Tables xiii

1 Introduction 1

1.1 Problem description . . . 2
1.2 Goal . . . 2
1.3 Framework . . . 3

2 Theory and Background 4

2.1 Convolutional Neural Network (CNN) . . . 4
2.2 Training Convolution Neural Network . . . 21
2.3 Hyperparameter . . . 23
2.4 AlexNet Model . . . 26
2.4.1 Calculate the Tensor Size at Each Stage . . . 28
2.4.2 Calculation of Parameters in the Neural Network (NN) . . . 29

3 State of the Art 32

3.1 Adversarial Machine Learning . . . 32
3.2 Literature study . . . 34
3.3 Adversarial Threat Model and Taxonomy . . . 38
3.4 Adversarial Examples . . . 45
3.4.1 Fast gradient sign method (FGSM) . . . 47
3.4.2 Iterative Fast Gradient Sign Method (I-FGSM) . . . 48
3.5 Defensive Techniques . . . 49

4 Methodology **51**
4.1 Collection and dataset Preparation 53
4.2 GTSAlexNet Model Architecture 56
4.3 Adversarial Attacks and Their Implementations 59
4.3.1 White Box Attack and FGSM Technique 60
4.3.2 Black Box Attack and I-FGSM Technique 61

5 Results and Evaluations **64**
5.1 Dataset Distribution Structure 64
5.2 Training and Evaluating the Model 65
5.3 Evaluation of Model Prediction 67
5.4 Metadata Analysis Between Models Architecture 68
5.5 CNN Algorithm Comparison 69
5.6 White-box and Black-box Attack 71
5.7 Influence of Epsilon on Images 74
5.8 Adversarial Model Influences Accuracy 75
5.9 Measures of Image Quality: SSIM vs MSE 77
5.10 Defensive Techniques for GTSAlexNet 79

6 Conclusion **81**

Bibliography **83**

A GTSRB Assigned Class Label **A-1**

B Attack Vectors Analysis Datasets **B-1**

List of Abbreviations

2D	Two Dimensional
3D	Three Dimensional
ADAF	Advance Development and Analysis Framework
Adam	Adaptive Moment Estimation
API	Application Program Interfaces
ASP	Accuracy Security Product
AUC	Area Under The Curve
BCE	Binary Crossentropy
BN	Batch Normalization
CCE	Categorical Crossentropy
CIA	Confidentiality, Integrity, and Availability
CL	Convolutional Layer
CNN	Convolutional Neural Network
conv	Convolution Layer
CPU	Central Processing Unit
CSV	Comma Separated Values
DDoS	Distributed Denial of Service
DL	Deep Learning
DNN	Deep Neural Network
DoS	Denial of Service
FC	Fully Connected
FGSM	Fast Gradient Sign Method
GD	Gradient Descent

GPU	Graphics processing unit
GTSRB	German Traffic Sign Recognition Benchmark
I-FGSM	Iterative Fast Gradient Sign Method
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
IP	Internet Protocol Address
JSMA	Jacobian-based Saliency Maps Attacks
KB	Kilobyte
L-BFGS	Limited-memory Broyden– Fletcher–Goldfarb–Shanno
LRN	Local Response Normalization
M	Million
MAC	Multiplier and Accumulate Unit
MAE	Mean Absolute Error
MB	Megabyte
MD5	Message Digest (version 5)
MITM	Man in the Middle
ML	Machine Learning
MPL	Multilayer Perceptron
MSE	Mean Squared Error
NN	Neural Network
PCA	Principal Component Analysis
PE	Processing Engine
PPM	Portable Pixmap, P6
RAM	Random Access Memory
ReLU	Rectified Linear Unit
RGB	Red Green and Blue
RMSE	Root Mean Square Error
ROC	Receiver Operating Characteristic Curve

RONI	Reject on Negative Impact
RSM	Random Subspace Method
RTL	Register Transfer Level
SCCE	Sparse Categorical Crossentropy
SGD	Stochastic Gradient Descent
SHA1	Secure Hash Algorithm (version 1)
SVM	Support Vector Machine
TEP	Timing Error Propagation
TPU	Tensor Processing Unit

List of Algorithms

1 Fast gradient sign method(FGSM) 48
2 Iterative fast gradient sign method(I-FGSM) 49
3 Fast gradient sign method(FGSM) technique for white-box attack 61
4 Iterative fast gradient sign method(I-FGSM) technique for black-box attack 63

List of Figures

2.1 Basic Layout of Machine Learning system. 5
2.2 A basic CNN architecture. 6
2.3 Inside operations of a convolutional neural network. 6
2.4 Image matrix of three dimensions. 8
2.5 Deep learning dataset features of Scalars, Vectors, Matrices, and Tensors 9
2.6 RGB images as stack of 3-tensor. 10
2.7 Image dataset: 4-tensor. 11
2.8 (a) Convolutional layers with feature maps and (b, c) image matrix multiplies with kernel or filter matrix. 12
2.9 Stride of 2 pixels with filter size 3×3. 14
2.10 Convolutional process example with padding = "valid" or "same" and strides = 1. 15
2.11 Max pooling layer (kernel 2×2, stride 2, and no padding). 16
2.12 ReLU activation function. 17
2.13 Example of the Softmax activation function. 18
2.14 Fully-Connected Neural Network Layer. 20
2.15 Training the CNN with forward propagation and backpropagation. 22
2.16 An illustration of the architecture of AlexNet CNN architecture Model. 27

3.1 Road traffic sign with putting stickers. 33
3.2 Adversarial Image generated by L-BFGS Method and FSGM. In FSGM experiment, the value for "ϵ" is 0.007 to make it imperceptible. 33
3.3 A workflow representation of the reactive (left) and proactive (right) defensive mechanisms. 34
3.4 Attacks and Challenges for Training and Inference. 36

3.5 Security threat/attack model on CNN-based system during training, runtime and hardware implementations. 40
3.6 CIA triad, looks simple but actually complex. 41
3.7 Taxonomy of adversarial security threat/attack towards machine learning model. 42
3.8 The adversary adds small perturbations (noises) to the original image "panda," which results in the ML model labeling this image as a "gibbon," with high confidence. Here error parameter ϵ of .007. 46
3.9 The basic methodology of adversarial example generation. 46

4.1 Brief overview of the project's Methodology. 52
4.2 The annotations were created with ADAF software tools by Nisys GmbH 54
4.3 Representation of the GTSRB pre-processed 43 traffic sign classes dataset. 54
4.4 Datasets splitting Output into three categories. 55
4.5 An illustration GTSAlexNetarchitecture presenting its five convolution and three fully connected layers. 56
4.6 Security threat model for white box attack with FGSM technique on GTSAlexNet. 60
4.7 Security threat model for black box attack with I-FGSM technique on GTSAlexNet. 62

5.1 A histogram representation of the training, validation and testing dataset. 65
5.2 Accuracy and loss representation for GTSAlexNet(CNN) model evaluation. 66
5.3 Correctly predicted image with model. 68
5.4 CNN algorithm results overview of the GTSRB. 70
5.5 Evaluation of white box attack with FGSM method for *Stop* image against *different epsilon value* range from 0.0 to 0.40. 72
5.6 Evaluation of black box attack with I-FGSM method for *Stop* image against *epsilon* = 0.034 value misclassified as *Speed limit (120km/h)*. 72
5.7 Execution time for inference, white-box(untargeted) and black-box(targeted) attack. 74

5.8 Measure epsilon with respect to cumulative image misclassification using white-box(FGSM) attack technique. 75
5.9 The value of epsilon is enough to lead to misclassification. 76
5.10 For a human eye can notice something is wrong in the image. . . . 76
5.11 Image quality estimates with SSIM and MSE for FGSM attack. . 77
5.12 Image quality measurement with SSIM and MSE of I-FGSM attack. 78
5.13 Measure Image quality with SSIM to identify noise by human eye. 79

B.1 Dataset for FGSM Attack Analysis. B-1
B.2 Dataset for Image Quality Analysis for FGSM method with SSIM and MSE. B-1
B.3 Dataset for I-FGSM Attack Analysis. B-2
B.4 Dataset for Image Quality Analysis for I-FGSM method with SSIM and MSE. B-2

List of Tables

2.1 Optimizer Comparison 25
2.2 AlexNet CNN - Architecture Details 28
2.3 Comparison of the two normalization technique 28

3.1 A summary of existing attacks for the AlexNet CNN model 35
3.2 Comparative analysis concerning different attacking technique . . 42

4.1 GTSAlexNet CNN - Architecture Details 58
4.2 Summarizes the typical architecture of GTSAlexNet model with hyperparameters 59

5.1 Overview of of GTSAlexNet Model Prediction with GTSRB 2019 dataset. 67
5.2 Relative summary metadata analysis of AlexNet and GTSAlexNet. 69
5.3 Parameter and results overview of the GTSRB. 70
5.4 Memory requires of CNN for used parameters. 71
5.5 Statistical inference of execution time(s) for inference, targeted and untargeted attack 73
5.6 Categorization of attack based on the threat model of CIA attributes 74

A.1 GTSRB Class Label used in the dataset A-1

Chapter 1

Introduction

Nowadays, machine learning (ML) is growing popular for research because of its various application areas such as big data analysis, pattern recognition, image identification, computer vision, clustering analysis, network intrusion detection, autonomous driving, and many more [1, 2, 3]. Machine learning is also used in information security and privacy issues such as malware detection, voice recognition, and face detection by analyzing the corresponding data. Different deep learning (DL) algorithms are being used to predict accurately than ever before to build a robust model [4, 5]. Currently, to improve the accuracy, many researchers are focusing on developing Deep Neural Network (DNN) which is the state-of-the-art of ML. Referring to the robustness of an ML model is divided into two main characteristics. The first characteristic is to adapt against security vulnerabilities. The second feature is quick resilience against reliability threats [2] of ML models such as spam email detection, network intrusion detection, malware analysis, voice recognition, and face detection [6, 7, 8]. If the ML model is not secure and reliable, then an adversary can exploit the ML algorithm easily. Further, it is possible to misclassify the output with both targeted and non-targeted labels for an attacker.

In particular, the ML algorithm focuses on accurate prediction or classification. Further, the ML model requires a minimum amount of CPU computational cost with high efficiency, which implies that a newly implemented model will perform well real-life intelligent systems deployment. In the future, technology and society will fundamentally depend on intelligent automation in everyday life. Hence, the new ML algorithm design and development is accelerating day by day. However, this involves a handful of vulnerabilities, threats, and attacks on the ML algorithm itself. For example, some adversaries can impersonate safety-critical applications by exploiting the vulnerabilities of autonomous driving [1, 9], smart healthcare [10], face recognition systems [3], the sensitive privacy data processing [11] and voice control system [12]. Now, the autonomous vehicles can make wrong decisions on recognizing traffic signs with seizing control. Hence, this research is devoted to reviewing challenges and opportunities to learn about the ML model attacks by the adversaries. Moreover, due to robustness constraints, state-of-the-art

technology is exploring risk and exploiting vulnerabilities.

1.1 Problem description

In 2004, Dalvi et al. [13] developed adversarial learning techniques to defeat the classifier algorithm and analyzed the evasion problem between a classifier and an adversary for producing false negatives. After that, Lowd and Meek [14] introduced the reverse engineering concept of adversarial learning in 2005 and proposed efficient algorithms to construct adversarial attacks against a classifier.

Nonetheless, many research studies are going on to defend against security threats of ML algorithms and models. To address defensive mechanisms regarding security and privacy towards the ML model consists of countermeasures, both the training and testing phases. Furthermore, researchers focused on advanced security threats and defensive techniques against existing ML algorithms with intelligent arguments in the past decades.

1.2 Goal

This study focuses on learning about untargeted and targeted attacks, how these attacks work, how much computational cost is required for specific attacks, and how to build a robust DNN to defend against these attacks.

The primary goal of this thesis is to:

- Research how a neural network's accuracy can degrade quickly after adding perturbation noise into the image.
- Detect which minimum standard noise(error parameter) and modifications need to be deployed in the image that can be deceived by a CNN classifier by an adversary.
- Analyze behavior of neural network model after gradually increasing perturbation
- Evaluate the performance and computational complexity of untargeted and targeted attacks.

For achieving the goal of this thesis, a convolutional neural network (CNN) architectures (example: AlexNet) is built to classify German traffic sign recognition benchmark (GTSRB) to perform ML classification. Further, it is used two categories of datasets: GTSRB 2017 [15] for training, validation, & testing; and GTSRB 2019 [16] for analyzing and evaluation of testing to generalize a generated model. Scripts are then written to generate specific noise for a targeted attack

and random noise for untargeted attacks for the adversarial machine learning model. Finally, it is implemented fast gradient sign method (FGSM) and iterative fast gradient sign method (I-FGSM) for untargeted (white-box) and targeted (black-box) attacks, respectively.

Delimitation

In this project, a CNN machine learning algorithm implemented to classify traffic signs is called AlexNet. No other classic CNN, such as LeNet, GoogLeNet, is deployed. The GTSRB 2017 dataset [15] is used to train the ML model. At the same time, attack techniques are tested with the GTSRB 2019 dataset [16]. Again, attack techniques are also limited within FGSM and I-FGSM methods. Due to resource constraints, the project is installed and fully supported by the Google Colab cloud system.

1.3 Framework

The structure of this thesis is organized as follows. Chapter 2 briefly introduces the basics of machine learning and discusses the inner architecture of the AlexNet model. Chapter 3 represents related work, several challenges in cybersecurity, the definition of adversarial examples, designs adversarial threat model, security threat taxonomy, and various defensive techniques. Chapter 4 gives a detailed description of the implementation ML model followed by corresponding two security issues: a white-box attack with FGSM and a black-box attack with I-FGSM. Chapter 5 summarizes and evaluates attack vectors to assess the time complexity of generating the attacks. Finally, Chapter 6 concludes this paper with remarks.

Chapter 2

Theory and Background

This chapter describes the basic theory of the convolutional neural network(CNN) and AlexNet model. The fundamental architecture of the CNN model is briefly discussed in the CNN section. Further, it is explained various components used in CNN and a comparison between hyperparameters and the necessity to know which parameters are required to fine-tune to produce a higher accuracy for CNN. It is also shown all equations for calculating parameters with examples to build an optimized neural network.

Furthermore, the AlexNet section discusses the inner architecture of the model with a brief illustration concerning the number of parameters that are used in the AlexNet model. Finally, it is discussed all equations for calculating tensor's size at each stage as well as parameters (such as weights) in the neural network.

2.1 Convolutional Neural Network (CNN)

Convolutional neural network (CNN) is a special type of DNN that is developed in the 1980s [17] from the visual cortex of the human brain. Therefore, CNN is considered as an emulation of human sensory. Since then, it has been used for image recognition. At present, CNN is actively used in some complex tasks with high probable accuracies such as self-driving cars, face recognition, and many more applications. CNN models need high computational power, and these requirements are increased day by day [18]. However, beyond the visual perception application CNN can be used for several other tasks such as natural language processing, voice recognition, malware detection, pattern recognition, and other fields as well [19, 20, 21]. Figure 2.1 shows a basic block diagram of the ML system, and each of these parts will be explained in the next sections. Since CNN is a subdivision of the ML system, so it has the best ability to generate features and classify [22] subsequently.

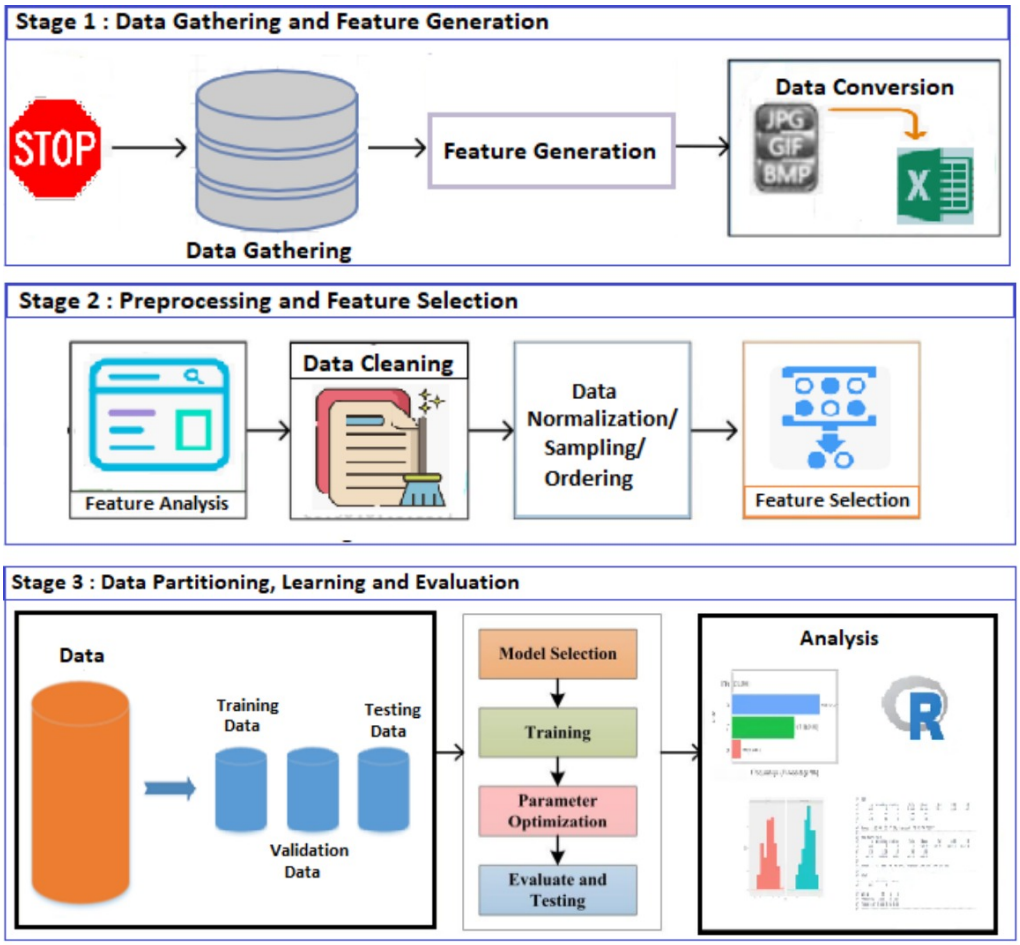


Figure 2.1: Basic Layout of Machine Learning system. (Image based on: [22])

CNN Architecture

This section presents the basics of CNN architectures, components, building blocks, and implementation.

Figure 2.2 represents a typical building block architecture of CNN. It demonstrates inputs, outputs, some convolutional layers, pooling layers, activation functions such as sigmoid, and fully connected layers. CNN can optimize by tuning the hyperparameters. Moreover, CNN has the capability to incorporate batch normalization and dropout regularization units as well. The best performance for CNN architecture can be achieved through proper design and arrangement of a model's components [22]. In the following sections, the role of these components is briefly discussed for a CNN architecture.

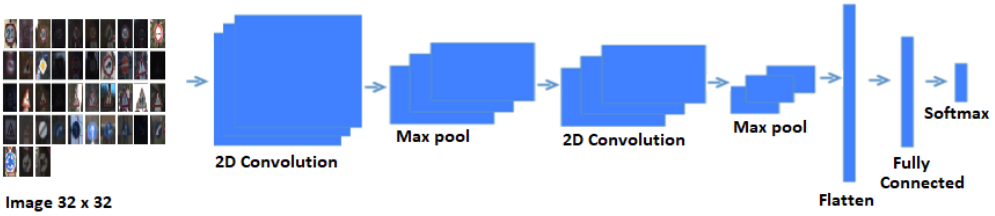


Figure 2.2: A basic CNN architecture.

Basic Neural Network Components

For this research, to optimize and subsequently build a deep learning model with following components and equations are necessary to be studied.

Equation (2.1) [17] is the basic function of a artificial neural network.

$$y = \sum_{i=1}^{N} (w_i \times x_i) + b \tag{2.1}$$

where, x represents the input features vector matrix, w represents the weight matrix per input features except for the bias, b represents the bias weight connected between the bias and the neuron.

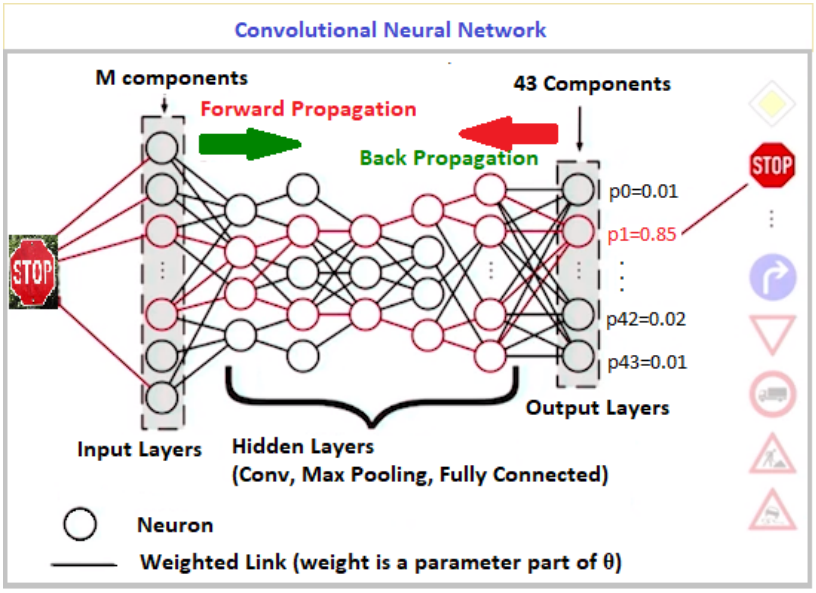


Figure 2.3: Inside operations of a convolutional neural network.(Image based on: [23])

a) Neuron (Node) — *Neuron* is the basic unit of the CNN, that takes input features associated with weights and bias [17] from the previous layer to compute the output, as shown in Figure 2.3 . It obtains the output by multiplying a weight value with a certain number of inputs, and adding a bias value as presented in Equation (2.1) . Initially, the weight parameters of the neuron will be adjusted during training time. For example, if the input features are 4 and there are 4 associated weights, then the Equation (2.2) provides as follows:

$$y = w_1 \times x_1 + w_2 \times x_2 + w_3 \times x_3 + w_4 \times x_4 + b \times 1 \tag{2.2}$$

Figure 2.1 shows in 3^{rd} stage CNN training steps. The main goal of these steps is to update these weight values to decrease the loss(error). In particular, the weight parameters of the neuron will adjust during training time.

b) Bias Neuron (Offset) — It is an extra input with output always 1 to neurons [17], and it has its connection weight represented as b in Equation (2.2) The purpose of the bias is to make sure that the neuron is activated even when all the inputs are zeros.

c) Connections — Figure 2.3 shows the black or red lines/connections, which connect one neuron in one layer to another neuron in the next layer. These connections always associate a weight value with it.

Basic Neural Network Layout

The basic neural network can be organized with the following layout.

1. Input Layer —In Figure 2.3, this is the first layer in the CNN without any associated weights and biases values [17]. It takes input images and passes them on to the next layer. In particular, it does not apply any mathematical operations on the input images.

2. Hidden Layers — Hidden layers consist of neurons(nodes) [17]. Figure 2.3 shows the neurons stack vertically in one hidden layer that represents by circles(e.g., black, red). Again, Figure 2.2 shows a fully connected layer, which means all neurons in a hidden layer are connected to each and every neuron in the next layer. The difference with a convolutional layer, only a specific area (called a receptive field) passes input to the neurons of the next layer [24]. It applies mathematical operations and represents the input data with different transformations. The last hidden layer carries on final output values before prediction.

3. Output Layer — In Figure 2.3, it is the last layer in the neural network [17]. In particular, it receives input from the last hidden layer. With this layer, CNN will classify the number of desired classes and predict a class with the highest accuracy.

4. Input Shape — It is the shape of the input image, which yields to the input layer and subsequently hidden layers [25]. Figure 2.2 shows the network fed input image to the input layer has input shape 32×32.

5. Weights (Parameters) — Weights are also known as a parameter. These vectors will decide the influence the input data have on the output. In particular, weights associated with input represent the strength of the connection between two connected nodes [25]. The weight affects the importance of an input image. For example, neuron 1 has greater influence over neuron 2 if node 1 has greater weight than node 2 in magnitude. It can be summarized that input associated with near-zero weights will not change the output [25]. In contrast, the input associated with negative weight will increase, which means that the probable output class will decrease and vice versa.

Image Data

In this project, the image data is usually made up of three-dimensions ($3D$). These three dimensions are as follows:

- Image height
- Image width
- Color channels

The critical item in the above is color channels. For the colored images matrix, the number of color channels represents the depth of three [26]. That is, one color for each of the three *channels*, and the colors are made up of RGB (red, green, blue) pixels [26]. So, the colors define by three numeric values in the range of 0 - 255 for each pixel, as shown in Figure 2.4.

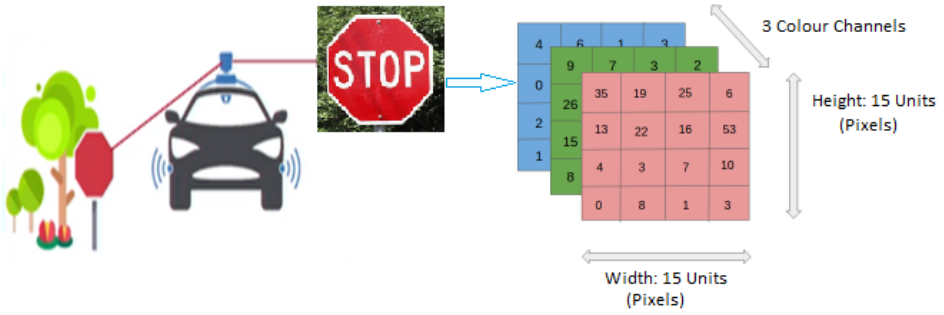
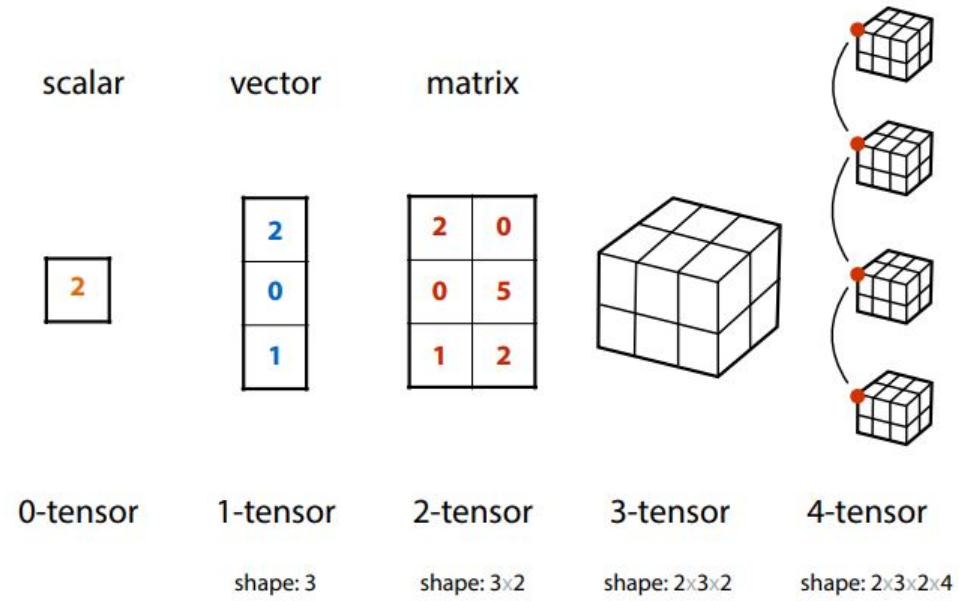


Figure 2.4: Image matrix of three dimensions. (Image based on: [23])

Tensors

Goodfellow et al. defined, a tensor is an n-dimensional array of numbers arranged on a regular grid, where n is variable number of axes and $n > 2$ [27], as shown in Figure 2.5(a,b). However, a tensor can be cast straightforward generalization of vectors and matrices to higher dimensionalities [28] to deal with any kind of dataset, as shown in Figure 2.5(b)

(a) Difference between a scalar, a vector, a matrix and a tensor.(Source of Image: [29])



(b) Deep learning systems: tensors.(Source of Image: [28])

Figure 2.5: Deep learning dataset features of Scalars, Vectors, Matrices, and Tensors

Here is two common data types that can be stored in tensors [30]:

- $3D$ **tensors** is consist of width, height, color depth, e.g. time series dataset
- $4D$ **tensors** is consist of sample size, width, height, color depth, e.g. images dataset, where, *sample size* is the number of things in the dataset which is one of the common thread of every $4D$ *tensors*.

$3D$ Tensors

In an RGB image, images can be represented as $3D$ tensors or 3-tensors where the color of a single-pixel can be represented using three values(how red it is, how green it is, and how blue it is) between 0 and 1, as shown in Figure 2.6. Moreover, $3D$ tensor (an RGB image) can be thought of as a stack of three color channels, this means that it can be represented by matrices.

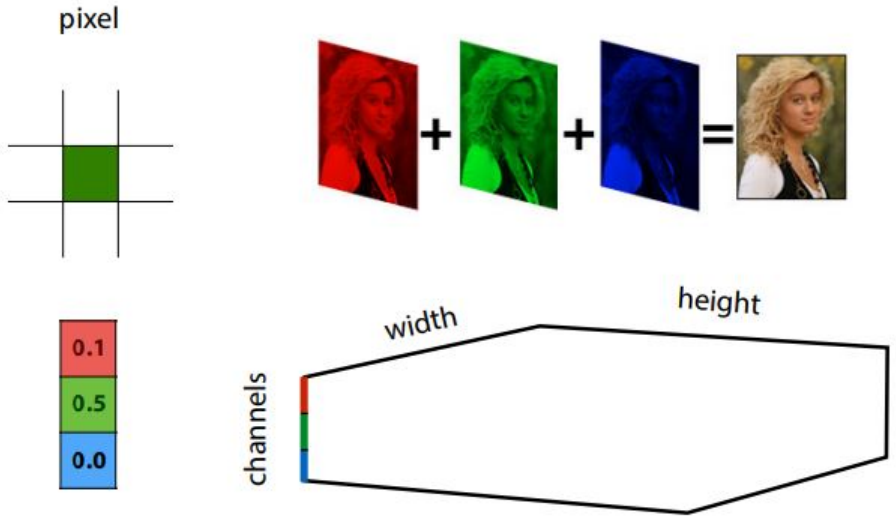


Figure 2.6: RGB images as stack of 3-tensor.(Source of Image: [28, 31])

$4D$ Tensors

For this project, it is used a dataset called GTSRB of multiple images, which is a $4D$ tensor or 4-tensor, as shown in Figure 2.7. This means that there is one extra, *indexing(sample size)* for the different images in the dataset and it is already discussed above of original image dimensions. For Example, in this project, it is loaded with the GTSRB 2017 dataset, where The training data contains $34,799$ images (details discussion is in section 4.1), each with 32 by 32 resolution and 3 color channels.

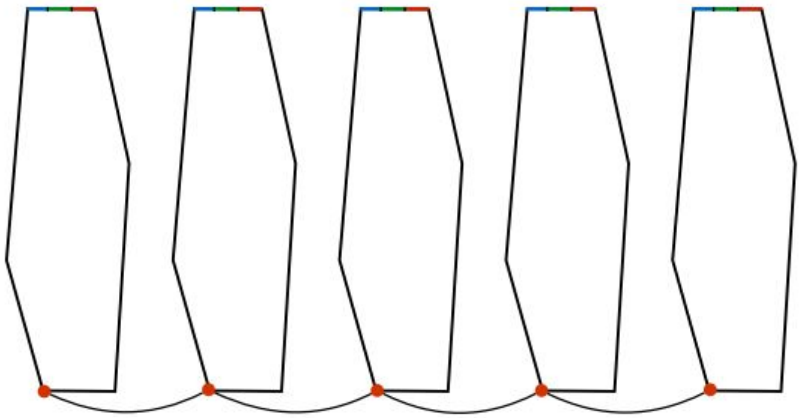


Figure 2.7: Image dataset: 4-tensor.(Source of Image: [28])

Convolutional Layer (CL)

The convolutional layer (CL) is the two-dimensional ($2D$) layer in CNN [17], which is composed of kernels or filters to extract features from an input image [22]. It divides the image into small slices and looks at specific parts of the image. Each small part is commonly known as *receptive fields*, as shown in Figure 2.8(a) with the highlighted parts.

Convolutional *kernel or filters* is in $f_h \times f_w$ pattern of pixels which works by dividing an image [17]. In practice, *filters* are defined as a number of *channels* in the output of a CL whereas convolution *kernel* size of the filters is represented in two dimensions (mostly $2D$ square size) in CL, as shown in Figure 2.8(b,c). In addition, the majority of the researchers are using the kernel(filter) size of odd numbers that are within $\{1, 3, 5, 7, or 11\}$ for getting the best possible results without distortion of an image. Furthermore, *feature maps* are also known as *channels*, the term simply stands for a $3D$ tensor with two spatial axes (width and height) and one depth axis [26]. Convolution uses the cross-correlation between input image features and kernel using small squares of input data. The mathematical operation takes two inputs. One is an input image matrix, and another one is a filter or kernel.

In Figure 2.8(b) shows the convolution operation [32] can be expressed as follows:

- Convolution takes two inputs: An image matrix (volume) of dimension $(i \times j \times k)$ and a filter $(f_h \times f_w \times d)$ and
- Outputs matrix will be a volume of dimension $(i - f_h + 1) \times (j - f_w + 1)$.

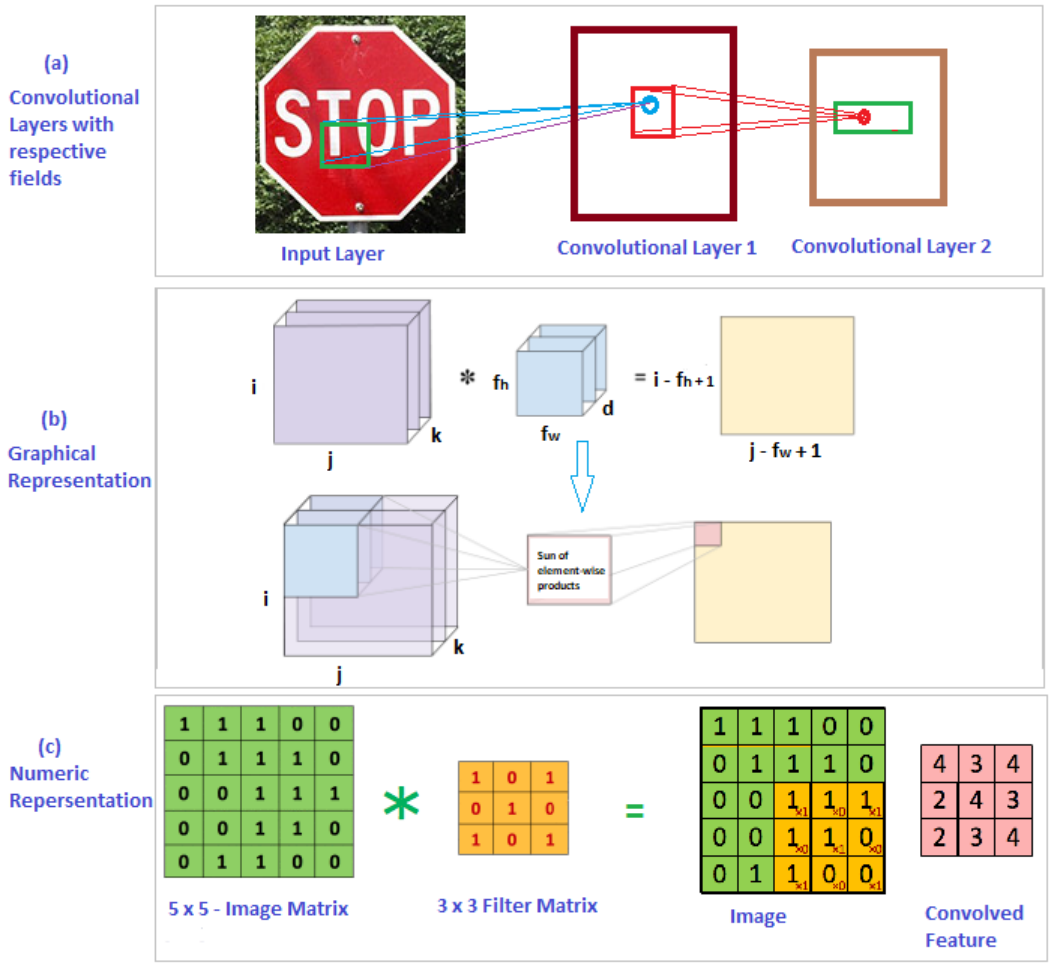


Figure 2.8: (a) Convolutional layers with feature maps and (b, c) image matrix multiplies with kernel or filter matrix. (Image based on: [33])

Figure 2.8(c) shows the matrix multiplication [33] operation involves both the input image and filter (kernel) refers to the sum of element-wise product [32] operation in the field of computer vision for the convolution layer. The computation of the last one output is as follows:

1. The filter places on the - most down-right corner of the input image
2. To calculate the element-wise product of corresponding each image pixel value and the filter value.
3. To obtain the output value of the most down-right pixel, the final operation is summing the products of the resultant matrix.

Furthermore, to obtain the next output pixel value, first, the filter will move to the right by one position and repeat steps 2 and 3. Keep iterating this until the original input image will fully be covered.

Sometimes, the input image dimension does not fully fit by the filter [17]. Nonetheless, this problem overcomes in a way, is called *padding*, which is briefly discussed in the next section. Again, another important concept is to define *stride* size, which controls every move of filters around the pixel of the image to allocate the spatial dimensions (height and width).

As shown in Figure 2.8(b), Equation (2.3) determines the spatial size of output image for an input image that can be computed [24] as a function $f(.)$ for calculating how many neurons "fit" in output with receptive parameters is given by

$$\begin{aligned} o &= f(n, p, k, s) \\ o &= \frac{n + 2p - k}{s} + 1 \end{aligned} \tag{2.3}$$

where, o refers to size (width/height) of output image, n refers to size (width/height) of input image, k refers to size (width/height) of kernels (filters) is used in the CL, s refers stride of the convolution operation, p refers padding (the amount of zero padding).

Again, different types of *filters (f)* are available such as edge detection, blur, and sharpen, which can perform operations on the convolution of an image by applying these filters [33]. Moreover, a convolutional layer consists of multiple filters. And the output of a filter is one *feature map*, which represents each pattern for the convolutional layer [26]. For example, if there are 9 different patterns/filters, the output feature map will have a depth of 9 as well.

Feature map size or map size(M_F) is defined by the product of the output height(h) and width(w) [34]. Here, output size ($M_F = h \times w$) is calculated through Equation (2.3). Therefore, the total number of *neurons* in the output size of a CL is $M_F \times f$. For example, an input image size is a 32-by-32-by-3 RGB image. For a CL with 9 filters and a kernel size of 5-by-5. If the *stride* is 1 in each direction and *padding* size is specified of "valid" (discussed in the next section), then each output feature map is 28-by-28. Using Equation (2.3), it can be calculated for $(32 - 5 + 2 \times 0)/1 + 1 = 28$, and the outermost will be automatically discarded due to applying of "valid" padding to the image data. Now, the total number of neurons in a CL layer is $28 \times 28 \times 9 = 7056$.

Strides

Stride size means for every move of the filter represents by counting the number of pixels in rows/columns [26]. For example, if the stride size is 1, then the filters move to 1 pixel at a time. For the convolution layer, the stride is used for fine-tuning the hyperparameters and reducing the size of the image. In particular, it is also possible to represent a bigger image by taking the closest sample of every pixel and obtaining quite a similar image information with a much smaller image size [32].

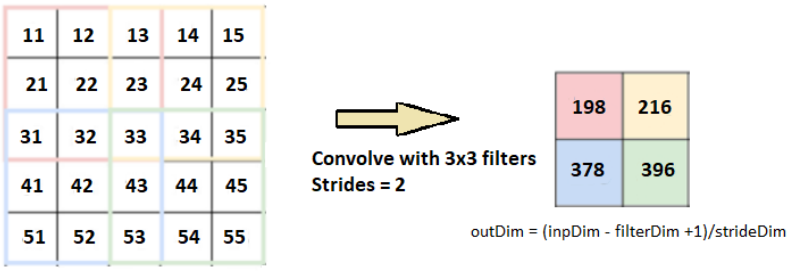


Figure 2.9: Stride of 2 pixels with filter size 3 × 3. (Image based on: [33])

Figure 2.9 shows the image size has changed by applying the stride size of 2 pixels. Therefore, the 5 × 5 image results in an output of 2 × 2 image with a stride of 2.

Padding

Padding is a necessary parameter to fit the filter into the image [33]. Generally, padding implies in two ways: one is "valid" padding, and another one is "same" padding [35], as shown in Figure 2.10.

- In the upper part of Figure 2.10 shows, if the parameter for CL is set to *valid*, then it is called valid padding. This option represents the valid portion of the image and drops the part where the filter does not perfectly fit the image. Therefore, CL discards the rows/columns from the bottom and right corner of the image for not using the zero-padding [17].
- In the bottom part of Figure 2.10 shows that, if the parameter for CL is set to *same*, then it is called zero-padding. Since the input image size shrinks and finally becomes too small after passing through many CL without padding. It results in an image of no use for accurate classification [32]. Therefore, the zero-padding is solving the problem of the reduction of the dimension of the image with a CL. Besides, the solution of zero-padding

with strides = 1 will be the output of the same spatial dimensions (width and height) [17] of the input image.

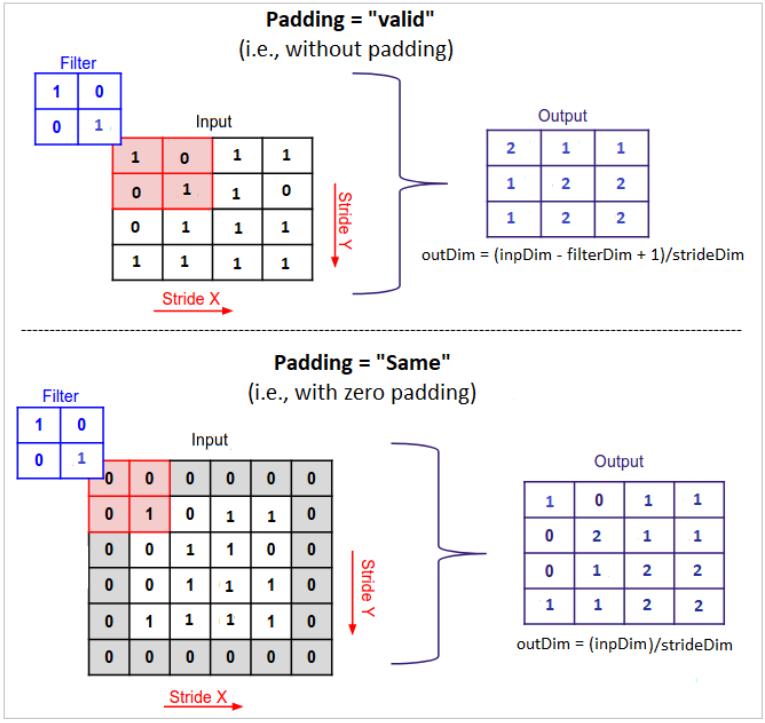


Figure 2.10: Convolutional process example with padding = "valid" or "same" and strides = 1. (Image based on: [35])

Pooling layer

The goal of the pooling layers is to decrease the number of parameters by sub-sampling too large images. Downsampling image reduces the risk of over-fitting but retains important information. Moreover, it helps to limit the memory usages and computation load [17]. There are different types of spatial pooling, such as max pooling, average pooling, L2 pooling, and overlapping pooling, which can be used in CNN [22].

Figure 2.11 shows the most common type of pooling layer, which is called the max-pooling layer with a kernel size of 2×2, the stride of 2, and no padding [17]. The operation of the max-pooling layer is to select the most significant value of the input image or from the modified feature map of the previous CL.

In this example, an input image of 4×4 size is downsampled using a kernel of size 2×2 and a stride of 2. Then, choosing the largest pixel value is done in

the max-pooling layer. The feature map will reduce the size by a factor of two and return a reduced matrix, that is $2x$ smaller [26].

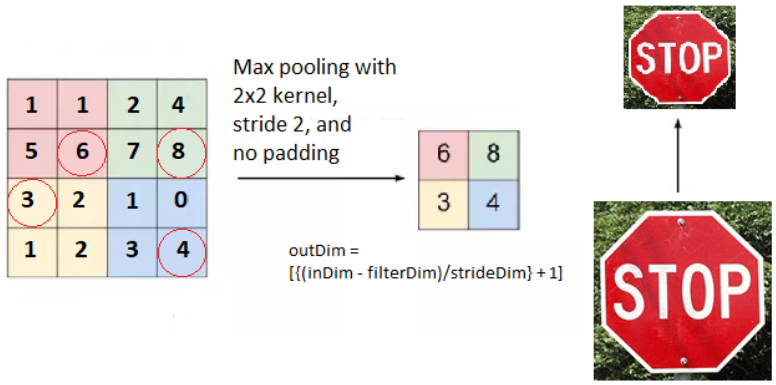


Figure 2.11: Max pooling layer (kernel 2×2, stride 2, and no padding). (Image based on: [33])

The researchers of computer vision generally prefer to reduce the size of the image again by max-pooling layer due to its ability to produce better results preserving the original image information with a smaller image [32].

Activation Function

Activation function introduces non-linearity via the linear CNN input by using a non-linear function [17] such as *sigmoid, Tanh, ReLU, and Softmax*. First, it is used mainly to reduce the values into a smaller range; for example, the *tanh* activation function values exist between a *range from* -1 *to* 1. Second, as a neural network system acts and simulates to work like the human brain, so the activation functions help to model complex relationships between an input image and an output prediction. In summary, CNN solves an accurate prediction with the help of a complex math equation.

In this project, there are two main activation functions are being used, which are explained below.

1. Rectified Linear Unit(ReLU) Activation Function

The Rectified Linear Unit(ReLU) function is used in default, for all the convolutional neural networks as it is faster [17] than other functions such as *sigmoid, tanh*, etc. The function and its derivative both are not distinguishable *at* $x = 0$

or for $x < 0$. Thus, it is called monotonic function [36]. Figure 2.12 shows the output value of the function $f(x)$ is zero when x is less than zero. Otherwise, the function $f(x)$ is equal to x when x is above or equal to zero, as presented in Figure 2.12. In particular, the output range of the function is from 0 to infinity.

The equation (2.4) for ReLU activation function can be written as :

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{2.4}$$

The ReLU activation function turns all the negative values to zero, which means any negative input gives to the resulting image does not affect the output matrix appropriately [36].

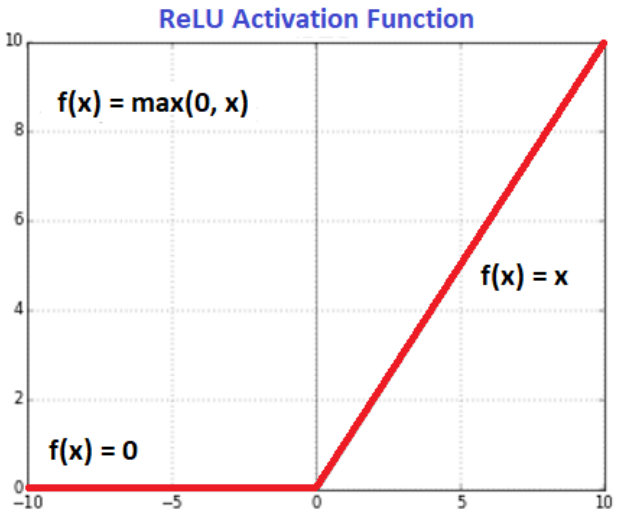


Figure 2.12: ReLU activation function.

The following are the advantages of ReLU over linear function [37].

- It is very quick to evaluate.
- The computational cost is less.
- ReLU fires a less number of neurons, and successive updates are fairly effective.
- Most importantly, the ReLU leads to gradient descent is high when the neuron activates without saturation of output value.

2. Softmax Activation Function

The softmax activation function is a special kind of function that refers to each element in the output depends on the entire set of elements of the input. It can express with the exponential of the element at position j, divided by the sum of the exponential of all elements of the vectors. So it considers the information from the whole set of input vectors. The output value of the softmax function is in the range between 0 and 1. Moreover, the softmax function converts input values into a categorical probability distribution [38]. In particular, it transforms the large/small input value of previous layers and fits them into a valid probability distribution. Therefore, the total sum of each output is equal to 1.

The Equation (2.5) [38] for softmax function is written as:

$$f(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_j}}, \text{for } j = 1, \ldots, K \tag{2.5}$$

Figure 2.13 shows the example of the softmax function. To calculate the denominator with the softmax function of $f(x)$, first, each input vector will apply to the given exponential to the power of elements. The results produce $[2.72, 7.39, 20.08, 54.60, 148.41, 403.43]$. Then to add all resultant values is equal to 636.63. Finally, to calculate the individual input value, it is needed to divide by the sum (i.e., 636.63), and the activation function output for a specific input vector provides the probability of certain input. Thus, for the real input vector 6, and the softmax output represents that it is 63% certain that the value is 6, as shown in Figure 2.13.

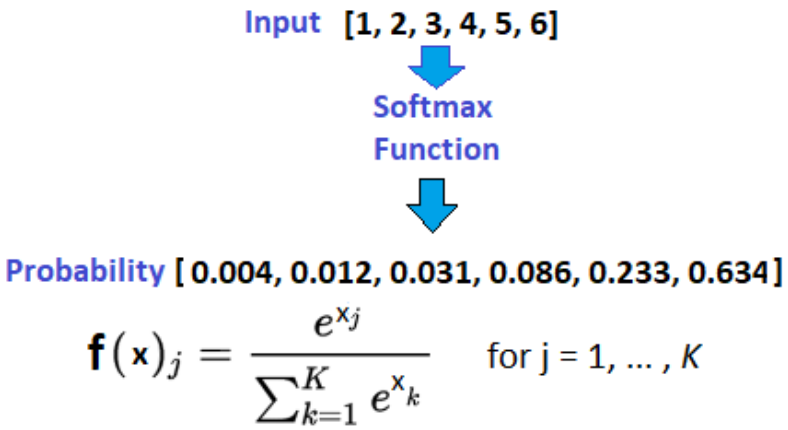


Figure 2.13: Example of the Softmax activation function.

Following are some of the advantages [17] of softmax function.

- It uses in a multi-class problem
- It shows the probability distribution for the output classifications
- Softmax activation is a normalized exponential probability for active neurons.
- Softmax can cut out the effect of cross-entropy as a loss function (discussed in section 2.3).

Batch Normalization (BN)

Batch normalization (BN) is used to significantly reduce the problem of the vanishing/exploding gradients [17, 39] at the beginning of training as well as during training. This technique can be introduced into the model before or after the activation function of each hidden layer. The main operation divides into two parts, one is simply zero-centering, and another is normalizing each input. Equation (2.6) shows both the zero-center and normalize operations are estimated from each input's mean and standard deviation. Furthermore, BN results using two new parameter vectors, one for scaling and another for shifting of each layer [17].

Moreover, if a BN layer is added in the first layer of a CNN, then the BN layer performs standardized [17] operation (e.g., using a standard scaler approach). Equation (2.6) is as follows for batch normalization.

$$\widehat{x}^{(i)} = \frac{x^{(i)} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{2.6}$$

where, μ_B represents the vector of input means, σ_B refers to the vector of input standard deviations, $\widehat{x}^{(i)}$ refers to the vector of zero-centered and normalized inputs for each instance i and ϵ is called a smoothing term. The value of 10^{-5} (typically a tiny number) [17] is used to avoid division by zero.

Regularization (Dropout)

Dropout is a popular regularization technique for DNN, and it follows a fairly simple algorithm. At every training step, every neuron (including the input neurons except all the output neurons) has a probability p [17] of being randomly "skip out". In particular, some neurons will be temporarily ignored during a specific training step, whereas during the succeeding step, the same neuron can be active. This layer can boost 1% to 2% of accuracy [17] for state-of-the-art CNN.

There are several advantages to use a dropout layer. First, it is used to defeat the over-fitting problem of CNN [25]. Moreover, it can produce thinned CNN

architectures during training by randomly "dropping out" of some connections, and finally, one representative CNN is selected with associated smaller weights [22]. The primary realization here is after completing the training process, the neurons become constant and do not skip.

Fully Connected (FC) Layers (Dense Layer)

The fully connected (FC) layer is also called a dense layer. Figure 2.14 shows a fully connected layer that depicts all the nodes in the N^{th} layer (i.e., its input neuron) connect to all the nodes in the $(N+1)^{\text{th}}$ layer [17]. A fully connected layer uses at the last layer of the CNN for classification. This layer supports a global operation, which means this layer globally analyses the output of all the preceding neurons by taking input from the feature extraction (i.e., convolution and max-pooling) layers. The advantage is, this layer is used for selecting the sample data for making non-linear combinations to predict accurate classification [22].

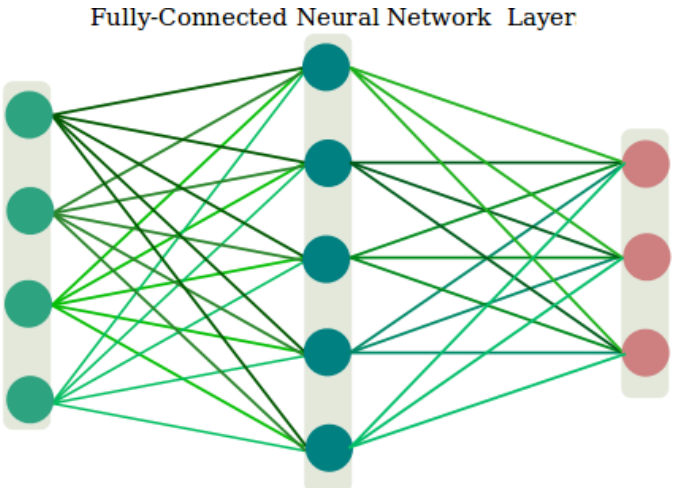


Figure 2.14: Fully-Connected Neural Network Layer.(Source of Image: [40])

Equation (2.13) calculates the size neuron for a CL to FC layer (discussed in section 2.4.1). Further, it is briefly discussed in section 2.4.1 for calculation of the size neuron from a previous FC to next FC layer.

2.2 Training Convolution Neural Network

Forward Propagation

In forward propagation, the input data is fed into the hidden layer after accepting data in the forward direction and passes to the successive layer through the network [41]. Figure 2.15 (a) shows forward propagation which can be referred to inference. In this case, the CNN model feeds input values (e.g., image) and gets an output class label (e.g., classification) value as prediction [17]. For the first layer of CNN, the input values feed without any operations, and the second hidden layer takes input from the first layer and applies equations (2.1) and (2.4) multiplication, addition, and activation operations (ReLU) and passes the output values to the next layer. The subsequent layers are following the same process and, finally, get a predicted output value from the last layer. The activation operation applies softmax in the last layer, as shown in Equations (2.5).

Feed-forward Neural Network

The feed-forward neural network is such a network that consists of pairs of input and output values that are fed in the forward direction, and it will not form a cycle; otherwise, the output could never be generated [41]. It helps in the forward propagation. The network learns from the relationship between the input and output, and this learning belongs to the realm of the neural network.

Backward Propagation

Backward propagation, in short, *backpropagation* of errors, is a widely used technique in deep learning for calculating derivatives inside feed-forward neural networks [42]. Backpropagation forms an important part of supervised learning algorithms for training DNN. Now, as discussed above, the forward propagation predicts an output value. Figure 2.15 (a) shows, after comparing the predicted value with the actual and output value, the error is calculated. To calculate the error value, CNN uses a loss function (for example, MSE, binary cross-entropy, and categorical cross-entropy). Then the derivative of the error value with respect to respective weight on CNN [17] is calculated. In particular, the chain rule of differential calculus is used in the backpropagation method.

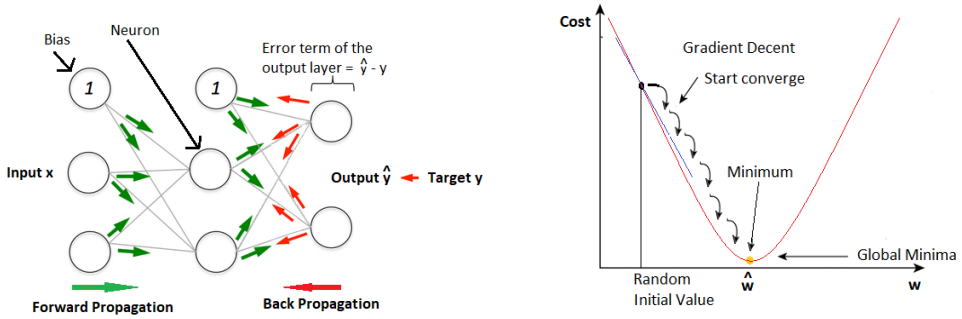
In backpropagation, to update the weights in each and every hidden layer, first, calculating the derivatives (gradients or slope) of error value with respect to the weight values of the last layer. These derivatives are called optimizer (for example, gradient descent, stochastic gradient descent, Adam [17]) values, and then the old weight value is subtracted with this gradient value to reduce the

error value. Again, for the second last layer, it uses these new weight values to calculate the derivatives.

This process will repeat until the loss between the prediction and original output class label is minimized and the gradient moves to the closer global minima [17], as shown in Figure 2.15 (b).

Convergence

Convergence is required when the iterations precede closer towards a specific output value [25], as shown in Figure 2.15 (b).



(a) Forward Propagation and back Propagation (b) Gradient decent

Figure 2.15: Training the CNN with forward propagation and backpropagation. (Image based on: [25])

Accuracy

Measuring the accuracy refers to the ratio of the correct prediction [17] to a standard or known value. Equation (2.7) shows accuracy is equal to the ratio of the accuracy of true prediction of positives and negative with respect to the overall number of instances [25].

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.7}$$

where, TP refers to the number of the true-positive, TN refers to the number of the true-negative, FP refers to the number of false-positive, and FN refers to the number of false-negative.

Precision

Precision refers to the ratio of correct true positive classifications from the total amount of measurements that are predicted as positive [17]. In Equation (2.8) shows, precision is the ratio of the accuracy of true positive (TP) prediction from the sum of true positive (TP) and false positive (FP) prediction [25].

$$Precision = \frac{TP}{TP + FP} \tag{2.8}$$

Recall (Sensitivity)

The recall is also known as sensitivity, refers to the ratio of correct positive classifications from the total amount of measurements that are actually positive [17]. Equation (2.9) shows recall is equal to the ratio of the accuracy of TP prediction with the sum of TP and TN prediction [25].

$$Recall = \frac{TP}{TP + FN} \tag{2.9}$$

2.3 Hyperparameter

A hyperparameter is a parameter that is used during the training of the ML algorithm but not affected after the creation of a model [17]. That is, when building ML models, it needs to choose various hyperparameters, such as learning rate, loss function, optimizer, the dropout rate in a layer. In particular, it must be set before training the ML system and remains constant during the model training process [17]. Further, these decisions impact of model's performance measures matrix, such as accuracy. Therefore, the ML workflow system, as shown in Figure 2.1(Stage 3), is to identify the best hyperparameters for the ML algorithm through several experiments. This experimentations an important step and process which is known as *Hyperparameter Optimization* or *Hyperparameter Tuning* [43]. The common hyperparameters are discussed in the following section.

Learning rate

It presents how fast a CNN adapts to changing data [17]. The *Gradient Descent* is used to optimize the weights on CNN. The learning rate can determine how fast or slow the update process can be conducted on CNN *weight values*. If the learning rate is *high* enough, then CNN has the possibility of overfitting. On the other hand, if the learning rate is very low, then the convergence will take many iterations. Therefore, to find the local minima, a learning rate determination is a

very important hyperparameter. Moreover, during the *backpropagation* calculation, the learning rate is multiplied with the derivative of the loss function concerning each weight and subtract from the old weight.

Loss Function or Cost Function

The loss function, is also called the cost function, computes the difference or error between the original output and predicted output for a training step [17]. The types of loss or cost functions are as follows:

Here, this project will cover the following essential loss functions [44].

- **Mean Squared Error (MSE)** - this cost function computes with regression classification problem [17]. The mathematical operation of this loss by taking the mean of squared differences between target and predicted values.
- **Binary Crossentropy (BCE)** - this cost function computes with the binary classification problem. The BCE loss function just needs one output node to identify between two classes. The output value of BSC is the range between (0 to 1) and passes through a sigmoid activation function [44].
- **Categorical Crossentropy (CCE)** - This cost function is used for the multi-class classification problem. The target value must be encoded to one-hot before feeding the target value into the last layer [17]. The output nodes must be as same as the classes for using the CCE loss function. As the final layer needs to obtain a probability value between (0 to 1) so that each node of output value passes through a softmax activation.
- **Sparse Categorical Crossentropy (SCCE)** - This cost function is identical to CCE except for a single change [17]. The SCCE loss function doesn't need to encode one-hot for the target vector. In particular, the final predicted class is just passed through the index value of that class.

Faster Model Optimizers

The main purpose of an optimizer to optimize the CNN model during training. Optimizer algorithms [17] are used to reduce the loss functions or cost functions of the neural network by changing the weights and learning rate. The CNN uses an optimizer for training due to it helps to boost huge training speed for getting results faster. Some popular optimizer algorithms will represent in this section:

- **Gradient Descent (GD)** - *Gradient descent (GD)* is the most popular and used algorithm as an optimizer. It is used in linear regression, different classification models, and neural network's backpropagation method. Gradient descent performs the first-order derivative operation of a loss

function, so it is called a first-order optimization algorithm [45]. Through backpropagation, the loss function reaches global minima by transferring, modifying, and minimizing the model's parameters (i.e., weights) from one layer to another layer starting from the end.

Equation (2.10) for GD is:

$$\theta \leftarrow \theta - \eta \nabla_{\theta} J(\theta) \tag{2.10}$$

where, θ represents weights, $J(\theta)$ is a cost function, η refers to learning rate, and $\nabla_{\theta} J(\theta)$ represents derivative of the weights.

The main advantages of GD are easy to compute, implement, and understand. However, the problems with GD are: it may *trap* at local minima, and for the larger dataset, it may take a *long time* to reach the global minima. The memory consumption is also very high for the larger dataset.

- **Stochastic Gradient Descent (SGD)** - *Stochastic gradient descent (SGD)* updates the model's parameters faster [17] than Gradient Descent. It picks the model parameters randomly and computes the loss (i.e., calculate gradient) on that training parameters.

  The main advantages of the optimizer are it requires less converge time and memory. It reaches close to the global minimum by bouncing up and down [17]. However, the disadvantage is it may never reach the minima, or even, for the low learning rate value, the SDG can stick to the halfway of the minima.

- **Adaptive Moment Estimation (Adam)** - *Adaptive moment estimation (Adam)* works with the newly introduced hyperparameter called momentums. It can keep track of the past gradients [17] and the gradient value is dropping exponentially. The first momentum ($\beta 1$) value is initialized to 0.9 and the second momentum ($\beta 2$) value is initialized to 0.999. The advantage of the optimizer is, the algorithm converges rapidly and it overcomes the problem of *vanishing learning rate* [45]. However, the algorithm may computationally costly for some datasets.

In Table 2.1 shows the comparison of all the optimizer are discussed above.

Table 2.1: Optimizer Comparison

Optimizer Class	**Convergence speed**	**Convergence quality**
GD	Bad	Bad
SGD	Bad	Good
Adam	Good	Average or Good

Performance Metrics

Performance metrics are used to evaluate the performance of a classifier of ML such as logistic regression, random forest, support vector machine (SVM) , and neural network (NN) . There are many performance matrices available such as *accuracy*, area under the curve (AUC) , confusion matrix, F_1 score, *loss*, *validation accuracy*, *validation loss*, mean absolute error (MAE) , mean squared error (MSE), precision, recall, root mean square error (RMSE) , and receiver operating characteristic (ROC) curve [17].

Batch Size

The number of training samples are utilized in one iteration (in *forward/backward pass*) [17]. The main benefit of the batch size measures the performance and training time of the classifiers. Masters et al. [46] proposed that using the small batch size (from 2 to 32) would lead to better classifiers in less training time and hardware such as *GPU* , which can perform and process the training examples efficiently. However, a larger batch size can lead to training instability [17] and need more memory space. Consequently, the classifiers' performance will not be satisfactory.

Training Epochs

The classifier model iterates by round [17] and each round is equal to *one epoch*. Several numbers of iteration (for example, m iteration) can be possible in each round. More specifically, one epoch represents one forward pass and one backward pass for all the training dataset.

Dropout rate

Nitish Srivastava et al. [47] demonstrated that dropout has a tunable hyperparameter p, which is the probability of neuron retaining in the training step in the neural network. The dropout rate is closer to the $40-50\%$ in CNN [17], and it is typically set to 50% by default. However, after completion of the training process, the neurons don't get dropped anymore.

2.4 AlexNet Model

In Figure 2.16, AlexNet CNN architecture was developed by Alex Krizhevsky et al. [48], and it was the first work that popularized CNN in computer vision. This CNN won the *ImageNet ILSVRC-2012 challenge* with an error rate of 15.3%. The

neural network had a very similar architecture to LeNet. The common traditional architecture (such as LeNet) was to have a single convolutional layer always immediately followed by a pooling layer. However, it was stacked convolutional layers directly on top of each other without stacking a pooling layer on top of each convolutional layer [17]. Table 2.2 presents this AlexNet architecture, which has more than 62 million trainable variables.

It was described, the architecture of AlexNet CNN was illustrated explicitly to show the representation of responsibilities between the two GPUs. Figure 2.16 shows the top layer-parts ran into one GPU while the bottom layer-parts ran at the other GPU. Further, the GPUs communicated only at certain layers. Furthermore, it was explained input dimension of AlexNet network was $150,528$, and the number of neurons in the network's remaining layers were given by $253,440$ - $186,624$ - $64,896$ – $64,896$ – $43,264$ – 4096 – 4096 – 1000 [48].

The authors used in his model two regularization techniques for reducing *overfitting*: at first, they introduced a *dropout layer* for fully connected (FC) layers of 8 and 9 with a 50% *dropout rate* [48]. Secondly, they used random data augmentation (such as shifting, flipping, and changing light conditions of an image) technique for training the model.

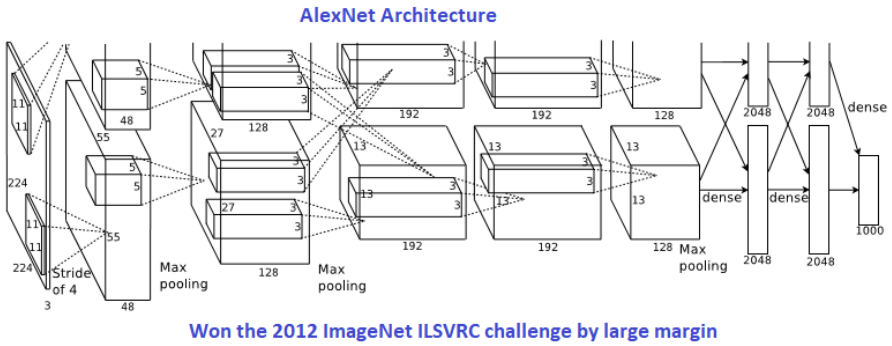


Figure 2.16: An illustration of the architecture of AlexNet CNN architecture Model.(Source of Image: [48])

Table 2.2: AlexNet CNN - Architecture Details

Layer	Input	Maps	Kernel Size	Stride	Padding	Activation	Output	Parameters #
Input	227x227	3(RGB)	-	-	-	-	-	-
Convolution	227x227	96	11x11	4	valid	ReLU	55x55	34944
Max Pooling	55x55	96	3x3	2	valid	-	27x27	0
Convolution	27x27	256	5x5	1	same	ReLU	27x27	614656
Max Pooling	27x27	256	3x3	2	valid	-	13x13	0
Convolution	13x13	384	3x3	1	same	ReLU	13x13	885120
Convolution	13x13	384	3x3	1	same	ReLU	13x13	1327488
Convolution	13x13	256	3x3	1	same	ReLU	13x13	884992
Max Pooling	13x13	256	3x3	2	valid	-	6x6	0
FC	9216	-	1x1	-	-	ReLU	4096	37752832
FC	4096	-	1x1	-	-	ReLU	4096	16781312
FC	4096	-	1x1	-	-	Softmax	4096	4097000
			Total=					62378344

Table 2.3: Comparison of the two normalization technique

Normalization type	Actions	Trainable	Training parameters	Regularization
LRN	Inhibition across neuron	No	0	No
BN	zero-centric and normalize neuron	Yes	2 (Scale and shift)	Yes

AlexNet model used the local response normalization (LRN) method for both convolutional 1 and 3 layers immediately after the ReLU activation function applied [48]. Table 2.3 shows a comparison of the LRN and BN techniques [39, 49].

BN technique consists of zero-centric and normalizes each input position by scaling and shifting operation, and it also acts as a regularizer [17]. Furthermore, the LRN performs normalization across the same position neighboring neurons by observing activated neurons inhibit other neurons and improving generalization by exploring a wider range of the feature maps [17].

2.4.1 Calculate the Tensor Size at Each Stage

In this section, the calculation of the tensor size at each layer is explained.

- **Size of the output images (tensors) of convolutional layers**
  Equation 2.11 explains the calculation of convolutional Layers. The size, define with (O) of the output image is given by

$$O = \frac{I - K + 2P}{S} + 1 \tag{2.11}$$

where, O is size of output image, I is size of input image, K is size of kernels used in the convolutional layer, S is strides in convolution operation, and P is Padding of the convolutional layer.

- **Size of output images (tensor) of a max pooling layer**
  Equation (2.12) explains the calculation of max pooling layers. The size, define with (O) of the output image is given by

$$O = \frac{I - P_s}{S} + 1 \tag{2.12}$$

  where, O is size (i.e, height, width) of output image, I is size (i.e, height, width) of input image, S is strides of the convolution operation, and P_s is pool (filter) size.

  Unlike the convolution layer, the number of channels in the max-pooling layer's output is not changed.

- **Size of the output neuron for a fully connected layer**
  A *fully connected layer* outputs a vector with product of height(h), width(w) and number of channels(f) with respect to last convolutional layer. Therefore, the number of neurons in the FC layer is calculated, as given in Equation (2.13).

$$O = h \times w \times f \tag{2.13}$$

  There is *no mathematical rule* behind the choice of the number of neurons in a dense(FC) layer if it comes after another FC layer [27]. Since the layer is FC, it is possible to choose any size, just like typical artificial neural networks such as multilayer perceptron(MPL). Hence, all neurons in a fully connected such as FC_2 layer connect to all the neurons in the previous FC such as FC_1 layer [34].

2.4.2 Calculation of Parameters in the Neural Network (NN)

In this section, the number of parameters of the NN will calculate for each layer.

- **Number of parameters of a convolutional layer**

  Each convolutional layer of CNN consists of two kinds of parameters, i.e., weights and biases (discussed in section 2.1). Therefore, Equation (2.14) shows that the total number of parameters is the sum of all weights and biases.

$$\begin{aligned} W_c &= K^2 \times C \times N \\ B_c &= N \\ P_c &= W_c + B_c \end{aligned} \tag{2.14}$$

where, W_c refers to the number of weights of the convolutional layer, B_c refers to the number of biases of the convolutional layer, P_c refers to the number of parameters of the convolutional layer, K refers to the size (e.g., height, width) of kernels used in the convolutional layer, N refers to the kernel's number, and C refers to the input image channels in the layer.

The *channels* (defined in section 2.1) of output images are always equal to the number of kernels N.

It is shown in the previous sections 2.1 that in a convolutional layer, the depth of every kernel and the number of channels of an input image are equal. In Equation (2.14), if there are N number of kernels in any NN convolutional layer, then every kernel will consist of $K^2 \times C$ parameters.

- **Number of parameters for a max pooling layer**

  There are no associated parameters that can learn in a Max Pooling layer [50]. This layer consists of only hyperparameters, e.g., pool size, strides, and padding. In particular, this layer is used to reduce the dimension size of the image. In this layer, no backward propagation operation is involved because of no learnable parameters (weights) [17].

- **Number of parameters in fully connected (FC) layers**

  There are *two types* of FC layers on CNN. *First type*, FC layer is connected to other FC layer, and *second type*, FC layer is connected to the last convolutional layer. In Equation (2.15) and (2.16), both cases are presented separately.

  (i) In Equation (2.15), the number of parameter calculations to one FC layer connected from another CL is given as

  $$\begin{aligned} W_{cf} &= O^2 \times N \times F \\ B_{cf} &= F \\ P_{cf} &= W_{cf} + B_{cf} \end{aligned} \tag{2.15}$$

  where, W_{cf} refers to the number of weights of FC layer, B_{cf} refers to the number of biases of FC layer, P_{cf} refers to the number of parameters of FC layer, O refers to size (i.e., height, width) of the output image of the CL, N refers to the number of kernels in the CL and F refers to the number of neurons in the FC layer.

(ii) In Equation (2.16), the number of parameters calculation from a FC layer connected to other FC layer is demonstrated.

In Equation (2.16), the total number of connection weights of neurons from the first FC layer to the current FC layer are calculated with $F_{(-1)} \times F$. The total number of biases is equal to the number of neurons of the current layer(F).

$$\begin{aligned} W_{ff} &= F_{(-1)} \times F \\ B_{ff} &= F \\ P_{ff} &= W_{ff} + B_{ff} \end{aligned} \tag{2.16}$$

where, W_{ff} refers to the number of weights in the FC layer, B_{ff} refers to the number of biases in the FC layer, P_{ff} refers to the number of parameters of FC layer which is connected to other FC layer, $F_{(-1)}$ refers to the number of neurons in the first FC layer and F refers to the number of neurons in the current FC layer.

Chapter 3

State of the Art

This chapter describes the general and cybersecurity viewpoint of the state-of-the-art adversarial machine learning (ML) (for example, CNN) model. In this chapter, three types of adversarial threat modeling for the ML algorithm are simulated: training, testing, and manufacturing. This chapter also focuses on several categories of security threats such as white box and black box attack for different perspectives of the ML model.

Further, the basic theory and procedure for generating adversarial examples are discussed to understand how a neural network can be deceived by an adversary and misclassified in the output label. Two attack techniques, such as the fast gradient sign method (FGSM) for the white-box attack and the iterative fast gradient sign method (I-FGSM) for the black-box attack, are described in this chapter.

3.1 Adversarial Machine Learning

The most alarming situation in the state-of-the-art CNN models is that it is currently adopted in real-world applications without adequately revealing the security risk. Therefore, it is crucial to understand why cybersecurity is vital in the CNN model revolution. In this project, some of the real-life examples are discussed to understand this situation.

The first example is adopted by Tesla cars. Tesla has made progress enormously in a self-driving car. However, performing research by Tencent Keen Security Lab [51] has found security defects in the system design of Tesla. The research group found that Tesla could be deceived to recognize an accurate road traffic sign by putting stickers [52] on it, as shown in Figure 3.1. The Tesla car interpreted the traffic sing as the lane diverging, and it kept driving into oncoming traffic. Tesla has been underestimated the severity of this situation as well. Furthermore, in another experiment, the weaknesses were found in the perception systems [52] of the vehicle by the research team. The visual sensor system could be fooled by projecting noise, which triggered the auto wipers by placing an electronic display

in front of the vehicle.

Figure 3.1: Road traffic sign with putting stickers.

Faiq et al. [53] demonstrated that a stop sign could be manipulated with adversarial patches by adding noise (e.g., applying Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) Method [54, 55] and Fast Gradient Sign Method (FSGM) [56, 57, 58]) in the image and could be fooled the state-of-the-art model by thinking that it was a speed limit (60 km/h) sign, as shown in Figure 3.2 [53]. In particular, from this example, it can be possible to imagine that an attacker (such as a black-hat, gray-hat hacker, and outsider) can manipulate the road traffic signs in such a way that will help self-driving cars to break traffic rules.

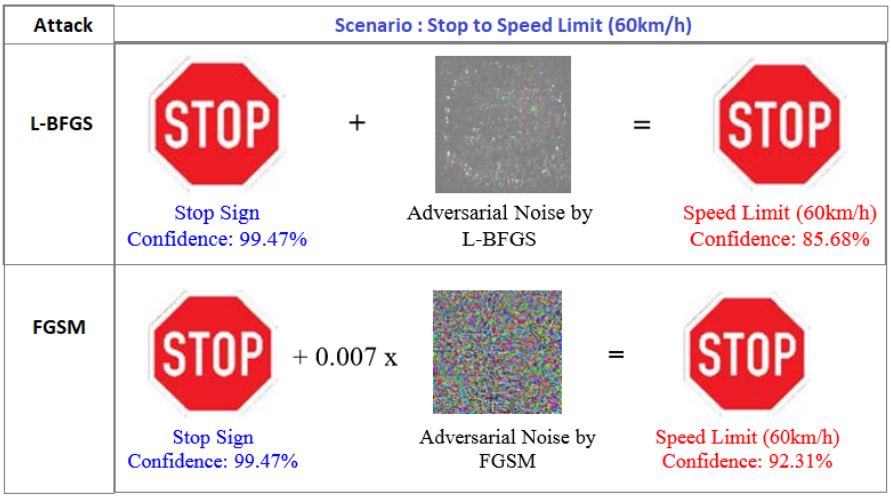


Figure 3.2: Adversarial Image generated by L-BFGS Method and FSGM. In FSGM experiment, the value for "ϵ" is 0.007 to make it imperceptible.(Source of Image: [53])

The Viewpoint of Cybersecurity

Currently, state-of-the-art CNN is integrated into many systems, such as autonomous vehicles, voice recognition, face detection, medical devices, and many more. Therefore, after training the ML model, it can predict unknown data such as image, sound, and pattern. Whereas for defense against attacks on CNN, models have to deal with unknown datasets. Thus, there are differences between the neural network's accuracy and robustness, which is explained later in this section.

Accuracy refers to how much the ML model can predict the precise class of data for a 100% test dataset. At the same time, the robustness [2] of a model refers to reliability as well as protection from security vulnerabilities.

Biggio et al. [59] represent about three golden rules in cybersecurity: the first one is to know your adversary, then, to be proactive, and finally, protect yourself from the attacker.

Knowing that adversaries can represent as the defender has fully understood the goals of the attackers. There are two types of defense techniques that are illustrated in Figure 3.3. Those are (a) proactive and (b) reactive [60] defensive method. Qiang Liu et al. [60] argued that the defender should perform proactive defense assessment instead of a reactive one. Figure 3.3 shows difference between proactive and reactive defense mechanisms.

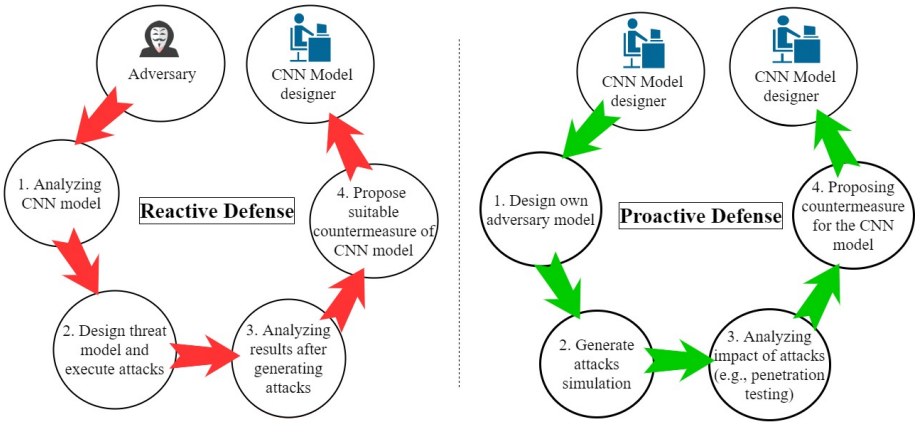


Figure 3.3: A workflow representation of the reactive (left) and proactive (right) defensive mechanisms.

3.2 Literature study

In this section, several state-of-the-art methods for generating adversarial threat model for different datasets is described, which has been studied in recent years.

Szegedy et al. (2014) [54] demonstrated that DNNs (AlexNet) had counter-intuitive properties that were referred to as *adversarial negatives (adversarial examples)*. Such property can be misleading to AlexNet. For studying the AlexNet network MNIST and ImageNet datasets were considered. L-BFGS techniques were used to generate adversarial negatives. The formal method of L-BFGS was to minimize random perturbation noise (r), which was not unique and was added to the regular image(x). It was observed that if the target label was l, and minimum random perturbation noise was r, then $x + r$ would be an arbitrarily chosen image, where "loss function" $D(x, l)$ should be minimum. It was found that the exact computation of $D(x, l)$ was a hard problem, and it was also non-trivial. Concretely, it was also found that an approximation of $D(x, l)$ by performing *iterative-search* to find minimum error parameter (c) for which the above problem satisfies $f(x + r) = l$ with the minimum r. The penalty function method for L-BFGS to generate adversarial negatives is presented in Equation (3.1).

$$\text{Minimize } c.\,|\,x\,| + loss_f(x + r, l) \text{ subject to } x + r \in [0, 1]^m \tag{3.1}$$

Szegedy et al. (2014) did not present the frequency of adversarial negatives that appears in the image, which they left and addressed for the future research [54].

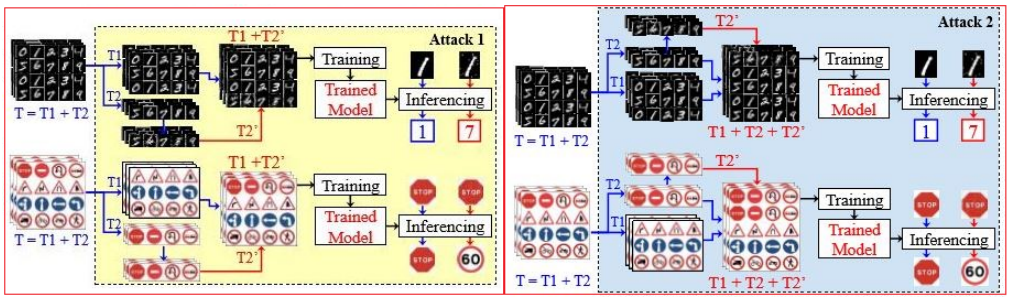
Sabour et al. (2015) [61] developed a new method for creating adversarial examples images that could be mimic natural images, with only imperceptible and minor perturbations to the original image. This new method was called *feature adversaries*. In this case, they misclassified the class label with the DNN (AlexNet). However, they also had looked into the internal layers of DNN representations. AlexNet network was used with ImageNet ILSVRC dataset for preparing adversarial examples using *feature adversaries* attack method. While the AlexNet image's internal representation was similar to perceptually generated one image by an adversary, it was found to form a different class label. Finally, no defense method was mentioned, and a concerning issue of *discriminative model* was left unclear that might be trained to *detect feature adversaries* with a large diverse, and relative dataset of adversarial images.

Table 3.1 [5] shows a summary of vulnerability and threat that exists with the AlexNet CNN model for adversarial examples.

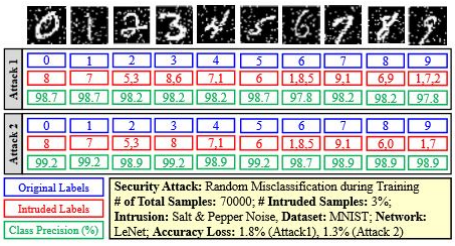
Table 3.1: A summary of existing attacks for the AlexNet CNN model

Attack Method	Adversarial Falsification	Adversary's Knowledge	Adversarial Specificity	Attack Frequency	Perturbation Measurement	Datasets
L-BFGS attack	False Negative	White-Box	Targeted	Iterative	l_2	MNIST, ImageNet
Feature Adversary	False Negative	White-Box	Targeted	Iterative	l_2	ImageNet

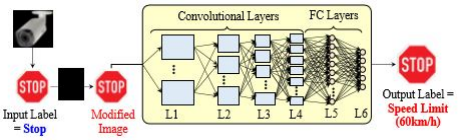
Khalid et al. (2018) [62] discussed training data poisoning attack and adversarial examples using L-BFGS and FSGM methods. To identify the possible security threats on the training phase, they developed threat models, attack methodologies, and payloads against the CNN algorithm such as the LeNet for the MNIST dataset, and the VGGNet for the German traffic sign recognition benchmarks (GTSRB) dataset, as shown in Figure 3.4(a).



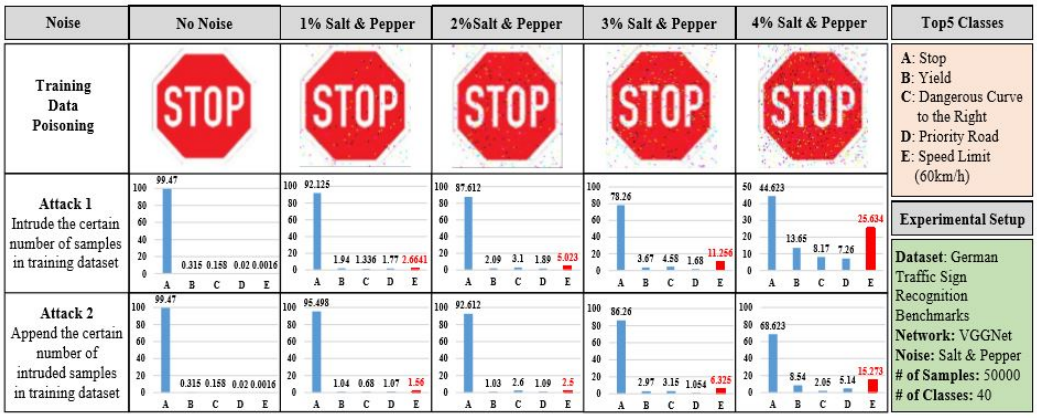
(a) Random misclassification attacks on LeNet and VGGNet for MNIST and GSRB dataset



(b) Random misclassification attack (i.e., introducing Salt & Pepper noise in 3% data samples of the MNIST dataset) on LeNet during the training phase.



(c) Threat model and experimental setup for inference attacks, i.e., adversarial examples.



(d) Random misclassification attack, i.e., introducing the salt and pepper noise in GTSR.

Figure 3.4: Attacks and Challenges for Training and Inference.(Source of Image: [62])

In this work, the noise was used against perturbed images, which was called *Salt & Pepper*. At first, they demonstrated the *traditional attack* (attack 1 in Figure 3.4(b)) on LeNet, which was trained for the MNIST dataset with *intruded samples* (3%). Similarly, they also performed a *poisoning attack* (Attack 2 in Figure 3.4(b)) by training LeNet with *appending* only 3% intruded samples in the MNIST dataset. Further, they illustrated both the attack (the traditional attack and poisoning attack) on the VGGNet for GTSRB dataset towards random misclassification, as shown in Figure 3.4(c). For example, when 1% salt and pepper noise was added with original "Stop" traffic sign image, then the image was misclassified to traffic sign "Yield" for both *attack 1 (traditional attack)* and *attack 2 (poisoning attack)*.

Furthermore, *Khalid et al. (2018)* demonstrated the impact of security threats on the inference phase by producing adversarial examples, as shown in Figure 3.4(d). To illustrate the adversarial examples, they considered a couple of attack techniques (L-BFGS and FSGM) on the GTSRB dataset. Finally, *Khalid et al. (2018)* provided a general overview of research challenges in developing secure/robust ML-based system against these attack methods and highlighting possible countermeasures concerning training, inference, and hardware implementation phase.

Liu et al. (2018) [60] discussed taxonomy of attack and defense technique. They also made comparative analysis for different attacking and defensive techniques for machine learning. Further, a summary of related works regarding security threats, modeling, and attack against machine learning had been prepared. It was argued that the research on security threats and defensive techniques of machine learning based on the existing literature could be in the following trends:

(i) New security threats are constantly emerging towards machine learning. Hence, the ML algorithm relies on the high quality of a dataset, a weak defensive technique against adversarial samples. Furthermore, adversarial samples' detection methods are difficult and create a significant challenge on performance against machine learning. Hence, *Liu et al. (2018)* argued that a meaningful research point could be designed for new adversary models from the perspective of an attacker.

(ii) Further, *Liu et al. (2018)* argued that a well-defined and widely adapted security assessment standard on machine learning could become a prevailing research area of decision systems in adversarial environments.

(iii) *Liu et al. (2018)* also argued that adversarial samples in training models and improving the robustness of ML algorithms are new attraction point in the security field of ML.

(iv) Finally, a larger overhead and a lower generalization performance are induced towards ML algorithms when a higher security level is introduced.

There are also several challenges of ML application for security optimization. Hence, the design of secure machine learning algorithms with proper balancing among security optimization, lower overhead, and larger performance generalization was recommended to facilitate the ML models' practical deployment.

Khalid et al. (2019) [63] discussed the limitation of the FGSM method in their research. FGSM method was robust with white box attacks rather than for black-box attack. They changed several variants of FGSM to handle proposed black-box assumptions. However, it increased converging time, which conflicted from applicability in real-world applications.

Zhang et al. (2019) [2] focused on adversarial attacks which were targeted inference of DNNs. It was assumed that an attacker could not access the training process and the dataset, whereas the adversary could access the trained CNN model in both white-box and black-box settings. Therefore, the imperceptible noise was generated for the white-box attack by exploiting the gradient (e.g., FGSM, L-BFGS) method. Further, for the black-box attack, more complex Iterative procedures (e.g., basic iterative method (BIM)) were used for image perturbation. *Zhang et al.* discussed a few countermeasures to protect from exploiting the gradient of CNN. The proposed countermeasures were gradient masking, input noise filtering, and adversarial training based defenses. *Zhang et al.* represented a couple of research directions.

- First, only limited design methods are available for building robust CNNs, which inherently resilient the effects of reliability threats.
- Second, the robust ML Training, focus on achieving high accuracy and reliability not only for specific dataset but also unknown adversarial perturbations.

3.3 Adversarial Threat Model and Taxonomy

Lowd et al. [14] proposed the machine learning algorithm's adversarial learning framework. Biggio et al. [64] discussed the four criteria: specific goal, knowledge limitation, attacking strategy/model, and the capability to attack to construct the well-designed adversarial model. The adversary's goal should be clear and have a significant impact on the security threat of CNN. Furthermore, the adversary should have minimum knowledge or no knowledge of the neural network, such as the CNN algorithm, training dataset, testing dataset, CNN model architecture, and hardware implementation. The threat model is the specific state-of-the-art behaviors of the adversary for manipulating training and/or testing datasets to achieve his goals efficiently and effectively. Finally, the adversarial skillset of generating an attack may interpret from the following aspects:

1. causative or exploratory impact on security threats,
2. an attacker has a control on the percentage of training and/or testing dataset,
3. an adversary knows, such as features, parameters, and hyperparameters, of the neural network model.

For example, the adversary's goal is to lunch an attack on privacy violation for face detection [5]; thus, the adversary needs to know the CNN model, then it decides to obtain the sensitive data illegally and manipulate the integrity of the dataset; finally, the skilled adversary will lunch an attack via tampering the features and misclassifying of category labels for any targeted or untargeted user.

Adversarial Threat Model

Concerning Figure 3.5(a-c), the major types of threat models can be developed for the neural network security attacks. For a systematic threat model [63], it is important to know about the possible actors (attackers and users), the intension of the attack, the skillset needed for generating an attack and discriminate attacks on the non-targeted and targeted attack. The best practice to design a threat model to adopt all assumptions and analyze the worst-case scenarios. In this project, the following three threat models are discussed:

1. **Threat Model I**: An attacker has access to the input (such as training dataset and server) and training of the model. Hence, the assumption is an adversary can generate its own perturb image dataset and store those data with the data server's training dataset. Therefore, when CNN processes both the original and perturb image datasets during the training phase, it will develop a poisonous model, as shown in Figure 3.5(a).
2. **Threat Model II**: In Figure 3.5(b), an attacker may exploit the vulnerability of the faulty hardware design such as hardware Trojan, permanent fault, and timing error. Moreover, an attacker may change the CNN model's parameter in the memory segment with privilege escalation. Further, an attacker may manage to generate the key (such as a private and public key) to access the confidential data with a side-channel attack.
3. **Threat Model III**: In Figure 3.5(c), an attacker may generate a perturb image with the testing data set and pass through the validated CNN model to misclassify the category of the classification. Furthermore, attackers have access to the traffic sign in the physical-world and add a sticker [65] as a pre-processing noise so that the CNN model acquired an incorrect output classification label, as shown in Figure 3.1.

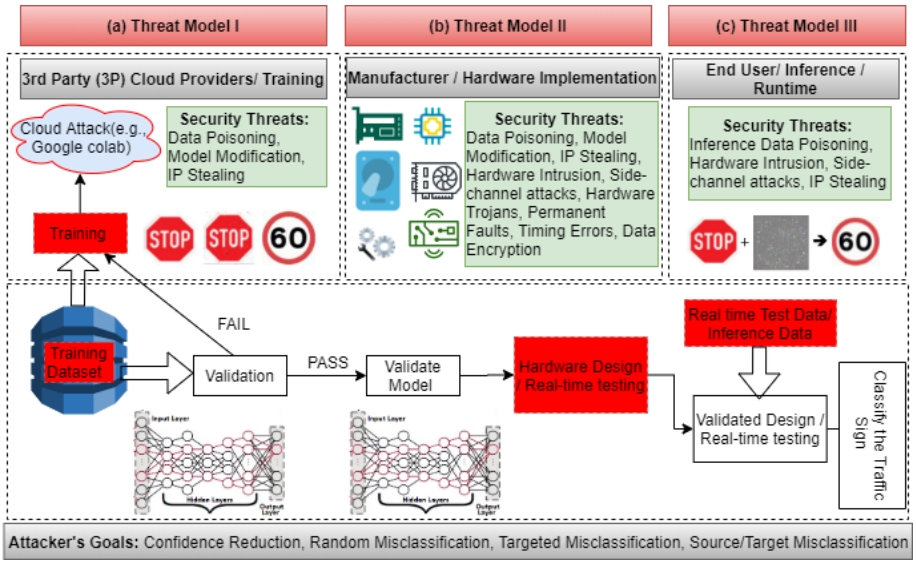


Figure 3.5: Security threat/attack model on CNN-based system during training, runtime and hardware implementations. (Image based on: [53])

CIA Triad

Figure 3.6 shows, confidentiality, integrity, and availability are abbreviated as CIA triad. It is mainly the heart of information security for building a secure system. The CIA triad is widely used in information security to model and guide a system/application to secure data.

The three parts of the CIA triad is summarized as follows:

- **Confidentiality -** to limit the least privileges to whom can access to information. *Confidentiality* allows only authorized people to have access to data and prevents the disclosure of data to unauthorized people, which is known as the need to know basis. For example, the medical history of a patient will only available to doctors and nobody else. Further, confidential data must be kept as *encrypted*.

- **Integrity -** to govern how and when authorized people can modify information. *Integrity* means that to recognize that data has not been altered or tampered. *Hashing* is a technique that takes the data and converts it into a numerical value. Therefore, if tampered will be taken place, then the numerical value will also change. Common hashing algorithms are Secure Hash Algorithm version 1 (SHA1) and Message Digest version 5 (MD5).

- **Availability -** to ensure authorized people can be able to access information when they need to do so. Therefore, *availability* ensures that data will always

be available to authorized users. For example, if a credit card user wants to purchase an airplane ticket with his card, the system comes back with an error and can not purchase it. Hence, this must be frustrating. Furthermore, A *denial-of-service*(DoS) attack is a cyber-attack on the availability of a machine, a computing system, or network resources, in which an adversary seeks to make service unavailable to its intended users by temporarily or indefinitely disrupting services.

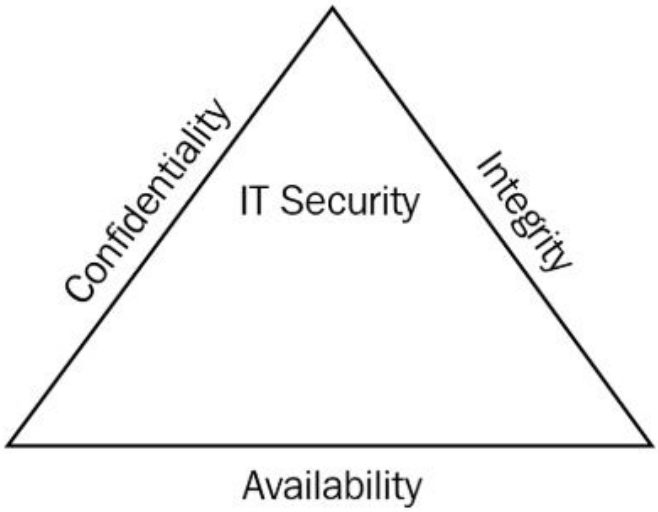


Figure 3.6: CIA triad, looks simple but actually complex.(Source of Image: [66, 67])

Security Threat Taxonomy

In Figure 3.7, the taxonomy of security threats [2, 53, 60, 4] in different perspective [68], due to adversarial capability of manipulating sample (i.e., dataset, model, etc.) [69], and finally, the reliability threats of hardware architecture and security vulnerability toward the machine learning algorithms, such as logistics regression, support vector machine (SVM), and DNNs.

1. **White-Box Attacks** - The white-box attack means an adversary has full knowledge and total access to a machine learning model such as type of neural network training algorithm, number of layers, and parameters.

2. **Black-Box Attacks** - The black-box attack assumes an adversary has no knowledge about the inner architecture of the model and uses information about analyzing the settings or feeding inputs to the model.

   Black-Box attacks can be classified [4] into three categories : (i) Non-Adaptive Black-Box Attack, (ii) Adaptive Black-Box Attack, (iii) Strict Black-Box

Attack. Furthermore, the adaptive black-box attack assumption allows the adversary to access the target ML model, which does not know about the model's training process and inner architecture.

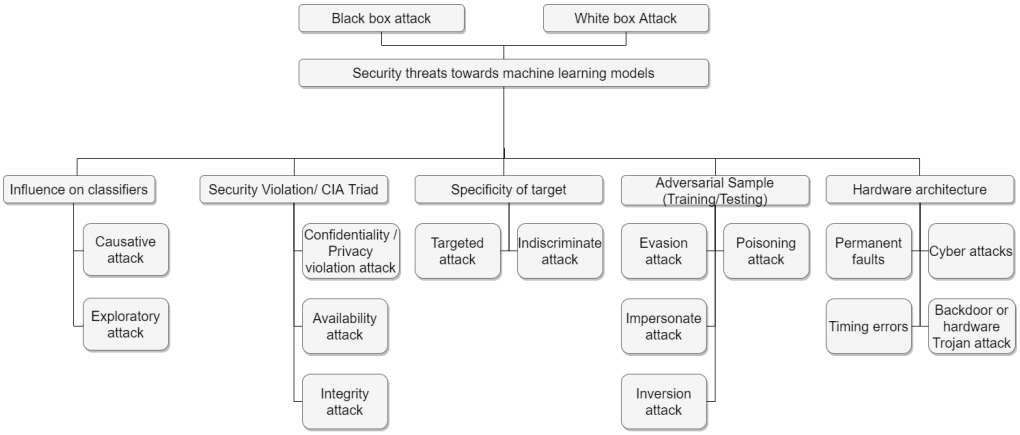


Figure 3.7: Taxonomy of adversarial security threat/attack towards machine learning model.

Form the above-mentioned taxonomy of security threat/attack in Figure 3.7 towards machine learning model, Table 3.2 presents a summary of the qualitative analysis [2, 60, 5] of different *threat perspectives/techniques of attacks* with respect to an adversary.

Table 3.2: Comparative analysis concerning different attacking technique

Attack Technique	Influence on Classifier	Adversary's Knowledge	Adversarial Specificity	Adversarial Characteristics	Hardware Perturbation
Poisoning	Training	White-box	Targeted	Causative attack, Availability/ integrity attack	Backdoor or hardware Trojan attacks
Evasion	Testing	White-box & Black-box	Targeted/ Indiscriminate	Exploratory attack, Availability/ integrity attack	Cyber Attacks
Impersonate	Testing	White-box & Black-box	Targeted/ Indiscriminate	Exploratory attack, Privacy violation/ integrity attack	Permanent Faults, Timing Errors
Inversion	Testing	White-box & Black-box	Targeted/ Indiscriminate	Exploratory attack, Confidentiality/ privacy violation attack	Permanent Faults, Timing Errors

Figure 3.7 shows security threats classifies into **two categories** for *influence on classifiers*:

(a) **Causative attack** means adversaries can change the training data [60]. It influences parameter adjustments when retraining occurs of learning models. The goal of the adversary is, to reduce significantly in performance and confidence of classifiers, as shown in Figure 3.5.

(b) **Exploratory attack** means the adversary tries to obtain knowledge of the sensitive information about the learning algorithm and training data of the underlying system [4]. The attacker follows the black-box access pattern, and he does not modify the model's training dataset. The goal of the adversary is to misclassify the source/target, as shown in Figure 3.5.

Figure 3.7 shows security threats divide into **three types** concerning the *security violation* with respect to CIA triad:

(a) **Confidentiality/ privacy violation attack** [59] - the attackers try to obtain confidential and sensitive information from training data and ML models. The goal of an adversary is to violate the privacy of the targeted user after launching such attack, as shown in Figure 3.5.

(b) **Integrity attack** - the attackers try to produce harmful samples as false negatives [60]; hence the ML model will incorrectly classify the output as negative labels.

(c) **Availability attack** [59] - the adversary compromises benign samples, and such attack increases the false positives [60] to classify samples; hence the usual system functionalities will not available to the legitimate users.

Figure 3.7 shows security threats towards machine learning model that categorize into **two types** from the *specificity of the attack* perspective:

(a) **Targeted attack** - the adversary tries to construct inputs for one specific target sample that directed the output of the classification model [4] to be the particular target class. The goal of the adversary is highly focused on a reduction in performance and misclassification of targeted class in classifiers, as shown in Figure 3.5. For example, in *Threat Model III*, an input image *'stop'* sign to the classification model will be predicted as a class *'speed limit (60km/h)'* sign, as shown in Figure 3.5(c) [63].

(b) **Indiscriminate attack** [60] - the adversary tries to alter the output classification indiscriminately so that such an attack causes the classifier to fail to predict the original class. The goal of the adversary is highly focused on a reduction in performance and random misclassification of classifiers, as shown in Figure 3.5. For example, an input image of a *'stop'* sign to the classification model will be predicted as *any other class different* from the 'stop' sign.

Many researchers demonstrated and explained that an adversary could create an adversarial sample [56, 60, 4, 69] which exploit many security vulnerabilities and threats towards machine learning. Adversarial samples are the harmful samples that can manipulate the original data and create a counterintuitive problem in DNNs. Figure 3.7 shows the extended categories [60, 4, 7] of adversarial samples:

(a) **Poisoning attack** – Such attack is also categorized in the perspective of *causative attack*. Training is a crucial and essential part of machine learning to obtain a proper classification with a target dataset. Hence, many attackers target the training phase, injecting harmful sample data with the original training dataset, resulting in a significant reduction of the ML model's performance. Such an attack causes disruption of the availability and integrity [60] towards the ML model.

(b) **Evasion attack** - Such attack generates adversarial samples during *testing phase* that can evade detection for the ML model [60, 4] such that the overall performance is significantly decreased because of the severe security threat of target systems. It is the most common type of attack [4] on the ML model in the real world, and it does not have any influence in the training phase.

(c) **Impersonate attack** - Such an attack prefers to imitate victims' data samples [70] such that the adversary can succeed to access or control the victims' systems [60] using his authority. Many researchers represent an attacker who aims to impersonate and fool [3, 11] the ML model by generating specific adversarial samples such that the existing model classifies the original samples incorrectly with different labels. The adversary's goal is to disrupt the privacy violation of ML models via injecting adversarial samples to the testing data set, and it is a type of *integrity attack*.

(d) **Inversion attack** - This type of attack is also categorized in the perspective of *confidentiality/ privacy violation attack*. It utilizes the application program interfaces (APIs) [60] of the existing ML model to collect information regarding targeted models. Further, the adversary is applied to the reverse analysis [60] method on the gathered privacy data followed by feeding the malicious data as a security threat towards the targeted ML models, e.g., facial recognition data of users [68].

Currently, research is focused on developing state-of-the-art DNNs to significantly improve and optimize the ML algorithm, such as compute-intensive, reduce the computational complexity, and the memory requirements [2]. The aforementioned-objectives need to focus on developing efficient application-specific hardware design for DNN-based applications [71]. Moreover, current progress, DNNs are also being explored and analyzed using the Tensor processing unit (TPU) [72] . These types of systems in the ML model must include inherent

reliability threats and security vulnerabilities [2]. Figure 3.7 shows different type of attack on Hardware architecture:

(a) **Permanent faults** - Figure 3.5 shows the *threat model II* which describes the possibility of injection stuck-at-faults at the internal nodes [2] in the hardware such as multiplier-and-accumulate (MAC) units, register transfer level(RTL) , and processing engine (PE) .

(b) **Timing errors** - The adversary tries to analyze the timing error propagation (TEP) approach [73] such that error will propagate to subsequent stages of computation. The TEP can reduce the ML model classification accuracy quickly.

(c) **Cyber attacks** - Figure 3.5 shows cyber security threat and traditional obfuscation techniques for *remote cyber-attacks* [2] and manipulation, in training, hardware implementation, and runtime stages, such as malware injection, side-channel analysis, IP stealing, distributed denial-of-service (DDoS) attack and man-in-the-middle (MITM) attack [53, 74].

(d) **Backdoor or hardware Trojan attacks** - Huang et al. defined hardware Trojan [75], which is the defects of ICs or intellectual property devices such as malicious and intentional inclusions/deletions/ alterations, or inadvertent design of ICs that can be security threats and exploited by knowledgeable attackers. Again, the training procedure's noise pattern and train dataset of the ML model, which can be accessed by the adversaries, act as a *hidden backdoor* for the attacker [2]. The goal of this backdoor creation is to perform targeted or random misclassifications toward the ML algorithm, as shown in Figure 3.5.

3.4 Adversarial Examples

Adversarial examples are malicious inputs created by the adversary to deceive the neural network, resulting in misclassification toward the output label. In this case, the *assumption* is adversarial sample inputs should be indistinguishable to the human eye [53, 56, 76], which causes the ML model to fail to identify the perturbation of the image. Goodfellow et al. [56] introduced one of the most famous adversarial examples, as shown below in Figure 3.8.

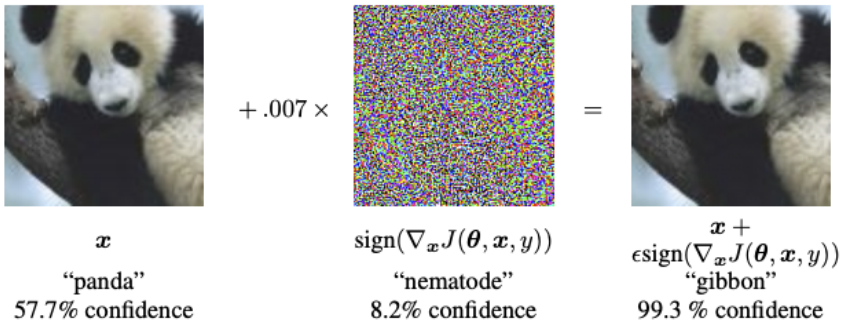


Figure 3.8: The adversary adds small perturbations (noises) to the original image "panda," which results in the ML model labeling this image as a "gibbon" with high confidence. Here error parameter ϵ of .007. (Source of Image: [56])

Figure 3.9 shows the most common attacks generation process towards the neural network with the adversarial examples [53, 58, 77]. The attack technique's fundamental goal is to misclassify the targeted ML model by adding an imperceptible noise into the input data (i.e., image) to create an adversarial example. The methodology of the generation of an adversarial example follows the two-step process, as discussed below and shown in Figure 3.9.

1. In the first step, an attacker chooses one image and introduces *random noise* and an *error parameter* to compute an optimal imperceptible image. The goal is to achieve optimal imperceptibility. The malicious image is considered an adversarial image; otherwise, the error parameter should be updated to generate a new intruder image.

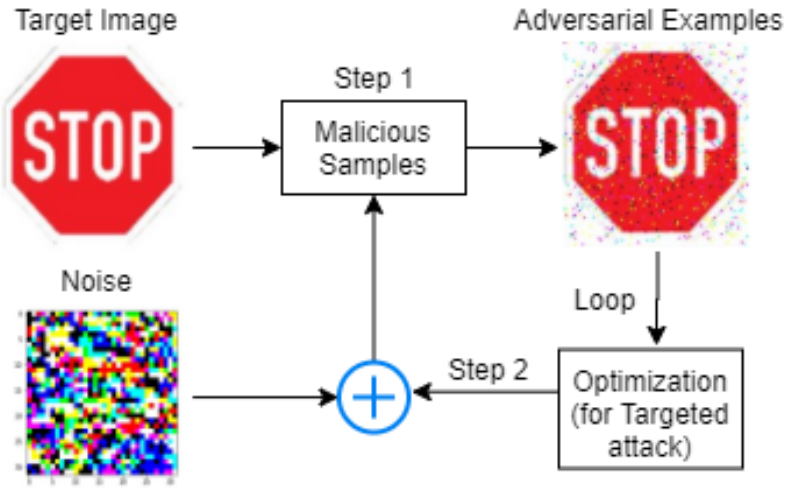


Figure 3.9: The basic methodology of adversarial example generation.

2. In the second step, an attacker chooses the target image and target output class label. Then after completion of *step 1*, the intruded image again feeds together with additional random noise and error parameters in case of targeted misclassification and analyzes imperceptibility of the output adversarial image for defining the optimization goals.

3.4.1 Fast gradient sign method (FGSM)

Goodfellow et al. [56] developed the fast gradient sign method (FGSM) for generating adversarial examples. The FGSM uses the gradient values of the underlying model to create adversarial images. The main goal of the FGSM method is for an input image; it uses the gradients of the loss to the input image for maximizing the loss by misclassifying the true output label [76]. An interesting property is the gradients, which are taken with respect to the input image to create a new image that maximizes the loss. This method tries to determine the contribution of each pixel of the input image to the loss value and adjusts a noise and the error parameter accordingly [56]. Figure 3.8 shows the original image x is manipulated by adding a noise along with a small error parameter ϵ to each pixel.

The Equation (3.2) [78] describes the FGSM method:

$$x^{adv} = x + \epsilon \cdot sign(\nabla_x J(\theta, x, y_{true})) \tag{3.2}$$

where, x^{adv} is the adversarial image, x is the original input image, y is true input label for x, ϵ is error parameter as a multiplier to ensure the perturbations are small, θ is the model parameters (weights), J is loss function or cost function and $\nabla_x J$ is the gradient (i.e., differentiation) of the model's loss function.

The addition and subtraction of the ϵ (error parameter) also depend on whether the gradient sign is positive or negative for each pixel of the input image. That is the sign of the gradient (the vector of the input pixels) takes to determine the direction (negative slope or the positive slope) of the gradient to the global minima. For example, if the gradient value is 0.13 then the sign of the gradient: $sign(0.13) = 1$ which means positive and vice versa. The positive sign ($+1$) of the gradient represents the increase in pixel intensity, which also increases the loss of the output image [76]. Further, due to the error introduced in the ML model, the negative sign (-1) of the gradient represents the decrease in pixel intensity, which increases the loss accordingly [76].

It is also a reliable and fast method to produce an adversarial example. Adding noise and error parameters means altering the input image intentionally so that the attacker succeeds in misclassification of the ML model.

In this project, an adversarial example is generated for a random target on

a neural network using the FGSM technique with a white-box attack method, which is as shown in Algorithm 1 [57, 78]:

Algorithm 1 Fast gradient sign method(FGSM)

1: **procedure** FGSM(x, y_{true}, f_θ)
2: $\widehat{y} \leftarrow f_\theta(x)$
3: $\eta \leftarrow \nabla_x J(\widehat{y}, y_{target})$ ▷ Take gradient w.r.t input image
4: $\eta^* \leftarrow sign(\eta)$ ▷ Computer sign of gradient at each pixel
5: $x^{adv} \leftarrow x + \epsilon.\eta^*$ ▷ Generate adversarial examples
6: **return** x^{adv}
7: **end procedure**

3.4.2 Iterative Fast Gradient Sign Method (I-FGSM)

Kurakin et al. [79] argued that I-FGSM is basically the extension of FGSM which introduces a new error parameter called α. The I-FGSM method takes T iteration to generate a targeted adversarial examples image where the new error parameter $\alpha = \frac{\epsilon}{T}$. Furthermore, T is the iterative gradient steps.Equation (3.3) [78] describes below the I-FGSM method:

$$x_0^{adv} = x,\ x_{t+1}^{adv} = x_t^{adv} + \alpha \cdot sign(\nabla_x J(\theta, x_t^{adv}, y_{target})) \tag{3.3}$$

where, x_t^{adv} is the adversarial image at iteration t, and y_{target} is the target label of the adversarial examples. However, FGSM is one-shot methods with lower success rates than I-FGSM in white-box attacks [79]. Furthermore, I-FGSM can be used as a black-box attack technique. Thus, the chance of success of the attack rate is higher than the basic single-shot methods [79] with the black-box approach, and it is also more effective. This model's drawback is that the iterative methods tend to overfit the ML model and never reach the global minima.

In this project, adversarial example is generated for a specific target on neural network using the I-FGSM technique with black-box attack method which is shown in Algorithm 2 [78, 79]:

Algorithm 2 Iterative fast gradient sign method(I-FGSM)

procedure I-FGSM(x, y_{target}, f_θ)
  $\alpha \leftarrow 1$
  $x_0^{adv} \leftarrow x$
  for $t = 0$ to $T - 1$ **do**
    $\widehat{y}_t \leftarrow f_\theta(x_t^{adv})$
    $\eta \leftarrow \nabla_x J(\widehat{y}_t, y_{target})$ ▷ Take gradient w.r.t input image
    $\eta^* \leftarrow sign(\eta)$ ▷ Computer sign of gradient at each pixel
    $x_{t+1}^{adv} \leftarrow x_t^{adv} + \alpha.\eta^*$ ▷ Generate adversarial examples
  end for
  return x_{t+1}^{adv}
end procedure

3.5 Defensive Techniques

The taxonomy for ML's defensive techniques can be divided into several phases such as training, testing/ inferring, data security, and privacy, which are explained briefly in this section.

Countermeasures in the Training Phase

For the poisoning attack, countermeasures can be ensured by two main components at the training phase: a) the purity of training data such as reject on negative impact(RONI) with data sanitization defending technique and b) another defending process is to improve the robustness of machine learning algorithms, for example, bootstrap aggregating, random subspace method (RSM), principal component analysis(PCA)-based anomaly detection, and distributed feature weights more evenly for the linear classifiers [60].

Countermeasures in Testing/Inferring Phase

In comparison with the training phase defensive techniques, the defensive strategy in the testing/inferring phase mainly focuses on the development of *robust ML algorithms* [60] as a countermeasure.

The second countermeasure is possible with *proactive defense* to alter the data distributions and result in significant deviation for the distribution of training data. This method could decrease to generate the number of false alarms of classifiers against attacks in the testing/inferring phase. This technique can be called the *adversarial training* [60] method. However, the disadvantage of this method is that the ML model's effectiveness depends on the training phase data distributions.

The third countermeasure introduces here is called *defensive distillation*. Papernot et al. [23] analyzed the defensive distillation technique, effectively reducing adversarial samples' sensitivity on state-of-the-art DNNs. Nevertheless, recent research works explain that the defensive technique can not defend the DNNs' algorithms if the adversarial examples are slightly modified by Jacobian-based saliency maps attacks (JSMA) [60].

Furthermore, many researchers proposed some *ensemble* methods [60] as countermeasures to improve the security against ML algorithms. Sengupta et al. [80] proposed an ensemble defensive method framework for combining multiple DNNs to defend against evasion attacks. Furthermore, this framework's disadvantage is that it is not robust and difficult to protect from the transferability of the original sample image to an adversarial example.

Data Privacy and Security

The modern world enters into the era of the significant and high volume of data; therefore, it is required to save such data for a long time due to further research. So, data collectors must focus on data privacy and security from the leakage of individuals' private and sensitive information, such as photos, medical records, and personal information. Therefore, it is obvious to find an effective way of protecting data privacy and security from various attacks [60], for example, IP stealing, eavesdropping, side-channel attack, and reverse engineering (discussed in section 3.3 of Adversarial Threat Model II).

The first countermeasure of the data privacy is *differential privacy* (DP) [81] for the modern classifier ML algorithm (especially for DNNs) [60] via data encryption. The main idea to protect data privacy is to combine or utilize some random noise [60] with the original data in the training phase. DP achieves the following advantages: first, the DP model does not need to think about the attacker's knowledge because it is assumed that an attacker gains the maximal knowledge about the data, and second, the mathematical foundation of the DP model algorithm is solid [60] and complex enough that can't solve in polynomial time with great effort to break it.

The second countermeasure is *homomorphic encryption* (HE) [60] which is another data encryption technique to protect data privacy. HE can decrypt the encrypted data directly without using private keys. Further, HE can use for protecting data privacy and security in a multi-party calculation environment [60] which makes it suitable for deploying in cloud environments.

Chapter 4

Methodology

This chapter describes step by step methodology for accomplishing the thesis goal (discussed in section 1.2) to collect the dataset, data pre-processing, and splitting the dataset. Then, the original Alexnet model is customized and optimized to build a robust ML model architecture. Further, the adversarial threat (such as white-box, black-box attack) model and the algorithm for developing adversarial examples (such as FGSM, I-FGSM method) are also discussed in this chapter.

Research Steps

Based on the security threats/vulnerabilities is discussed in section 3.2 and section 3.3, the following research challenges are addressed for developing secure/robust ML-based systems in this project:

- First important step of this research is to design an algorithm model that is available for building robust CNNs for allowing inherently resilient effects of reliability threats.
- Then, it is necessary to check the robust ML training. It is accomplished by using two categories of datasets that will be collected and downloaded from the GTSRB webpages [15, 16]. The first kind of dataset is GTSRB 2017, which is used for training, validation, and testing a CNN model. However, it is introduced GTSRB 2019 for analyzing and evaluating to generalize an already trained CNN model. The primary approach focuses on achieving high accuracy and reliability for specific datasets and unknown adversarial perturbations.
- Further, to detect the computational cost and remove significant challenges on performance to improve security optimization, lower overhead, and larger performance generalization towards machine learning by using a low number of parameters, and fine-tune of hyperparameters are utilized.

- Afterwards, two adversary models from the perspective of an attacker, such as white-box (using FGSM technique) and black-box (using I-FGSM technique) attack models, are designed.
- Furthermore, an argument is established with a well-defined security assessment standard on the machine learning model (such as AlexNet) for decision systems in adversarial examples environments.
- Finally, datasets are prepared to measure image quality using SSIM and MSE techniques for both targeted (black-box) and untargeted (white-box) attack.

Design Workflow CNN-Based System

The construction of a robust CNN network is not a trivial thing. Therefore, significant research work is going on in this area. In this project, the Alexnet model is constructed as a CNN architecture, and it is called the GTSAlexNet. During the CNN architecture build-up process, several works of literature and books have been studied to gather knowledge in the field of the machine learning algorithm and ML attack technique (i.e., white box and black box attacks technique). The attack techniques are created for image prediction at the time of inference by applying an adversarial example. Then four datasets were prepared for analyzing results based on two attack methods as per the adversary threat model. Figure 4.1 shows a short overview of the project methodology.

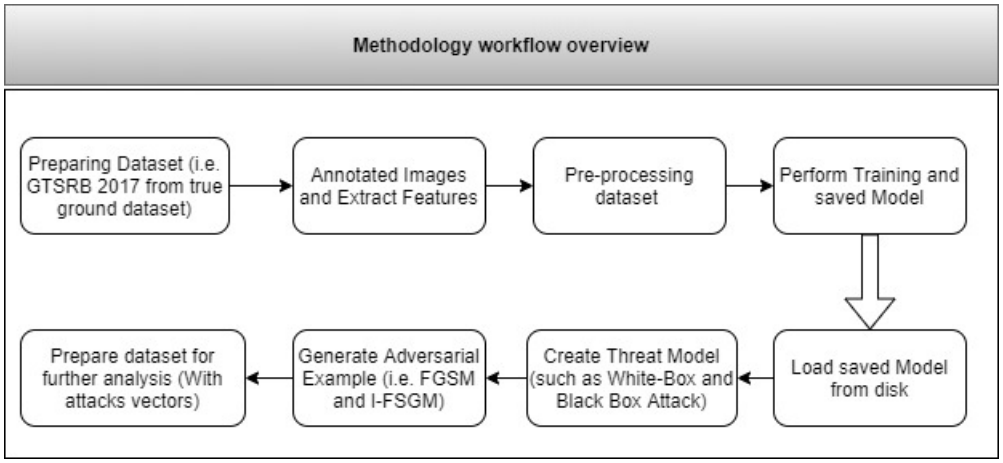


Figure 4.1: Brief overview of the project's Methodology.

4.1 Collection and dataset Preparation

Data gathering and dataset preparation with the extracted features are the first steps of a workflow, as shown in Figure 2.1(stage 1) and 4.1. There are several datasets of traffic signs [82] available publicly, such as the United States, Belgium, Germany, Croatia, Sweden, Italy, and China. However, this project focuses on Germany's traffic sign dataset called *German Traffic Sign Recognition Benchmark* (GTSRB). The main reason for focusing on the mentioned dataset, it can be used without loss of generality since many of these traffic signs are universal. The GTSRB traffic signs provide including variations in shape, color, pictograms or text, and different classes [16]. Hence, traffic signs recognition is challenging for the autonomous car but not for the human. This dataset is highly accepted as well as used them for public held competition.

Furthermore, the GSTRB dataset is tested by scientists from different fields, and it is currently contributing to their results massively. Recently, the significantly accepted results are published by GTSRB [16] website. The state-of-the-art methodologies can be recognized for traffic sign classification with the *overview of ranking* in the website's results. Additionally, the GTSRB dataset extracted from 1-second video sequences [82] and traffic sign samples contained different resolutions and image distortions. In particular, the dataset represents here has to cope with variations in visual appearances due to weather conditions, rotations, partial occlusions, and illumination changes [16].

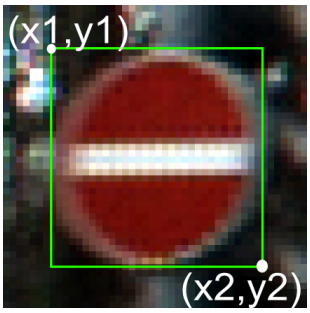
Annotation and Feature Extraction

Stallkamp et al. presented, the actual dataset was created by NISYS[1] from the *ground truth*[2] video (10 hours of length) [83]. This video had been captured on different roads along with the different types of road signs. The NISYS has used the advance development and analysis framework (ADAF) software tool for annotation the image and extracting the feature from the captured video, as shown in Figure 4.2(a) [16]. However, in this project, only pre-annotated data with extracted features [15, 16] in CSV files have been used, as shown in Figure 4.2(b). Moreover, annotated featured files contain eight categories of information: (i) *Filename* - the name of the corresponding image file, (ii) *Width* - the image Width, (iii) *Height* - the image Height, (iv) *ROI.*x1 - the top-left corner of bounding box in *X*-coordinate of a traffic sign, (v) *ROI.*y1 - the top-left corner of the bounding box in *Y*-coordinate of a traffic sign, (vi) *ROI.*x2 - the bottom-right corner of the bounding box in *X*-coordinate of a traffic sign, (vii) *ROI.*y2 - the bottom-right corner of the bounding box in *Y*-coordinate of the traffic sign and

[1] NISYS GmbH

[2] Approx. 10 hours of video of physical real-world traffic sign images were recorded and captured, while driving by a vehicle (such as a car) on different road types and passing the traffic sign in Germany during the daytime.

last but not the least, and (viii) *ClassId* - the class (name of the traffic sign) label assigned for specific sample (image) data. The feature, ClassId, is only available with the training annotation data as additional information for training the CNN (GTSAlexNet).



(a) Annotation format with ADAF tool(Source of Image: [16])

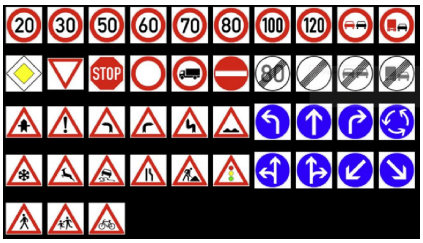
1	Filename;Width;Height;Roi.X1;Roi.Y1;Roi.X2;Roi.Y2
2	00000.ppm;53;54;6;5;48;49
3	00001.ppm;42;45;5;5;36;40
4	00002.ppm;48;52;6;6;43;47
5	00003.ppm;27;29;5;5;22;24
6	00004.ppm;60;57;5;5;55;52
7	00005.ppm;52;56;5;5;47;51
8	00006.ppm;147;130;12;12;135;119
9	00007.ppm;32;33;5;5;26;28
10	00008.ppm;45;50;6;5;40;45

(b) Extracted Image in CSV format

Figure 4.2: The annotations were created with ADAF software tools by Nisys GmbH

Data Pre-processing

Figure 2.1(stage 2) and 4.1 show, pre-processing of data and feature selection will be second step to model a ML algorithm. The GSTRB dataset has 43 traffic signs (details are shown in Appendix A Table A.1) as multi-class classification problems, which are used in this study project to build a model the GTSAlexNet(CNN) algorithm. The annotation is semi-automatic, and the ground-truth data is more reliable. A total of 51, 899 of life-like images have to split into the training set, the validation set, and the training set images.



(a) Original German Traffic Signs (Source of Image: [83])

(b) Ground-truth generated traffic signs

Figure 4.3: Representation of the GTSRB pre-processed 43 traffic sign classes dataset.

Each raw RGB images contain one traffic sign, and the sample image size varies from 15×15 to 250×250 pixels. For this project, all the sample images are down-sampled or up-sampled to 32×32 pixels in the pre-processing stage, as shown in Figure 4.3(a [83], b). The dataset is structured as follows: three pickled files contain all information for training, validation, and test set for individual images tracks, which are initially stored in Portable Pixmap, P6(PPM) format [16]. The assigned 43 class label for image annotations provided in additional CSV files to train and test the dataset.

Splitting Dataset

The GTSAlexNet model generates $51,799$ images of the GTSRB consist of pixel size 32×32 (RGB). The dataset is divided into three compositions as $67\%-8\%-25\%$, as shown in Figure 4.4. In particular, 75% of the data is used for training and validation. Further, the rest portion of the data is used for testing purposes. The training dataset was split 67% for train and $8(=75-67)\%$ for using the iteratively trained and validated on these different sets for selecting the best-created model. From the multiple ways to do iterative model creation, it is commonly used as *cross-validation*[3] [84] (such as K-fold). The intuition is to use 10-*fold Cross-Validation* from the training set (75%) to generate multiple splits of the training and validation sets. Cross-validation is used to avoid overfitting [85] and is getting a more and more accurate model with K-*fold(such as* 10-*fold) cross-validation.*

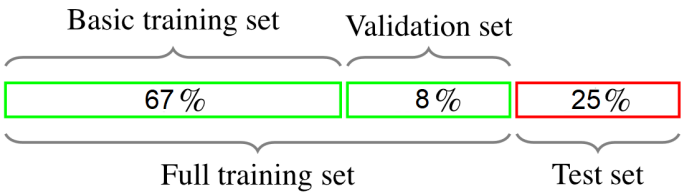


Figure 4.4: Datasets splitting Output into three categories.(Image based on: [83])

The dataset divided into a training dataset with $34,799$ images, a validation dataset with $4,410$ images, and a testing dataset with $12,630$ images of German traffic signs in 43 classes.

The dataset is randomly shuffled and split into three subsets of training, validation, and testing datasets in the first step. Moreover, the shuffling is performed

[3] **Cross-validation** is to split a training dataset into training and validation dataset, and it is applied to more subsets for creating a better accurate model. Meaning, it can be split the training data into k subsets (or fold), where a model will train on $k-1$ one of those subsets and hold one(such as last) subset for validation. This iteration process will able to do it for each of the subsets. Then it will average the model against each of the folds (subsets) and then finalize one model.

to maintain a rational class distribution for each class labels in three individual datasets. Furthermore, the full training datasets (training and validation datasets), as shown in Figure 4.4 in green color, have filled with an extra feature of traffic sign class label, called ClassID. For the testing dataset, in contrast to the full training dataset, the class label is not available. The aforementioned *class distribution* ensured and focused on the clear separation of the datasets.

4.2 GTSAlexNet Model Architecture

Description of vulnerability and threat in the original AlexNet CNN model can be found in the *literature study* section 3.2. In this study, the inner architecture of AlexNet has been modified, called GTSAlexNet architecture, to strengthen the model's accuracy and reliability. Further, AlexNet (discussed in section 2.4) and GTSAlexNet differ in the number of paraments(weights), neurons, CNN layer's architecture (such as batch normalization), and the number of images are used to train a model, which will be described in more details with calculation in subsequent sections of this chapter. Moreover, the visible difference of model architecture and metadata analysis between AlexNet and GTSAlexNet are discussed in the *results and evaluations* section 5.4.

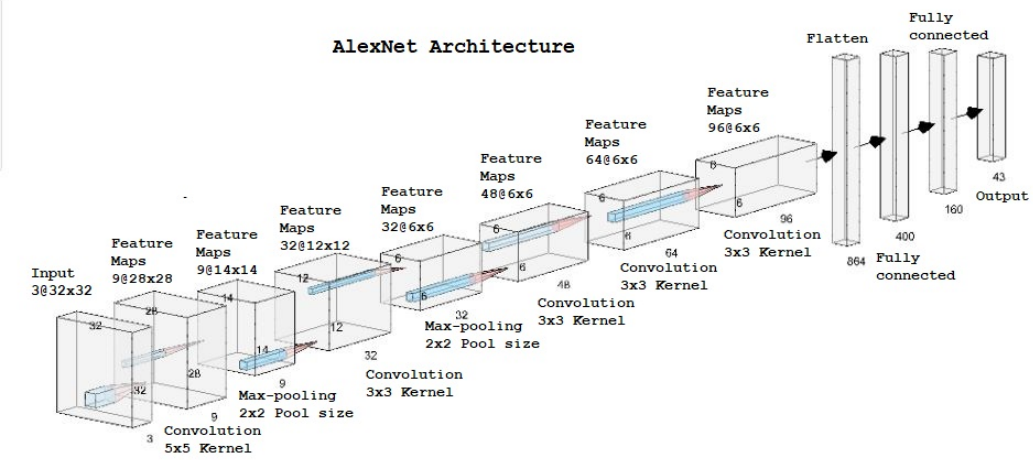


Figure 4.5: An illustration GTSAlexNetarchitecture presenting its five convolution and three fully connected layers.

The proposed model CNN is also combined with several layers: convolutional, batch normalization, max-pooling, fully connected, and dropout layers, as shown in Figure 4.5. These layers are tuned with various hyperparameters in such a way so the optimized neural network can minimize the error rate of the misclassification over the GTSRB training dataset and provide high accuracy during

inference/prediction of output class labels.

The goal of the GTSAlexNet model to resist the adversarial attack and obtain true robustness (such as reliability and protection) and high accuracy. The *assumption* is to maximize the inner parameter (weights) optimization [86] by squeezing the input image [87].

Thus, when CNN is trained with the raw input image, it will train as robustness and image augmentation based on malicious data.

Details of GTSAlexNet Model

The $2D$ convolutional layers take 32×32 input images and then pass through the filters. The size of the convolutional network is calculated as per Equation (2.3) (discussed in section 2.1). The filters (kernels) are fine-tuned to map the input image with the next output layer. Furthermore, the filters are also selected depends on how many different patterns will be the best fit in the output feature map, such as detect the edge, blur, and sharpen of the input traffic sign images. The stride is also selected based on the optimum number of pixels representing the actual traffic sign with high accuracy. The padding "same" (discussed in section 2.1) is used due to the necessity of parameters are properly fit into the feature maps. The ReLU activation function is used for all the convolutional layers. It provides the best performance during neural network computational cost, and fewer neurons are fired. Moreover, the ReLU introduces non-linearity in the neural network, which helps classify traffic signs with high accuracy. The output tensor's size for the convolutional layers is calculated with Equation (2.11).

Further, the second layer is kept the batch normalization (BN) layer to remove the vanishing gradient problems, and all the pixel values will be normalized for input traffic sign images. The smoothing factor (ϵ) for BN is 10^{-5} (discussed in section 2.1).

In this project, the pooling layer is chosen as max-pooling due to CNN takes images as input, and the max-pooling layer will give better performance for the computer vision. Further, the max-pooling layer is used to downsample the input traffic sign images. Moreover, the "valid" padding parameter is applied to keep consistency with the filter into the images. The size of the output tensor for the max-pooling layers is calculated with Equation (2.12).

In the GTSAlexNet model, the fully connected (FC) layer is used, followed by one dropout layer. The FC layers will support the output layer to predict the accurate class label. Moreover, the dropout layer will boost the prediction accuracy and eliminate the over-fitting problem for the state-of-the-art neural network. The size of the output tensor for the FC layers is calculated by multiplying the height, width, and channels ($height \times width \times channels$) of the input traffic sign images.

The number of Parameters and tensor sizes in GTSAlexNet are summarized in Table 4.1. The total number of parameters in GTSAlexNet is the sum of all parameters (weights) in the one batch normalization (BN) layer, five convolutions ($conv$) layers, and three FC layers. It comes out to a total of $517,403$ parameter.

The number of parameters is calculated with Equation (2.14) for the *conv layer*. For example, the number of parameters for the first conv layer is $(5 \times 5 \times 3 \times 9 + 9)$ total of 684. However, there are no parameters for the max-pooling layer (discussed in section 2.4.2). Similarly, there is no backward propagation involved in this layer.

The number of parameters is calculated for the BN layer is equal to 36 $(= 4 \times 9)$, where 4 is the number of new parameters (discussed in section 2.1) and 9 is the number of filters (kernels) size.

Furthermore, The number of parameters is calculated for the FC layers with Equation (2.15) and (2.16) consecutively. In the first example, the number of parameters for the first FC layer connected from a *conv layer* is $(3 \times 3 \times 96 \times 400 + 400)$ total of 346000, using Equation (2.15). In another example, the number of parameters for the second FC layer connected from the previous FC layer is $(400 \times 160 + 160)$ total of 64160, using Equation (2.16).

Table 4.1: GTSAlexNet CNN - Architecture Details

Layer	**Maps**	**Input**	**Kernel size**	**Stride**	**Padding**	**Activation**	**Parameters #**
Input	3	32x32	-		-	-	0
Convolution	9	32x32	5x5	1	-	ReLU	684
BN	9	28x28	0	0	-	-	36
Max Pooling	9	14x14	2x2	1	Valid	-	0
Convolution	32	12x12	3x3	1	Valid	ReLU	2624
Max Pooling	32	6x6	2x2	1	Valid	-	0
Convolution	48	6x6	3x3	1	Same	ReLU	13872
Convolution	64	6x6	3x3	1	Same	ReLU	27712
Convolution	96	6x6	3x3	1	Same	ReLU	55392
Max Pooling	96	3x3	2x2	1	Valid	-	0
Flatten	0	864	0	0	-	-	0
FC	0	400	0	0	-	ReLU	346000
Dropout	0	400	0	0	-	-	0
FC	0	160	0	0	-	ReLU	64160
Dropout	0	160	0	0	-	-	0
output	0	43	0	0	-	Softmax	6923
			Total=				517403

Hyperparameter Tuning

The tuning of the Hyperparameters for the GTSAlexNet model was the most challenging due to the complex overfitting problem of the CNN model, which can arise on the test set. For example, on the training dataset, the error is 5% for a neural network algorithm with a group of hyperparameter values whereas, the selected hyperparameters produce the unexpected 15% errors on the testing

dataset.

In this project, the hyperparameters are selected for the GTSAlexNet model carefully. For improving accuracy, two-step cautions are being utilized. The first one is a widespread solution for validation datasets to evaluate the trained CNN model. The second one is to use the new datasets, i.e., GTSRB 2019, for testing the CNN model. More concretely, the GTSAlexNet model is built on GTSRB 2017 datasets. On the other hand, it is tested with GTSRB 2019 datasets.

Therefore, the overfitting problem is measured by the generalization to reduce the error on the test dataset. Moreover, the adaptation of the hyperparameters will produce the best model with a high accuracy rate. It means that the GTSAlexNet model can perform the best accurate prediction on new datasets, as well.

Table 4.2 describes the summary of GTSAlexNet architecture's hyperparameters, which are used to fine-tuning the neural network for creating a robust model.

Table 4.2: Summarizes the typical architecture of GTSAlexNet model with hyperparameters

Hyperparameter	**Typical Value**
# input neurons	One per input feature and total of 3072 ($32 \times 32 \times 3$)
# hidden layers	Five convolutional, one batch normalization, three max-pooling, three fully connected and two dropout layers
# neurons in hidden layers	Total of 13664 number of neurons in $conv$ and FC layers (is calculated by Equation (2.3) and (2.13) respectively)
# output neurons	one per prediction dimension for 43 classification labels
Hidden activation	ReLU (discussed in section 2.1)
Output activation	Softmax (discussed in section 2.1)
Loss function	Sparse Categorical Crossentropy (discussed in section 2.3)
Optimizer	Adaptive Moment Estimation (Adam), first momentum ($\beta 1$) value is initialized to 0.9 and the second momentum($\beta 2$) value is initialized to 0.999 (discussed in section 2.3)
Learning rate	0.001 (to find the global minima and optimize weights)
Performance Metrics	Accuracy (discussed in section 2.2 and 2.3)
Batch size	128(small batch size for better classifiers in training time)
Epochs	Training Epochs $= 30$
Regularization	dropout tunable rate $= .50$ (discussed in section 2.1 and 2.3)
Batch Normalization	Smoothing term ϵ of value 10^{-5} (discussed in section 2.1)

4.3 Adversarial Attacks and Their Implementations

In this section, the two attack methodologies will be discussed with the input parameters, techniques, threat models, and then the dataset's preparation to run some further analysis and tests.

4.3.1 White Box Attack and FGSM Technique

In Figure 4.6 , there are three inputs for the white box FGSM attack technique and are defined as follows:

(i) **Epsilons (ϵ)** - A list of epsilon values range[4] from 0.05 to $.40$ with a gap of 0.05 units are used to the run the attack. It is discussed in section 3.4.1, that it is important to use the small error parameter value for ϵ to bring imperceptible change in the input image. However, intuitively it can be expected when the ϵ value is large the noticeable perturbations in the image are also the large as well as the attack is more effective as degrading the neural network model's accuracy.

(ii) **Pretrained Model Information** - for the white box attack, it is assumed that the adversary has a full idea about the pre-trained model, which is GTSAlexNet. For this project, the loss function in the GTSAlexNet model is used *categorical crossentropy*, (e.g., Sparse Categorical Crossentropy).

(iii) **True Label** - the randomly selected true label for each epsilon value is another important input parameter to create the small random perturbations (noises) which can help to misclassify the output label intentionally for the adversary.

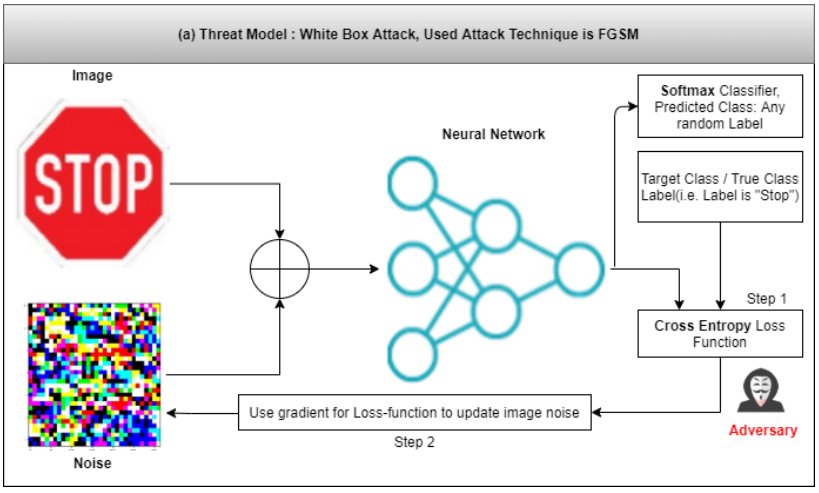


Figure 4.6: Security threat model for white box attack with FGSM technique on GTSAlexNet.

In Equation (3.2), the adversarial examples are created using a TensorFlow built-in function called *gradient*(.), which automatically calculated the gradients

[4] A set for epsilons is, $\epsilon \in \{0, 0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40\}$, total of 9 values.

of the loss with respect to the input image ($\nabla_x J(\theta, x, y_{true})$). Furthermore, another TensorFlow built-in function called $sign(.)$, to get the resulting gradients' sign to create the perturbation of the original inputs. Further, the pixel-wise error parameter ϵ is multiplied with the perturbed image. Finally, the original clean image (x) is added with the newly created perturbed image. Nonetheless, the adversarial image is *clipped* to range $[-1, 1]$ to maintain the original range of the original traffic sign image data. The summary of the whole procedure is described in Algorithm 3.

Algorithm 3 Fast gradient sign method(FGSM) technique for white-box attack

procedure GTS_FGSM(x, y_{true}, f_θ)
  $\epsilon \leftarrow range[0.05, 0.4, 0.05]$ ▷ List (start, final, steps) value
  $\widehat{y} \leftarrow f_\theta(x)$ ▷ Loss function Categorical Crossentropy
  $\eta \leftarrow \nabla_x J(\widehat{y}, y_{target})$ ▷ Take gradient w.r.t input image
  $\eta^* \leftarrow sign(\eta)$ ▷ Computer sign of gradient at each pixel
  $x^{'} \leftarrow x + \epsilon.\eta^*$ ▷ Generate adversarial examples
  $x^{adv} \leftarrow clip(x^{'})$ ▷ Adding clipping to maintain $[-1, 1]$ range
  return x^{adv}
end procedure

The test FGSM attack is run on randomly chosen 337 images of 43 number of classes from GTSRB 2019 dataset with a total of $3033 (= 337 \times 9)$ samples data. Further, a dataset was prepared for various features such as epsilon value, actual label, adversarial label, accuracy, execution time (*wall time*) and perturb image: yes or no. for further analysis about the behavior of the attack (the detailed dataset is in Appendix B Figure B.1).

Moreover, another dataset has been prepared with *structural similarity index* (SSIM) [88] and mean squared error (MSE) [89] to measure the image quality and noticeable difference between the original image and perturb image. The important features of the dataset are epsilon value, actual label, adversarial label, SSIM and MSE value of both original and perturb image [90] (the detailed dataset is presented in Appendix B Figure B.2).

4.3.2 Black Box Attack and I-FGSM Technique

In Figure 4.7, there are three inputs for the black-box I-FGSM attack technique and are defined as follows:

(i) **Epsilons** (ϵ) - A list of epsilon values range from 0.01 to 0.6 with a step of 0.001 units are used for the run of the attack. As discussed in section 3.4.2, it is important to take small steps for small error parameter value ϵ and measure the imperceptible change is approved in the input image

for the targeted attack. Although there is a possibility of overfitting with the I-FGSM technique, it is a very effective attack with less time-consuming.

(ii) **Pretrained Model Information** - for the black-box attack, it is assumed that the adversary has no idea about the pre-trained GTSAlexNet model. In this project, the neural network model's loss function is used *mean squared error* to determine the gradient loss of the softmax layer.

(iii) **Targeted Label** - one targeted label is selected for adversarial image and trying to add perturbations (noises) which can help to misclassify the original image to the targeted output label.

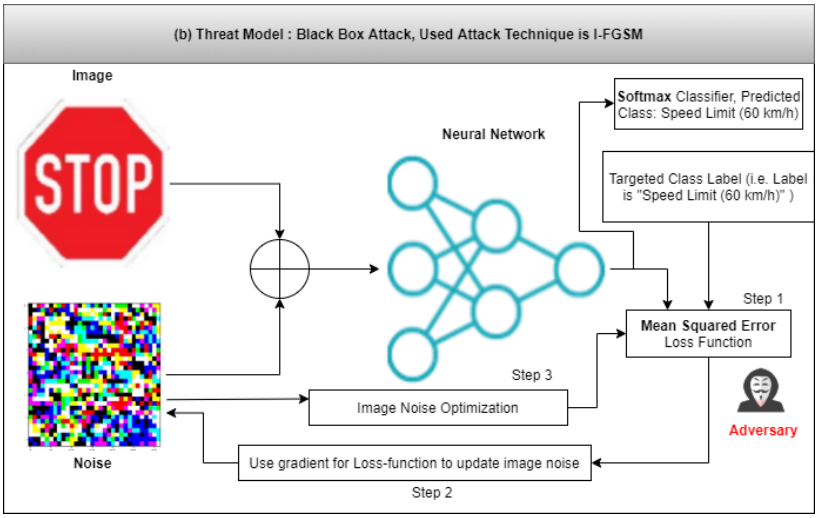


Figure 4.7: Security threat model for black box attack with I-FGSM technique on GTSAlexNet.

In Figure 3.6, I-FGSM creates adversarial examples in the multiple steps process as it is an iterative process.

1. First, one is to select a targeted traffic sign image, then to calculate gradients of the loss with respect to the loss function (*mean squared error*) and input image $(\nabla_x J(\theta, x'_t, y_{target}))$.
2. Further, using the $sign(.)$ function on the resulting gradients and multiplying the pixel-wise error parameter ϵ with the perturbed image.
3. Finally, the original clean image (x) is *subtracted* with the newly created perturbed image.
4. Further, the adversarial image is *clipped* to range $[-1, 1]$ to maintain the original range of the targeted traffic sign image data and

5. Both the labels, predicted label by the model and targeted input label, are *compared*

Suppose the predicted label by the model does not match with the targeted input label. In that case, the noise (the most recent generated perturbed image) is optimized again with a new ϵ parameter. Therefore, the I-FGSM technique is repeated several times until the targeted adversarial example is generated. The summary of the whole procedure is described in Algorithm 4.

Algorithm 4 Iterative fast gradient sign method(I-FGSM) technique for black-box attack

procedure GTS_I-FGSM(x, y_{target}, f_θ)
  $\alpha \leftarrow range[0.01, 0.6, 0.001]$ ▷ List (start, final, steps) value
  $x_0^{adv} \leftarrow x$
  for $t = 0$ `to` $T - 1$ **do**
    $\widehat{y}_t \leftarrow f_\theta(x_t^{adv})$ ▷ Loss function Mean Squared Error
    $\eta \leftarrow \nabla_x J(\widehat{y}_t, y_{target})$ ▷ Take gradient w.r.t input image
    $\eta^* \leftarrow sign(\eta)$ ▷ Computer sign of gradient at each pixel
    $x'_{t+1} \leftarrow x_t^{adv} - \alpha.\eta^*$ ▷ Generate adversarial examples
    $x_{t+1}^{adv} \leftarrow clip(x'_{t+1})$ ▷ Adding clipping to maintain $[-1, 1]$ range
    $y^* \leftarrow predict(x_{t+1}^{adv})$
    if $y^* = y_{target}$ **then**
      Break
    end if
  end for
  return x_{t+1}^{adv}
end procedure

The test I-FGSM attack is run on chosen 23 target images from GTSRB 2019 dataset with total $46 (= 46x2)$ samples data for especially *two targeted labels* (speed limit (100km/h) and speed limit (120km/h)). Then a dataset was prepared for the same features as described for the FGSM attack dataset for further analysis of the behavior of the targeted attack (see the detailed dataset in Appendix B Figure B.3).

Further, another dataset has been prepared with the SSIM and MSE to measure the image quality and noticeable difference between the original and targeted perturb images. The features of the dataset are also as same as described toward the FGSM attack method dataset (see the detailed dataset in Appendix B Figure B.4).

Chapter 5

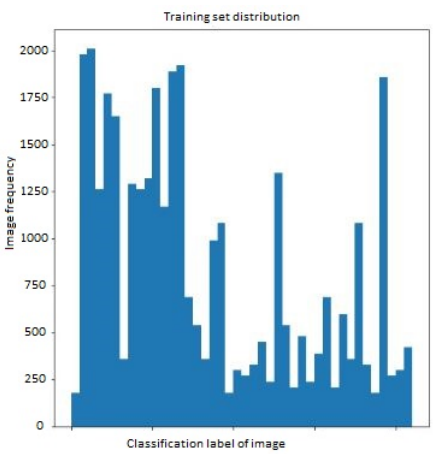
Results and Evaluations

This chapter describes the results and analyses of the neural network algorithms and evaluates the newly built architecture's performance with the existing model. Further, it is also discussed here the prediction accuracy with the sample dataset (GTSRB 2019). This chapter also evaluates the influence of noise (epsilon) and model robustness for a white-box attack. Furthermore, there is an evaluation for human eyes to notice perturbation in the image. Finally, two image quality measurement techniques such as SSIM and MSE, are discussed for white-box and black-box attacks.

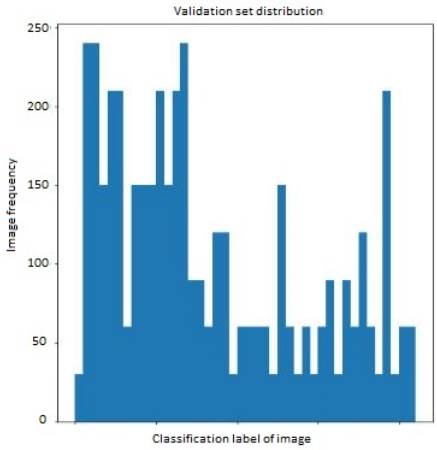
All the tests and analyses of this project were performed to find a well-defined security assessment standard. However, the neural network has a weak defense technique against adversarial samples (e.g., inputted traffic sign image with noise). Furthermore, the new adversary model can create new challenges in predicting and identifying samples data of malicious samples due to difficulties in the ML design. All the neural network framework development, training, testing, and experiment results were run on the *google colab* cloud environment with Intel Xeon 2 cores 2.3 GHz processors, 12.6 GB RAM, and Tesla $K80$ GPUs with $12GB$ GDDR5 VRAM, having 2496 CUDA cores [91].

5.1 Dataset Distribution Structure

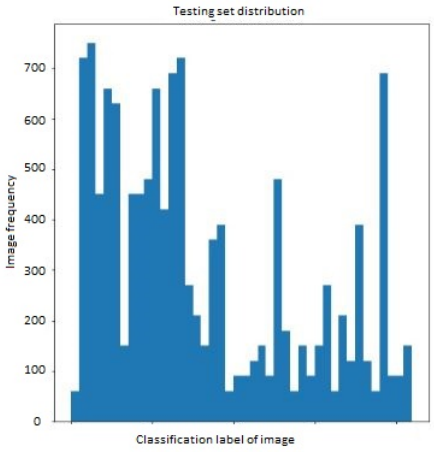
The dataset was prepared with the ratio of 67% training images, 8% validation images, and 25% testing images. That is, a total of 75% of data is used for training and the remaining 25% of data is used for testing. Figure 5.1 shows the distribution of GTSRB dataset after shuffling the whole dataset.



(a) Training dataset distribution



(b) Validation dataset distribution



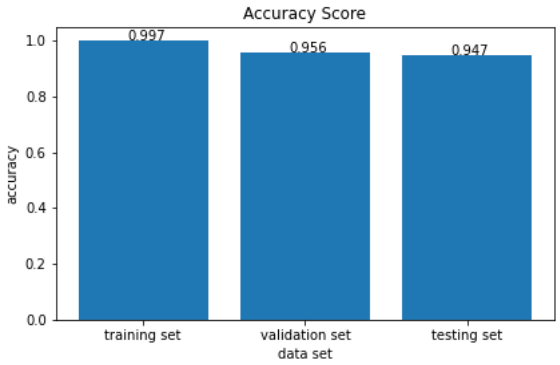
(c) Testing dataset distribution

Figure 5.1: A histogram representation of the training, validation and testing dataset.

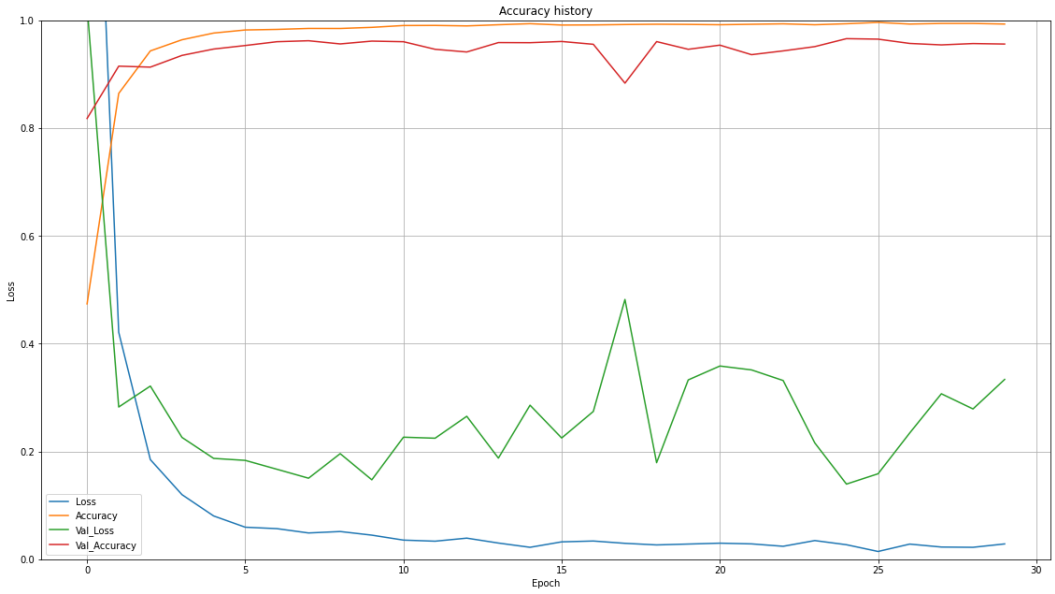
5.2 Training and Evaluating the Model

Figure 5.2(a) demonstrates the model accuracy score of training dataset, validation dataset, and testing dataset. In this case, the model estimated accuracy is 99.70% as the model hyperparameters are well-tuned. In contrast, the accuracy score for the validation dataset is 95.60%. However, the analysis is conducted to measure fluctuation validation and testing accuracy estimation. From this evaluation, it is observed that there is less than 1% (more particularly 0.90%) gap between validation and the testing accuracy due to not tuning any hyperparameters

after the NN model has been built [17]. Nonetheless, the hyperparameters are set and fine-tuned before training the ML model, which are only dependent on the learning model.



(a) Model accuracy score



(b) Learning curves for accuracy and loss measured signs

Figure 5.2: Accuracy and loss representation for GTSAlexNet(CNN) model evaluation.

In Figure 5.2(b) , the accuracy and loss measurement curves are demonstrated. When the dataset is fed into CNN (here, GTSAlexNet), a resultant learning curve history was returned for the list of 30 epochs. It could be analyzed from the curves that the history contained both mean loss and accuracy of the training against validation. The actual loss was measured at the end of each iteration (epoch) with the training and the validation set [17].

In Figure 5.2(b), it can be further analyzed that the accuracy of both the training and validation are steadily increasing. However, the loss for the training and validation are decreasing. Moreover, it represents in Figure 5.2(b), the training curves and validation curves are relatively close to each other except for epoch number 17. There is a sharp decline in the validation curves at epoch 17. Nonetheless, it can be decided that overfitting is not too much. In this particular project, the model performed as likely as on both the validation set and the training set. That is why no *K-fold cross-validation* method is used for this model to minimize the time and resource cost.

However, it reflects in Figure 5.2(b), the validation loss of the model has converged to the lowest point at epoch 24. Therefore, it can be decided that the training should continue to epoch number 24, and the validation accuracy of the model should also reach close to training accuracy at the same point.

After completing the training of the CNN model, the evaluation of the testing dataset gains 94.70% accuracy for this particular project showed in Figure 5.2(a).

5.3 Evaluation of Model Prediction

The evaluation of GTSAlexNet model testing in this thesis is conducted on randomly chosen 430 sample images using GTSRB 2019 dataset. That is, the *ten sample images* were collected randomly from every 43 classes that exist in the GTSRB 2019 dataset. The analysis shows in Table 5.1, that GTSAlexNet model can predict from different classes of images with 78.37% *accuracy*. For example, Figure 5.3(a) shows the actual images, and Figure 5.3(b) shows an incorrectly predicted image (marked in blue border) with the CNN model. In Figure 5.3(a, b), it can be shown that GTSAlexNet model can predict 9 out of 10 images correctly. Hence, it can be argued that the model accuracy is 90(= 9 ÷ 10)% for this particular example.

Table 5.1: Overview of of GTSAlexNet Model Prediction with GTSRB 2019 dataset.

GTSRB Class Label	**Sample Images (of each class)**	**Total sample images**	**Total Correct Prediction (by GTSAlexNet)**	**Testing Accuracy**	**Error Rate**
43	10	430	337	78.37%	21.63%

(a) Actual Image

(b) Predicted Image

Figure 5.3: Correctly predicted image with model.

5.4 Metadata Analysis Between Models Architecture

Table 5.2 shows the metadata analysis between AlexNet model and GTSAlexNet model. AlexNet model was built on ImageNet large-scale visual recognition challenge (ILSVRC[1]) dataset and the CNN architecture depth was *eight layers* (Convolution Layer and dense layers). It achieved the accuracy of 84.7% [48]. The hardware used to train the model was GPU NVIDIA GTX 580 with $3GB$ memory [22].

Whereas, GTSAlexNet model architecture has *nine layers* (Convolution Layer and dense layers) and trained on the GTSRB dataset. It is a finely tuned parameter model that can achieve the 99.70% accuracy for train the model with only half a million parameters (weights). However, the training period was average of 1 minute 17 seconds only using *google colab*, and the GPU model is Tesla $K80$, CUDA cores 2496 with $12GB$ GDDR5 VRAM memory [91].

[1] *ImageNet Large-Scale Visual Recognition Challenge (ILSVRC)* is called the ImageNet dataset, which contains 1.2 million(M) training images, $50,000$ validation images, and $150,000$ testing images belongs to 1000 categories.

Table 5.2: Relative summary metadata analysis of AlexNet and GTSAlexNet.

Architecture Name and Depth	**Datasets and Parameters**	**Training Accuracy**	**Main contribution**
AlexNet (2012) and Depth 8	ILSVRC and 62 million	84.7%	Training set of 1.2 million images has-trained the network for 90 epochs which took *five to six days* on GPUs with memory size 3*GB* of NVIDIA GTX 580 [17, 48]
GTSAlexNet and Depth 9	GTSRB and 0.5 million	99.70%	Training and testing set of 51,799. Sample images are used to train the network for 30 epochs which took average 1 *min* 17 *sec* on *google colab*, GPUs with 12*GB* GDDR5 VRAM Tesla *K*80, having 2496 CUDA cores, Intel CPU with two cores Xeon Processors @ 2.3 GHz and 12.6*GB* RAM [91]

In particular, GTSAlexNet has fewer parameters than AlexNet and is achieved 99.70% with GTSRB 2017 dataset. One crucial point can be noticed from Table 5.2, the AlexNet model was used for 1.20 million images dataset (training & testing) and 62 million parameters(weights) to achieve the best training accuracy results. On the contrary, to achieve the optimal training accuracy, the GTSAlexNet model has been trained with only 0.5 million parameters(weights) and 0.50 million(training & testing). Moreover, training AlexNet requires a lot of computation power than GTSAlexNet, which means more time complexity and resource energy cost involved for training the original AlexNet model [17].

5.5 CNN Algorithm Comparison

Table 5.3 shows a comparative analysis between two individual CNN models. The first ML model is called DeepKnowledge Seville [82] which had 16 *Layers (CL, max-pooling, ReLU and FC layers)*, and it used the GTSRB 2017 dataset. The model author used CNN with 3 Spatial Transformers[2] method and a total of 7.3 million(M) parameters. The model achieved 99.71% accuracy [82].

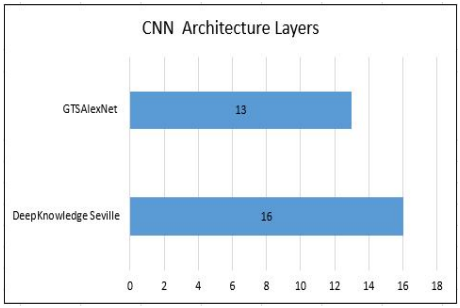
Finally, the method is used in this study is AlexNet and then optimized the model with 13 *layers (BN, CL, max-pooling, and FC layers)*. It has a total of 0.5 million(M) parameters and achieved 99.70% accuracy.

[2] *Spatial Transformers* [92] network is a specialized type of CNN, which contains spatial transformer for input data that attempts to make the neural network spatially invariant.

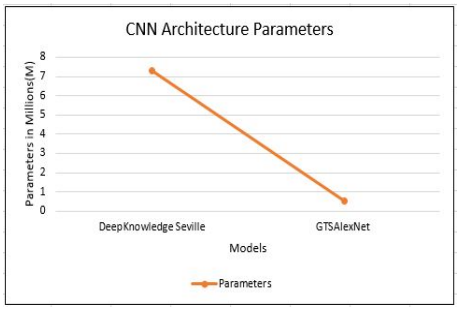
Table 5.3: Parameter and results overview of the GTSRB.

Architecture Name	**Method**	**Datasets**	**CNN Layers**	**Parameters**	**Training Accuracy**
DeepKnowledge Seville	CNN with 3 Spatial Transformers	GTSRB 2017	16	7.3 M	99.71%
GTSAlexNet	AlexNet	GTSRB 2017	13	0.5 M	99.70%

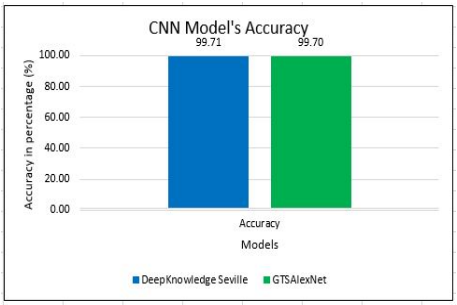
Figure 5.4(a) shows the layers comparison, and Figure 5.4(b) shows the total parameters are used in designing of CNN algorithm architecture. Furthermore, Figure 5.4(c) depicts the representation of the individual model accuracy.



(a) Model architecture layers



(b) Model architecture parameters



(c) Model Accuracy

(d) GTSAlexNet model training wall time

Figure 5.4: CNN algorithm results overview of the GTSRB.

In particular, performing experiments to those are shown in Figure 5.4 for the proposed GTSAlexNet model, and is identified the following key observations:

(i) Figure 5.4(c) shows a trade-off between GTSAlexNet and DeepKnowledge Seville model concerning accuracy is achieved because GTSAlexNet incorporates fine-tune hyperparameters during designing the algorithm and the resizing effects(such as downsampling and upsampling) of images.

(ii) Figure 5.4(d) shows another key observation is that the memory requirement [93, 94] and computation cost [95] for the training of GTSAlexNet model is lower than DeepKnowledge Seville model because of the smooth design of the algorithm with fewer parameters(weights) and neurons. Equation (5.1) shows how CNN's memory consumption can be calculated through the size of a model.

$$M_R(MB) = (N_{param} \times 4 \times 3) \div 1024^2 \tag{5.1}$$

where, M_R is memory (RAM) requirement (for using 32-bit float) to train per image and system of units is megabyte (MB), N_{param} is refer to *number of parameters* of CNN model, 4 is used to work with *float*32 values are converted into byte and 3 is used to have calculated the memory requirements for a forward pass and backward pass could take up to $3x$ memory [96, 97].

Table 5.4: Memory requires of CNN for used parameters.

Architecture Name	N_{param}	M_R **(MB) per epoch**	**Number of epochs**
DeepKnowledge Seville	7.3 M	83.59 MB	21
GTSAlexNet	0.5 M	5.92 MB	30

Table 5.4 shows the memory requirement (M_R) during training for both CNN algorithm (DeepKnowledge Seville and GTSAlexNet). It should be noted here that the M_R is just for one epoch. DeepKnowledge Seville GTSAlexNet achieved accuracy of 99.71% at 21 epochs [82] whereas GTSAlexNet used 30 epochs to achieve 99.70% accuracy.

5.6 White-box and Black-box Attack

Figure 5.5 reports the results of white-box attacks using FGSM technique. This analysis revealed that the misclassification label of the output image had been randomly changed for different error parameters (epsilon value). Further, it is also observed when the epsilon value was increased and added a random noise against the original images. The imperceptibility in the image can be easily recognized with bare eyes.

The above argument represents, it can produce a direct contradiction of existing *assumption* about the creation of the adversarial example (discussed in section 3.4). It might happen due to downsampling or upsampling in the preprocessing of the ground truth captured image (discussed in section 4.1). Furthermore, in most cases, the perturbed image can easily distinguishable to the human eye.

Figure 5.5: Evaluation of white box attack with FGSM method for *Stop* image against *different epsilon value* range from 0.0 to 0.40.

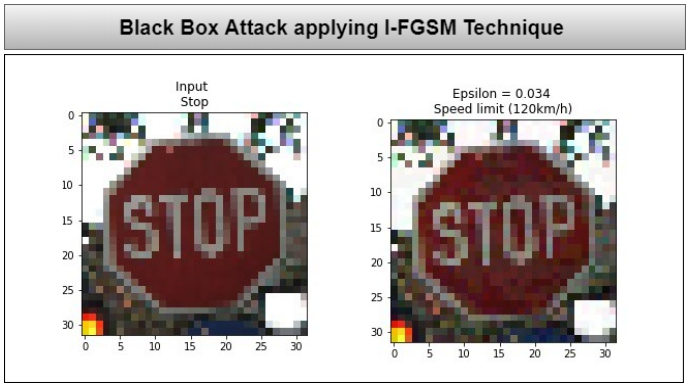


Figure 5.6: Evaluation of black box attack with I-FGSM method for *Stop* image against *epsilon* = 0.034 value misclassified as *Speed limit (120km/h)*.

Figure 5.6 reports the results of black-box attacks using I-FGSM technique. From this analysis, I-FGSM iterative method revealed the targeted misclassification label after multiple iterations. In each iteration, a tiny small portion of epsilon value is added in I-FGSM technique, which generated the desired adversarial example eventually. Further, during the attack generation process, overfitting problem did not arise.

Execution Time

Table 5.5 shows a summary statistical analysis for inference, targeted (I-FGSM), and untargeted (FGSM) attack of execution time (measuring in seconds). It is noticed here that the execution time for the I-FGSM method of black-box attack to generate adversarial examples was more than the single-shot FGSM white-box attack technique, as shown in Table 5.5.

Inference (prediction) execution time of GTSAlexNet model is evaluated with average value 0.034 seconds (s) considering error bar varying from standard deviation $2.6 \times 10^{-3} s(+/-)$, as shown in Figure 5.7. It can be analyzed that, distribution for inference of unknown execution time with respect to mean value within $(3.43 \times 10^{-2}, 3.37 \times 10^{-2})s$ with a 95% confidence interval [98].

Figure 5.7 shows the analysis of the average execution time needed for both targeted and non-targeted attacks with standard deviation $1.320s(+/-)$ and $5.3 \times 10^{-3}\,s(+/-)$ respectively. In Table 5.5, it is observed that the average execution time for the targeted attack is $2.563s$, whereas, for the non-targeted attack, it was only $0.034s$. Furthermore, Table 5.5 shows the minimum execution time for targeted attack and non-targeted attacks were $0.058s$ and $0.031s$, respectively. Whereas, it can be observed the maximum execution time for the non-targeted attack and targeted attack were $0.181s$ and $17.003s$, respectively. Furthermore, the distribution for the unknown execution time of the mean value is within $(3.846, 1.281)s$ for targeted attack and $(3.41 \times 10^{-2}, 3.36 \times 10^{-2})s$ for untargeted attack with a 95% confidence interval [98].

Table 5.5: Statistical inference of execution time(s) for inference, targeted and untargeted attack

Execution Time	Mean	Standard Error	Standard Deviation	Minimum	Maximum	Confidence Level(95.0%)
Inference	$0.034s$	1.4×10^{-4} s	2.6×10^{-3} s	$0.030s$	$0.047s$	2.8×10^{-4} s
Targeted attack	$2.563s$	$0.637s$	$1.320s$	$0.058s$	$17.003s$	$1.283s$
Untargeted attack	$0.034s$	9.0×10^{-5} s	5.3×10^{-3} s	$0.031s$	$0.181s$	1.9×10^{-4} s

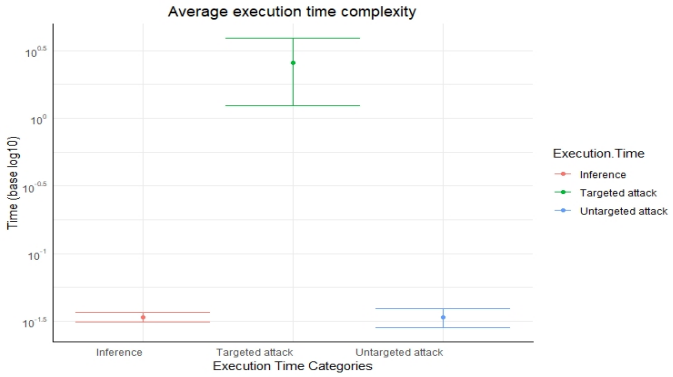


Figure 5.7: Execution time for inference, white-box(untargeted) and black-box(targeted) attack.

Attacker's Goal for CIA Triad

Table 5.6 shows a simple categorization of CIA triad in the perspective of security violation (discussed in section 3.3) [60] and goal of the white box and black attacks of the adversary against GTSAlexNet model based on the threat model, as shown in Figure 4.6 and 4.7 discussed in section 4.3. Furthermore, the evasion attack is included considering the attacker's capability against the testing dataset to misclassify the original image label based on a random or specific target.

Therefore, white-box attack(FGSM) and black-box Attack(I-FGSM) affect toward *integrity* of CNN model, as integrity attack [60] makes sure that the information alteration with respect to *accuracy and authenticity* [67] of GTSAlexNet model. Moreover, a black-box attack (I-FGSM) method is categorized towards *availability attack* [60], for example, DoS attack (discussed in section 3.6) [67] on GTSAlexNet model as I-FGSM is an *iterative* method.

Table 5.6: Categorization of attack based on the threat model of CIA attributes

	Attacker's goal in CIA triad		
Attacker's method or capabilities for evasion attack	Confidentiality or Privacy	Integrity	Availability
White Box Attack using FGSM method	✗	✓	✗
Black Box Attack using I-FGSM method	✗	✓	✓

5.7 Influence of Epsilon on Images

Figure 5.8 show the error parameter (ϵ) for adversarial examples is the set of $\epsilon \in \{0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.40\}$ which will affect the clean and original image samples. Further, when the ϵ value is multiplied with the

gradient value, a tiny change happens in every pixel of the original image, which leads toward misclassification of the output label.

It is also observed that the larger epsilons follow to maximize the loss in the gradient value. Furthermore, it is noticed an upward trend of cumulative value was not linear for the histogram curve. Finally, one important observation is, most of the images (89.33%) are misclassified with $\epsilon <= 0.2$ on the GTSRB dataset.

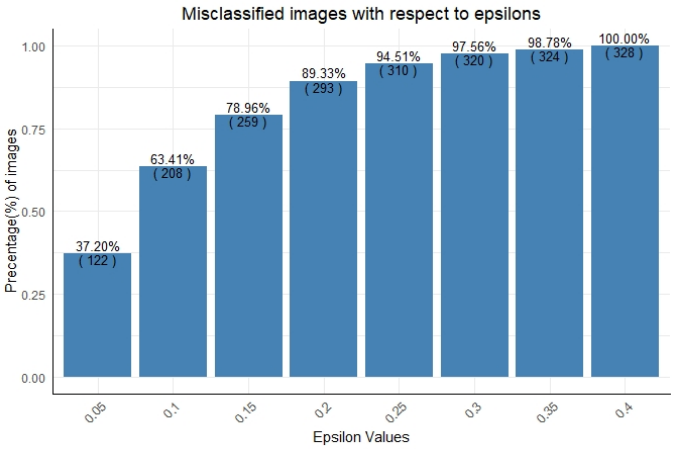


Figure 5.8: Measure epsilon with respect to cumulative image misclassification using white-box(FGSM) attack technique.

5.8 Adversarial Model Influences Accuracy

Figure 5.9 shows that adversarial examples can affect the accuracy of the model GTSAlexNet. It is observed, that the misclassification rate is increased double from $\epsilon = 0.05$ to $\epsilon = 0.15$ whereas the accuracy rate has a sharp decline between $\epsilon = 0.05$ and $\epsilon = 0.15$. For example, at $\epsilon = 0.15$ the misclassified image rate is 76.85%.

The measure of the robustness of the GTSAlexNet model is the ratio of accurate classification percentage on clean images to misclassify (error rate) classification percentage on adversarial images, as shown in Figure 5.9. Further, an increasing or decreasing of this ratio means that there is a reliability gap between the original images and adversarial examples.

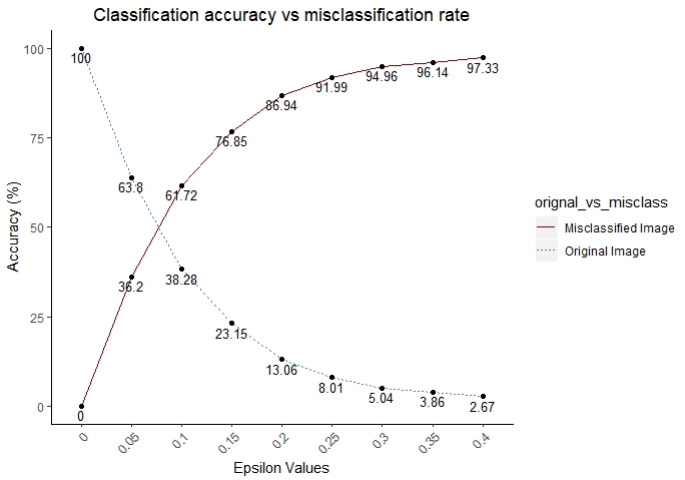


Figure 5.9: The value of epsilon is enough to lead to misclassification.

Furthermore, when evaluated on the original image's accuracy, the trend toward decreasing robustness with increasing ϵ remains with some exceptions at $\epsilon = 0.40$.

Moreover, it is noticed that a human eye can recognize the added imperceptible noise for most of the traffic signs at $\epsilon = 0.10$. For example, the number of observations for different traffic signs with random noise and the subsequent misclassification labels are represented in Figure 5.10.

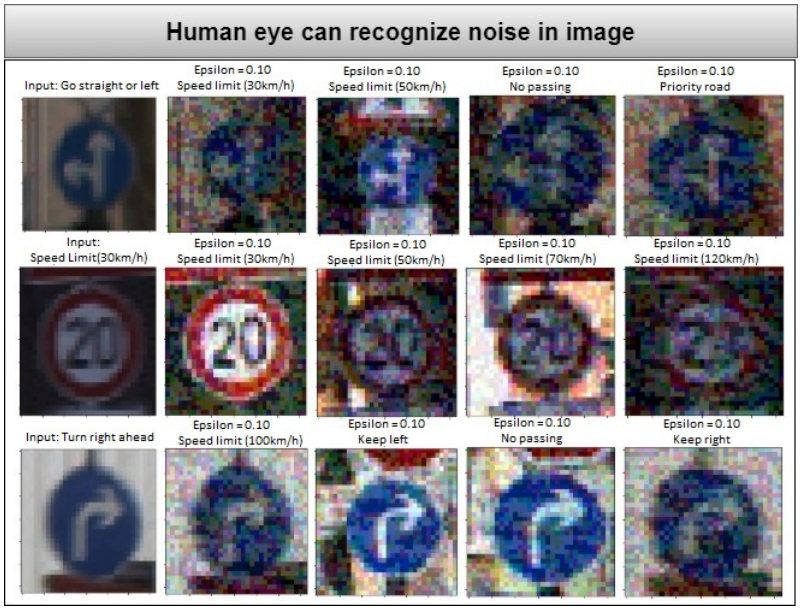
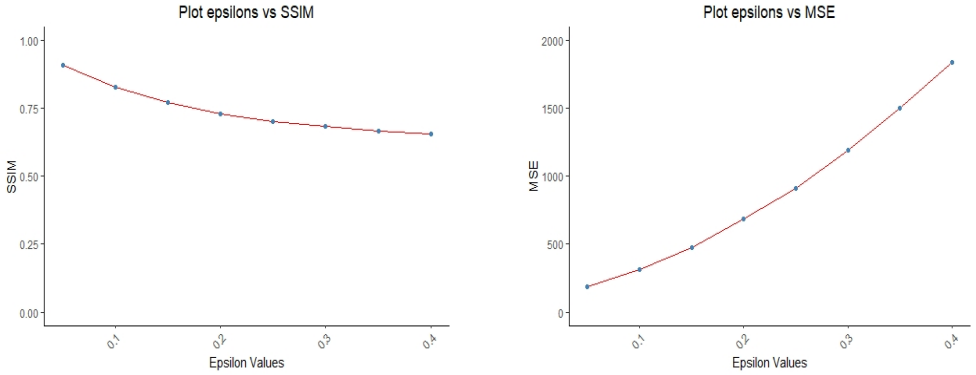


Figure 5.10: For a human eye can notice something is wrong in the image.

5.9 Measures of Image Quality: SSIM vs MSE

SSIM is the image quality estimation technique that perceives the less visible distortions in the quality in the texture of an image [90]. SSIM can analyze a *decrease* in quality images after compression to [89] whether the user can recognize the perceptible noise in it. The SSIM value can scale between -1 and 1, whereas the measurement can be influenced high if the weight value is equal to 1 which indicates *perfect similarity* between two images [90].



(a) Image quality Measurement with SSIM (b) Image quality Measurement with MSE

Figure 5.11: Image quality estimates with SSIM and MSE for FGSM attack.

Figure 5.11(a) shows the sharp downward trend of the image quality with an increasing amount of the error(ϵ) parameters adding into the original image.

Further, there is another image quality estimation technique called MSE. It is the most common image quality measurement estimator [90]. The MSE value of 0 indicates a perfect similarity between the two images. However, if the value of the MSE will *grow* continuously, it implies *less similarity*, and the difference of *pixel intensities* between two images increases.

Figure 5.11(b) shows the rising trend of the image quality with an increasing amount of the ϵ parameters adding into the original image. It means that the variance of the image quality is incorporated more noise with the clean image.

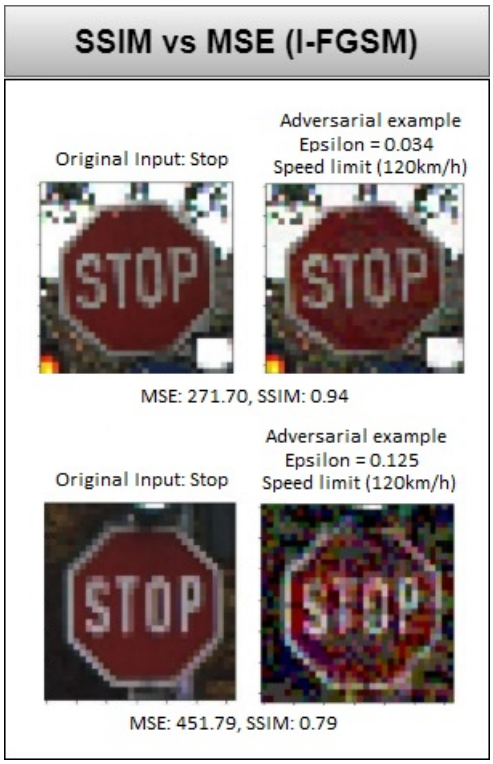


Figure 5.12: Image quality measurement with SSIM and MSE of I-FGSM attack.

Figure 5.12 shows the image quality estimator process for the I-FGSM attack technique. It can clearly analyze when $\epsilon = 0.034$ the difference of pixel intensities between two images is less than when $\epsilon = 0.125$. Further, when SSIM is close to 1 or MSE is close to 0, then the images imply imperceptible difference to the human eye.

Figure 5.13 shows an upward trend measurement of the average misclassified rate of images with respect to SSIM towards how image quality degrades when SSIM decreases and subsequently, misclassification increases. Therefore, the human eye can easily recognize imperceptible noise in images according to a quality metric (SSIM). The misclassification of the neural network model increases due to degradation of image intensity for adding a higher rate of error parameter (ϵ). For example, an approximate 58% increase in misclassification is achieved when 28% drops in quality metric (SSIM).

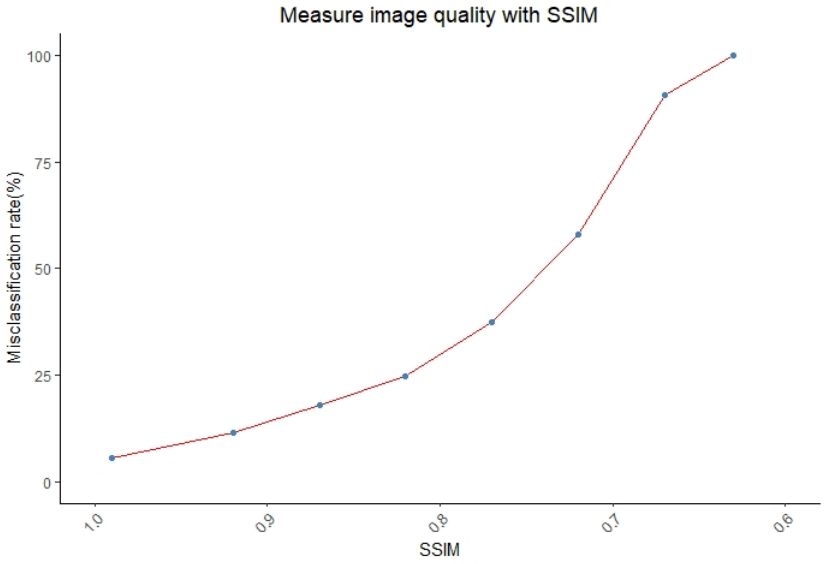


Figure 5.13: Measure Image quality with SSIM to identify noise by human eye.

5.10 Defensive Techniques for GTSAlexNet

First defensive technique, the *robust ML algorithm (GTSAlexNet)* is developed for the testing/inferring phase. The accuracy of GTSAlexNet is achieved to 63.80% against $\epsilon = 0.05$ (discussed in section 5.8). Further, it can be evaluated that the testing accuracy is achieved to 78.37% (discussed in section 5.3) with dataset GTSRB 2019. Therefore, the resilience against *security vulnerabilities and reliability threats* can be described with a general Equation (5.2) as *simple reliability function* $R(.)$ based on average inter-item(accurate classification and misclassification of images) correlation. In this project, Equation (5.2) is named as *accuracy security product* (ASP), which means, it is given a probability that an image will be misclassified rate (discussed in section 5.8) against error parameter (ϵ) considering the testing accuracy(78.37%) of the model to verify the *robustness(reliability)* of GTSAlexNet model.

$$ASP = R(CNN|\epsilon) = e^{-(A_{Test} \times \lambda_{ER|\epsilon})} \tag{5.2}$$

where $R(CNN|\epsilon)$ refers to probability of CNN still work at ϵ, A_{Test} refers to testing accuracy of GTSAlexNet model and $\lambda_{ER|\epsilon}$ refers to misclassification rate(probability) with a given ϵ. Therefore, the reliability of GTSAlexNet against adversarial model $\epsilon = 0.05$ is $ASP = 75.30\%$, where $A_{Test} = 78.37\%$ and $\lambda_{ER|\epsilon} = 36.20\%$ for 430 sample images only. However, the resiliency will be dropped with respect to the increasing amount of ϵ values (discussed in section 5.7 and 5.8).

The second proposed countermeasure is to use a threshold (band limit) parameter to act as a filter to measure the noise and subsequently cut off the noise from the images if the noise will not be less than or greater than the threshold parameter. One possibility for this proposed security measure may be achieved with image size measurement in a kilobyte(KB). Hence, whenever a noise was included with an original image during experiments, then the size of the image (KB) changed as the intensity of a pixel for an image was changed. Therefore, it is clear, for intensity definition, that more computer storage is needed for more colors and more pixels in an image file, and it is generally known that storage is always more costly [99]. Furthermore, another possibility, one can measure SSIM value or MSE value to build a filter based on measuring the image quality will be changed due to the addition of noises into images.

Chapter 6

Conclusion

In this thesis, it is researched that a convolutional neural network (CNN) can be easily deceived during the testing phase with adversarial examples using an evasion attack. Hence, it is obvious to build a robust CNN model with high accuracy as well as reliability. Therefore, this project focuses on designing a robust algorithm model called GTSAlexNet, which can achieve 99.70% training accuracy and 94.70% testing accuracy with dataset GTSRB 2017 dataset. This algorithm is designed to keep security challenges optimum for the white-box and black-box attack's availability and integrity. The newly created CNN mimics the original AlexNet model concerning a change in inner structure for a low number of parameters ($517,403$) and neurons ($13,707$). Because of reducing the more considerable overhead of parameters and highly customization of fine-tune hyperparameters, the CNN performance generalization can be increased. The CNN model's training with google cloud environment takes 1min 17sec for 30 epochs on average.

Additionally, two categories of datasets are used in this research to measure the robustness. The first dataset GTSRB 2017, is used for training. Another dataset, GTSRB 2019, is used to evaluate testing for GTSAlexNet and higher accuracy & reliability is achieved not only for specific datasets but also for unknown dataset due to adversarial perturbations. It is found that the resilience of CNN against security vulnerabilities and reliability threats is 75.30% of noise ($\epsilon = 0.05$).

Further, to simulate an attacker, two adversary threat models of the testing phase (evasion attack) are developed. The first model is developed for a white-box (untargeted) attack using the FGSM technique. The second model is simulated for a black-box (targeted) attack using the I-FGSM technique. In this work, perturbation is generated and measured through gradient value (noise) to distort the original image resulting in an adversarial image for random (indiscriminate) and targeted output label. Moreover, the mean value of execution time distribution is within ($3.85s, 1.28s$) for targeted attack and ($0.03405s, 0.03367s$) for untargeted attack with a 95% confidence interval. From this measurement, it is analyzed that the targeted attack takes more time than the untargeted attack.

Furthermore, in this project, a well-defined argument is established for security

assessment standards on the machine learning model against noise parameter (epsilon) for making decision systems of adversarial examples. Finally, by measuring image quality using SSIM and MSE techniques for both targeted and untargeted attacks, it is revealed that the noise in images can be easily recognized for human eyes when it is gradually increased.

Future Work

Future work can include creating a defense mechanism (filter) for the neural network model, which is suggested in this research. The GTSAlexNet model's robustness can be increased by including and extracting more training data by image augmentation as a proactive defense technique. Further other datasets and neural network algorithms should also be explored. One exciting security violation area can be identified and assessed for decision based deep learning systems of confidentiality and privacy perspective.

Bibliography

[1] M. Fink, Y. Liu, A. Engstle, and S.-A. Schneider, "Deep learning-based multi-scale multi-object detection and classification for autonomous driving," in *Fahrerassistenzsysteme 2018*. Springer, 2019, pp. 233–242.

[2] J. J. Zhang, K. Liu, F. Khalid, M. A. Hanif, S. Rehman, T. Theocharides, A. Artussi, M. Shafique, and S. Garg, "Building robust machine learning systems: Current progress, research challenges, and opportunities," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–4.

[3] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter, "Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition," in *Proceedings of the 2016 acm sigsac conference on computer and communications security*, 2016, pp. 1528–1540.

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

[5] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 9, pp. 2805–2824, 2019.

[6] W. Awad and S. ELseuofi, "Machine learning methods for spam e-mail classification," *International Journal of Computer Science & Information Technology (IJCSIT)*, vol. 3, no. 1, pp. 173–184, 2011.

[7] I. Corona, G. Giacinto, and F. Roli, "Adversarial attacks against intrusion detection systems: Taxonomy, solutions and open issues," *Information Sciences*, vol. 239, pp. 201–225, 2013.

[8] D. Ucci, L. Aniello, and R. Baldoni, "Survey of machine learning techniques for malware analysis," *Computers & Security*, vol. 81, pp. 123–147, 2019.

[9] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, 2017, pp. 506–519.

[10] A. Esteva, A. Robicquet, B. Ramsundar, V. Kuleshov, M. DePristo, K. Chou, C. Cui, G. Corrado, S. Thrun, and J. Dean, "A guide to deep learning in healthcare," *Nature medicine*, vol. 25, no. 1, pp. 24–29, 2019.

[11] B. Biggio, L. Didaci, G. Fumera, and F. Roli, "Poisoning attacks to compromise face templates," in *2013 International Conference on Biometrics (ICB)*. IEEE, 2013, pp. 1–7.

[12] N. Carlini, P. Mishra, T. Vaidya, Y. Zhang, M. Sherr, C. Shields, D. Wagner, and W. Zhou, "Hidden voice commands," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 513–530.

[13] N. Dalvi, P. Domingos, S. Sanghai, and D. Verma, "Adversarial classification," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2004, pp. 99–108.

[14] D. Lowd and C. Meek, "Adversarial learning," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, 2005, pp. 641–647.

[15] "German Traffic Sign Recognition Benchmark (GTSRB)," feb 2017, "https://d17h27t6h515a5.cloudfront.net/topher/2017/February/5898cd6f_traffic-signs-data/traffic-signs-data.zip",last accessed on 01/04/2020.

[16] C. Igel, "German Traffic Sign Recognition Benchmark (GTSRB)," Electronic Research Data Archive (ERDA), may 2019, "http://benchmark.ini.rub.de/index.php?section=gtsrb&subsection=dataset",last accessed on 30/03/2020.

[17] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.

[18] N. C. Thompson, K. Greenewald, K. Lee, and G. F. Manso, "The computational limits of deep learning," *arXiv preprint arXiv:2007.05558*, 2020.

[19] H. H. Aghdam and E. J. Heravi, "Guide to convolutional neural networks," *New York, NY: Springer*, vol. 10, pp. 978–973, 2017.

[20] Y. He, G. Meng, K. Chen, X. Hu, and J. He, "Towards privacy and security of deep learning systems: a survey," *arXiv preprint arXiv:1911.12562*, 2019.

[21] A. Darwish, A. E. Hassanien, and S. Das, "A survey of swarm and evolutionary computing approaches for deep learning," *Artificial Intelligence Review*, vol. 53, no. 3, pp. 1767–1812, 2020.

[22] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artificial Intelligence Review*, pp. 1–62, 2019.

[23] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "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.

[24] "Convolutional neural network," https://en.wikipedia.org/wiki/Convolutional_neural_network, last accessed on 02/07/2020.

[25] K. Ahirwar, "Everything you need to know about neural networks," Mate Labs, nov 2017, "https://hackernoon.com/everything-you-need-to-know-about-neural-networks-8988c3ee4491", last accessed on 02/07/2020.

[26] F. Chollet, *Deep Learning with Python*. MITP-Verlags GmbH & Co. KG Manning Publications Co., 2018.

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

[28] P. BLOEM, "Deep learning," Machine Learning at VU University Amsterdam, mar 2020, "https://mlvu.github.io", last accessed on 13/09/2020.

[29] HADRIENJ, "Deep learning book series · 2.1 scalars vectors matrices and tensors," MissingLink.ai: Neural Network Concepts, mar 2018, "https://hadrienj.github.io/posts/Deep-Learning-Book-Series-2.1-Scalars-Vectors-Matrices-and-Tensors/", last accessed on 19/09/2020.

[30] D. Jeffries, "Learning ai if you suck at math — p4 — tensors illustrated (with cats!)," hackernoon: AI, sep 2019, "https://hackernoon.com/learning-ai-if-you-suck-at-math-p4-tensors-illustrated-with-cats-27f0002c9b32", last accessed on 19/09/2020.

[31] "Scanning 101," AdSell Companies: Team member of AdSell, 2019, "http://www.adsell.com/scanning101.html", last accessed on 13/09/2020.

[32] P. Punyawiwat, "Interns explain CNN," Medium, feb 2018, "https://blog.datawow.io/interns-explain-cnn-8a669d053f8b", last accessed on 07/07/2020.

[33] Prabhu, "Understanding of convolutional neural network (CNN) — deep learning," Medium, mar 2018, "https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148", last accessed on 07/07/2020.

[34] "Convolutional neural network," Wikipedia, the free encyclopedia, sep 2020, "https://en.wikipedia.org/wiki/Convolutional_neural_network#Convolutional_layer", last accessed on 20/09/2020.

[35] A. Handan, "Deep learning: Convolutional neural networks," bawilabs, Medium, apr 2018, "https://labs.bawi.io/deep-learning-convolutional-neural-networks-7992985c9c7b", last accessed on 08/07/2020.

[36] S. Sharma, "Activation functions in neural networks," Towards Data Science, Medium, sep 2017, "https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6", last accessed on 08/07/2020.

[37] Prateek, "Statistics is freaking hard: Wtf is activation function," Towards Data Science, Medium, aug 2017, "https://towardsdatascience.com/statistics-is-freaking-hard-wtf-is-activation-function-df8342cdf292", last accessed on 09/07/2020.

[38] M. Labs, "Secret sauce behind the beauty of deep learning: Beginners guide to activation functions," Medium, aug 2017, "https://towardsdatascience.com/secret-sauce-behind-the-beauty-of-deep-learning-beginners-guide-to-activation-functions-a8e23a57d046", last accessed on 09/07/2020.

[39] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[40] "Fully-connected neural network layer," Gabor Melli's Research Knowledge Base, jun 2020, "https://www.gabormelli.com/RKB/Fully-Connected_Neural_Network_Layer", last accessed on 13/09/2020.

[41] vikashraj luhaniwal, "Forward propagation in neural networks — simplified math and code version," Medium: Towards Data Science, may 2019, "https://towardsdatascience.com/forward-propagation-in-neural-networks-simplified-math-and-code-version-bbcfef6f9250", last accessed on 13/09/2020.

[42] T. Wood, "Backpropagation: What is backpropagation?" Deepai, may 2019, "https://deepai.org/machine-learning-glossary-and-terms/backpropagation", last accessed on 13/09/2020.

[43] SiDdhartha, "Neural networks hyperparameter tuning in tensorflow 2.0," Medium: ML Book Follow Artificial Intelligence, Machine Learning, Deep Learning, Data Science, aug 2019, "https://medium.com/ml-book/neural-networks-hyperparameter-tuning-in-tensorflow-2-0-a7b4e2b574a1", last accessed on 19/09/2020.

[44] S. Verma, "Understanding different loss functions for neural networks," Medium, jan 2019, "https://towardsdatascience.com/understanding-different-loss-functions-for-neural-networks-dd1ed0274718", last accessed on 10/07/2020.

[45] S. Doshi, "Various optimization algorithms for training neural network," Medium, jan 2019, "https://medium.com/@sdoshi579/optimizers-for-training-neural-network-59450d71caf6", last accessed on 10/07/2020.

[46] D. Masters and C. Luschi, "Revisiting small batch training for deep neural networks," *arXiv preprint arXiv:1804.07612*, 2018.

[47] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[48] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[49] A. Anwar, "Difference between local response normalization and batch normalization," Medium, jun 2019, "https://towardsdatascience.com/difference-between-local-response-normalization-and-batch-normalization-272308c034ac", last accessed on 12/07/2020.

[50] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *International conference on artificial neural networks*. Springer, 2010, pp. 92–101.

[51] "Experimental security research of tesla autopilot," Tencent Keen Security Lab, mar 2019, "https://keenlab.tencent.com/en/whitepapers/Experimental_Security_Research_of_Tesla_Autopilot.pdf", last accessed on 14/07/2020.

[52] N. Morgulis, A. Kreines, S. Mendelowitz, and Y. Weisglass, "Fooling a real car with adversarial traffic signs," *arXiv preprint arXiv:1907.00374*, 2019.

[53] F. Khalid, M. A. Hanif, S. Rehman, and M. Shafique, "Security for machine learning-based systems: Attacks and challenges during training and inference," in *2018 International Conference on Frontiers of Information Technology (FIT)*. IEEE, 2018, pp. 327–332.

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

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

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

[57] A. Ashok, S. H. Kalli, and P. Sharma, "Fooling a deep neural network forever," *Semantic Scholar*, 2017.

[58] K. Grosse, P. Manoharan, N. Papernot, M. Backes, and P. McDaniel, "On the (statistical) detection of adversarial examples," *arXiv preprint arXiv:1702.06280*, 2017.

[59] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," *Pattern Recognition*, vol. 84, pp. 317–331, 2018.

[60] Q. Liu, P. Li, W. Zhao, W. Cai, S. Yu, and V. C. Leung, "A survey on security threats and defensive techniques of machine learning: A data driven view," *IEEE access*, vol. 6, pp. 12 103–12 117, 2018.

[61] S. Sabour, Y. Cao, F. Faghri, and D. J. Fleet, "Adversarial manipulation of deep representations," *arXiv preprint arXiv:1511.05122*, 2015.

[62] F. Khalid, M. A. Hanif, S. Rehman, and M. Shafique, "Security for machine learning-based systems: Attacks and challenges during training and inference," in *2018 International Conference on Frontiers of Information Technology (FIT)*. IEEE, 2018, pp. 327–332.

[63] F. Khalid, M. A. Hanif, S. Rehman, J. Qadir, and M. Shafique, "Fademl: understanding the impact of pre-processing noise filtering on adversarial machine learning," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 902–907.

[64] B. Biggio, G. Fumera, and F. Roli, "Security evaluation of pattern classifiers under attack," *IEEE transactions on knowledge and data engineering*, vol. 26, no. 4, pp. 984–996, 2013.

[65] I. Evtimov, K. Eykholt, E. Fernandes, T. Kohno, B. Li, A. Prakash, A. Rahmati, and D. Song, "Robust physical-world attacks on machine learning models," *arXiv preprint arXiv:1707.08945*, vol. 2, no. 3, p. 4, 2017.

[66] I. Neil, *CompTIA Security+ Certification Guide: Master IT security essentials and exam topics for CompTIA Security+ SY0-501 certification*. Packt Publishing Ltd, 2018.

[67] G. Agrawal, "Cia triad in details... looks simple but actually complex," MRCISSP CISSP Blog – Connecting the Unconnected, jan 2019, "https://mrcissp.com/2019/01/09/cia-triad-in-details-looks-simple-but-actually-complex/", last accessed on 14/10/2020.

[68] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 1322–1333.

[69] B. Biggio, "Machine learning under attack: Vulnerability exploitation and security measures," in *Proceedings of the 4th ACM Workshop on Information Hiding and Multimedia Security*, 2016, pp. 1–2.

[70] M. Itkina, Y. Wu, and B. Bahmani, "Adversarial attacks on image recognition," *CS229 - Stanford project*, 2016.

[71] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[72] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017, pp. 1–12.

[73] P. N. Whatmough, S. Das, D. M. Bull, and I. Darwazeh, "Circuit-level timing error tolerance for low-power dsp filters and transforms," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 6, pp. 989–999, 2012.

[74] X. Liu, R. Deng, K.-K. R. Choo, and Y. Yang, "Privacy-preserving outsourced support vector machine design for secure drug discovery," *IEEE Transactions on Cloud Computing*, 2018.

[75] Z. Huang, Q. Wang, Y. Chen, and X. Jiang, "A survey on machine learning against hardware trojan attacks: Recent advances and challenges," *IEEE Access*, vol. 8, pp. 10 796–10 826, 2020.

[76] C. Molnar, *Interpretable Machine Learning*. Lulu. com, 2020.

[77] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. D. Tygar, "Adversarial machine learning," in *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, 2011, pp. 43–58.

[78] J. Gomes, "Adversarial attacks and defences for convolutional neural networks," Medium, jan 2018, "https://medium.com/onfido-tech/adversarial-attacks-and-defences-for-convolutional-neural-networks-66915ece52e7", last accessed on 17/07/2020.

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

[80] S. Sengupta, T. Chakraborti, and S. Kambhampati, "Mtdeep: boosting the security of deep neural nets against adversarial attacks with moving target defense," in *Workshops at the thirty-second AAAI conference on artificial intelligence*, 2018.

[81] C. Dwork, "Differential privacy." Bugliesi M., Preneel B., Sassone V., Wegener I. (eds) Automata, Languages and Programming. ICALP 2006. Lecture Notes in Computer Science, vol 4052. Springer, Berlin, Heidelberg., 2006, "https://doi.org/10.1007/11787006_1", last accessed on 20/08/2020.

[82] A. Arcos-Garcia, J. A. Alvarez-Garcia, and L. M. Soria-Morillo, "Deep neural network for traffic sign recognition systems: An analysis of spatial

transformers and stochastic optimisation methods," *Neural Networks*, vol. 99, pp. 158–165, 2018.

[83] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural networks*, vol. 32, pp. 323–332, 2012.

[84] A. Bronshtein, "Train/test split and cross validation in python," Medium: Towards Data Science, may 2017, "https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6", last accessed on 11/09/2020.

[85] T. Shah, "About train, validation and test sets in machine learning," Medium: Towards Data Science, dec 2017, "https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7", last accessed on 11/09/2020.

[86] "Danskin's theorem," Wikipedia, may 2020, "https://en.wikipedia.org/wiki/Danskin%27s_theorem", last accessed on 24/07/2020.

[87] A. N. Bhagoji, D. Cullina, C. Sitawarin, and P. Mittal, "Enhancing robustness of machine learning systems via data transformations," in *2018 52nd Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2018, pp. 1–5.

[88] J. R. Flynn, S. Ward, J. Abich, and D. Poole, "Image quality assessment using the SSIM and the just noticeable difference paradigm," in *International Conference on Engineering Psychology and Cognitive Ergonomics*. Springer, 2013, pp. 23–30.

[89] H. L. Tan, Z. Li, Y. H. Tan, S. Rahardja, and C. Yeo, "A perceptually relevant MSE-based image quality metric," *IEEE Transactions on Image Processing*, vol. 22, no. 11, pp. 4447–4459, 2013.

[90] U. Sara, M. Akter, and M. S. Uddin, "Image quality assessment through FSIM, SSIM, MSE and PSNR—a comparative study," *Journal of Computer and Communications*, vol. 7, no. 3, pp. 8–18, 2019.

[91] A. Sharma, "Free gpus for everyone! get started with google colab for machine learning and deep learning," Analytics Vidhya, mar 2020, "https://www.analyticsvidhya.com/blog/2020/03/google-colab-machine-learning-deep-learning/", last accessed on 29/07/2020.

[92] A. Desmaison, "Spatial transformer network," deepai, sep 2015, "https://deepai.org/machine-learning-glossary-and-terms/spatial-transformer-network and http://torch.ch/blog/2015/09/07/spatial_transformers.html", last accessed on 26/09/2020.

[93] Z. Lu, S. Rallapalli, K. Chan, and T. La Porta, "Modeling the resource requirements of convolutional neural networks on mobile devices," in *Proceedings of the 25th ACM international conference on Multimedia*, 2017, pp. 1663–1671.

[94] K. Siu, D. M. Stuart, M. Mahmoud, and A. Moshovos, "Memory requirements for convolutional neural network hardware accelerators," in *2018 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2018, pp. 111–121.

[95] X. Giró-i Nieto, E. Sayrol, A. Salvador, J. Torres, E. Mohedano, and K. McGuinness, "Memory usage and computational considerations, course notes, accessed on jul. 5 2016," "http://imatge-upc.github.io/telecombcn-2016-dlcv/", last accessed on 04/10/2020.

[96] A. Karpathy, "Cs231n winter 2016: Lecture 4: Backpropagation," *Neural Networks*, vol. 1, 2016.

[97] A. Karpathy, J. Johnson, and L. Fei-Fei, "Visualizing and understanding recurrent networks," *arXiv preprint arXiv:1506.02078*, 2015.

[98] V. J. Easton and J. H. McColl, "Confidence intervals," Statistics Glossary v1.1, The University of Glasgow website, sep 1997, "http://www.stats.gla.ac.uk/steps/glossary/confidence_intervals.html", last accessed on 10/09/2020.

[99] D. C. G. Looney, "Part 1. image processing fundamentals, chapter 1. introduction to image processing," Emeritus Professor of Computer Science & Engineering University of Nevada, Reno, NV, U.S.A, jan 2006, "https://www.cse.unr.edu/~looney/cs674/storage/Chapter_1.htm", last accessed on 18/10/2020.

Appendix A

GTSRB Assigned Class Label

Table A.1: GTSRB Class Label used in the dataset

ClassId	SignName
0	Speed limit (20km/h)
1	Speed limit (30km/h)
2	Speed limit (50km/h)
3	Speed limit (60km/h)
4	Speed limit (70km/h)
5	Speed limit (80km/h)
6	End of speed limit (80km/h)
7	Speed limit (100km/h)
8	Speed limit (120km/h)
9	No passing
10	No passing for vehicles over 3.5 metric tons
11	Right-of-way at the next intersection
12	Priority road
13	Yield
14	Stop
15	No vehicles
16	Vehicles over 3.5 metric tons prohibited
17	No entry
18	General caution
19	Dangerous curve to the left
20	Dangerous curve to the right
21	Double curve
22	Bumpy road
23	Slippery road
24	Road narrows on the right
25	Road work
26	Traffic signals
27	Pedestrians
28	Children crossing
29	Bicycles crossing
30	Beware of ice/snow
31	Wild animals crossing
32	End of all speed and passing limits
33	Turn right ahead
34	Turn left ahead
35	Ahead only
36	Go straight or right
37	Go straight or left
38	Keep right
39	Keep left
40	Roundabout mandatory
41	End of no passing
42	End of no passing by vehicles over 3.5 metric tons

Appendix B

Attack Vectors Analysis Datasets

	A	B	C	D	E	F	G	H
1	Index	adv_ex_img_id	epsilon_value	actual_label	accuracy_value	adversatrial_label	execution_time	correct_or_incorrect
2	0	245	0	Roundabout mandatory	84.1419	Roundabout mandatory	0.042592	0
3	1	245	0.05	Roundabout mandatory	46.8629	Roundabout mandatory	0.033678	0
4	2	245	0.1	Roundabout mandatory	83.8433	Roundabout mandatory	0.033739	0
5	3	245	0.15	Roundabout mandatory	99.9998	Roundabout mandatory	0.032891	0
6	4	245	0.2	Roundabout mandatory	0.4604	Keep left	0.039209	1
7	5	245	0.25	Roundabout mandatory	1.1885	Priority road	0.034056	1
8	6	245	0.3	Roundabout mandatory	1.0349	Go straight or left	0.035047	1
9	7	245	0.35	Roundabout mandatory	30.9727	Priority road	0.037854	1
10	8	245	0.4	Roundabout mandatory	1.4959	Speed limit (50km/h)	0.032504	1
11	9	177	0	Roundabout mandatory	20.3532	Roundabout mandatory	0.035611	0
12	10	177	0.05	Roundabout mandatory	0.2294	Road work	0.033215	1
13	11	177	0.1	Roundabout mandatory	2.8156	Vehicles over 3.5 metric tons prohibited	0.032846	1
14	12	177	0.15	Roundabout mandatory	0.0668	No passing for vehicles over 3.5 metric tons	0.032269	1
15	13	177	0.2	Roundabout mandatory	0.3491	Speed limit (50km/h)	0.035135	1
16	14	177	0.25	Roundabout mandatory	0.0045	Speed limit (50km/h)	0.033051	1
17	15	177	0.3	Roundabout mandatory	0.1399	Keep right	0.032002	1
18	16	177	0.35	Roundabout mandatory	10.7615	No passing for vehicles over 3.5 metric tons	0.032187	1
19	17	177	0.4	Roundabout mandatory	3.4815	Speed limit (30km/h)	0.032235	1

Figure B.1: Dataset for FGSM Attack Analysis.

	A	B	C	D	E	F	G	H	I	J	K
1	index	img_id	imame_name_1	actual_label	imame_name_2	adversatrial_label	epsilon_v	MSE_value_org	SSIM_value	MSE_value_advex_	SSIM_value_advex_ima
2	0	245	1_00245_indi_0.jpg	Roundabout mandatory	1_00245_indi_1.jpg	Roundabout mandatory	0.05	0	1	138.87	0.94
3	1	245	1_00245_indi_0.jpg	Roundabout mandatory	1_00245_indi_2.jpg	Roundabout mandatory	0.1	0	1	106.29	0.99
4	2	245	1_00245_indi_0.jpg	Roundabout mandatory	1_00245_indi_3.jpg	Roundabout mandatory	0.15	0	1	373.91	0.83
5	3	245	1_00245_indi_0.jpg	Roundabout mandatory	1_00245_indi_4.jpg	Keep left	0.2	0	1	713.5	0.78
6	4	245	1_00245_indi_0.jpg	Roundabout mandatory	1_00245_indi_5.jpg	Priority road	0.25	0	1	941.79	0.75
7	5	245	1_00245_indi_0.jpg	Roundabout mandatory	1_00245_indi_6.jpg	Go straight or left	0.3	0	1	1211.93	0.72
8	6	245	1_00245_indi_0.jpg	Roundabout mandatory	1_00245_indi_7.jpg	Priority road	0.35	0	1	1533.82	0.71
9	7	245	1_00245_indi_0.jpg	Roundabout mandatory	1_00245_indi_8.jpg	Speed limit (50km/h)	0.4	0	1	1896.78	0.68
10	8	177	2_00177_indi_0.jpg	Roundabout mandatory	2_00177_indi_1.jpg	Road work	0.05	0	1	306.4	0.88
11	9	177	2_00177_indi_0.jpg	Roundabout mandatory	2_00177_indi_2.jpg	Vehicles over 3.5 metric tons prohibite	0.1	0	1	502.46	0.78
12	10	177	2_00177_indi_0.jpg	Roundabout mandatory	2_00177_indi_3.jpg	No passing for vehicles over 3.5 metric	0.15	0	1	695.99	0.72
13	11	177	2_00177_indi_0.jpg	Roundabout mandatory	2_00177_indi_4.jpg	Speed limit (50km/h)	0.2	0	1	713.37	0.7
14	12	177	2_00177_indi_0.jpg	Roundabout mandatory	2_00177_indi_5.jpg	Speed limit (50km/h)	0.25	0	1	947.42	0.68
15	13	177	2_00177_indi_0.jpg	Roundabout mandatory	2_00177_indi_6.jpg	Keep right	0.3	0	1	1329.93	0.66
16	14	177	2_00177_indi_0.jpg	Roundabout mandatory	2_00177_indi_7.jpg	No passing for vehicles over 3.5 metric	0.35	0	1	1759.42	0.64
17	15	177	2_00177_indi_0.jpg	Roundabout mandatory	2_00177_indi_8.jpg	Speed limit (30km/h)	0.4	0	1	1926.25	0.65
18	16	137	3_00137_indi_0.jpg	Roundabout mandatory	3_00137_indi_1.jpg	Roundabout mandatory	0.05	0	1	140.86	0.9
19	17	137	3_00137_indi_0.jpg	Roundabout mandatory	3_00137_indi_2.jpg	Turn right ahead	0.1	0	1	420.12	0.79
20	18	137	3_00137_indi_0.jpg	Roundabout mandatory	3_00137_indi_3.jpg	Priority road	0.15	0	1	564.64	0.73
21	19	137	3_00137_indi_0.jpg	Roundabout mandatory	3_00137_indi_4.jpg	Turn right ahead	0.2	0	1	793.4	0.7
22	20	137	3_00137_indi_0.jpg	Roundabout mandatory	3_00137_indi_5.jpg	Priority road	0.25	0	1	1008.55	0.68
23	21	137	3_00137_indi_0.jpg	Roundabout mandatory	3_00137_indi_6.jpg	Speed limit (20km/h)	0.3	0	1	1312.44	0.67

file_name_nontarget_SSIM_MSE_in

Figure B.2: Dataset for Image Quality Analysis for FGSM method with SSIM and MSE.

	A	B	C	D	E	F	G	H	I
1	index	adv_ex_img_id	epsilon_value	actual_label	accuracy_value	adversatrial_label	iteration_no	execution_time	correct_or_incorrect
2	0	252	0	Speed limit (20km/h)	85.1501	Speed limit (20km/h)	1	0.060128	0
3	1	252	0.058	Speed limit (20km/h)	0.444	Speed limit (100km/h)	59	3.89129	1
4	2	403	0	Speed limit (20km/h)	83.6272	Speed limit (20km/h)	1	0.058691	0
5	3	403	0.116	Speed limit (20km/h)	0.0261	Speed limit (100km/h)	117	7.85233	1
6	4	579	0	Speed limit (20km/h)	43.9634	Speed limit (20km/h)	1	0.058241	0
7	5	579	0.136	Speed limit (20km/h)	0.0006	Speed limit (100km/h)	137	9.733745	1
8	6	675	0	Speed limit (20km/h)	75.6501	Speed limit (20km/h)	1	0.067583	0
9	7	675	0.069	Speed limit (20km/h)	0.0908	Speed limit (100km/h)	70	4.984875	1
10	8	778	0	Speed limit (20km/h)	99.6628	Speed limit (20km/h)	1	0.066486	0
11	9	778	0.038	Speed limit (20km/h)	0.9752	Speed limit (100km/h)	39	2.662512	1
12	10	807	0	Speed limit (20km/h)	99.9992	Speed limit (20km/h)	1	0.057894	0
13	11	807	0.053	Speed limit (20km/h)	0.6976	Speed limit (100km/h)	54	4.010152	1
14	12	990	0	Speed limit (20km/h)	49.3687	Speed limit (20km/h)	1	0.062964	0
15	13	990	0.023	Speed limit (20km/h)	3.0337	Speed limit (100km/h)	24	1.682709	1
16	14	943	0	Speed limit (20km/h)	65.2645	Speed limit (20km/h)	1	0.066768	0
17	15	943	0.012	Speed limit (20km/h)	0.2347	Speed limit (100km/h)	13	0.871882	1
18	16	252	0	Speed limit (20km/h)	85.1501	Speed limit (20km/h)	1	0.060128	0
19	17	252	0.011	Speed limit (20km/h)	22.8454	Speed limit (120km/h)	12	0.850453	1

Figure B.3: Dataset for I-FGSM Attack Analysis.

	A	B	C	D	E	F	G	H	I	J	K
	Index	img_id	imame_name_1	actual_label	imame_name_2	adversatrial_label	epsilon_v	MSE_value_	SSIM_value_	MSE_value_	SSIM_value_
	0	252	1_00252_target_0.jpg	Speed limit (20km/h)	1_00252_target_1.jpg	Speed limit (100km/h)	0.058	0	1	293.67	0.86
	1	403	2_00403_target_0.jpg	Speed limit (20km/h)	2_00403_target_1.jpg	Speed limit (100km/h)	0.116	0	1	411.25	0.78
	2	579	3_00579_target_0.jpg	Speed limit (20km/h)	3_00579_target_1.jpg	Speed limit (100km/h)	0.136	0	1	467.13	0.82
	3	675	4_00675_target_0.jpg	Speed limit (20km/h)	4_00675_target_1.jpg	Speed limit (100km/h)	0.069	0	1	314.54	0.84
	4	778	5_00778_target_0.jpg	Speed limit (20km/h)	5_00778_target_1.jpg	Speed limit (100km/h)	0.038	0	1	267.39	0.92
	5	807	6_00807_target_0.jpg	Speed limit (20km/h)	6_00807_target_1.jpg	Speed limit (100km/h)	0.053	0	1	285.27	0.91
	6	990	7_00990_target_0.jpg	Speed limit (20km/h)	7_00990_target_1.jpg	Speed limit (100km/h)	0.023	0	1	252.38	0.95
	7	943	8_00943_target_0.jpg	Speed limit (20km/h)	8_00943_target_1.jpg	Speed limit (100km/h)	0.012	0	1	248.41	0.96
	8	252	9_00252_target_0.jpg	Speed limit (20km/h)	9_00252_target_1.jpg	Speed limit (120km/h)	0.011	0	1	243.52	0.97
	9	403	10_00403_target_0.jpg	Speed limit (20km/h)	10_00403_target_1.jpg	Speed limit (120km/h)	0.015	0	1	245.92	0.96
	10	579	11_00579_target_0.jpg	Speed limit (20km/h)	11_00579_target_1.jpg	Speed limit (120km/h)	0.039	0	1	264.08	0.94
	11	675	12_00675_target_0.jpg	Speed limit (20km/h)	12_00675_target_1.jpg	Speed limit (120km/h)	0.007	0	1	243.62	0.97
	12	778	13_00778_target_0.jpg	Speed limit (20km/h)	13_00778_target_1.jpg	Speed limit (120km/h)	0.031	0	1	255.22	0.94
	13	807	14_00807_target_0.jpg	Speed limit (20km/h)	14_00807_target_1.jpg	Speed limit (120km/h)	0.026	0	1	254.53	0.95
	14	990	15_00990_target_0.jpg	Speed limit (20km/h)	15_00990_target_1.jpg	Speed limit (120km/h)	0.016	0	1	249.11	0.96
	15	943	16_00943_target_0.jpg	Speed limit (20km/h)	16_00943_target_1.jpg	Speed limit (120km/h)	0.001	0	1	136.48	0.99
	16	132	17_00132_target_0.jpg	Stop	17_00132_target_1.jpg	Speed limit (120km/h)	0.05	0	1	291.44	0.89
	17	167	18_00167_target_0.jpg	Stop	18_00167_target_1.jpg	Speed limit (120km/h)	0.172	0	1	629.04	0.78
	18	111	19_00111_target_0.jpg	Stop	19_00111_target_1.jpg	Speed limit (120km/h)	0.125	0	1	451.79	0.79

Figure B.4: Dataset for Image Quality Analysis for I-FGSM method with SSIM and MSE.