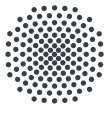


University of Stuttgart
Germany

University of Stuttgart

Institute of Computer Architecture and Computer Engineering

Chair of Hardware-oriented Computer Science

Master Thesis

Stochastic Computing in Neural Networks to Defend Against Adversarial Attacks

Masterarbeit Nr.: [redacted]

Manish Mahesh Dalvi

Course of Study:	Information Technology (InfoTech)
Examiner:	Prof. Dr. rer. nat. habil. Ilia Polian
Supervisor:	M.Sc Florian Neugebauer
Commenced:	09/04/2020
Completed:	09/10/2020

Erklärung:

Ich versichere, nach bestem Wissen und Gewissen, dass diese Arbeit kein Material und keine Quellen enthält, die bereits für das Erlangen eines akademischen Grades unter meinem Namen oder anderen Namen an einer weiteren Universität bzw. Hochschuleinrichtung veröffentlicht wurde, außer Quellen die als solche gekennzeichnet sind. Zusätzlich versichere ich, dass keine Teile dieser Arbeit in meinem Name in Zukunft eingereicht werden, um einen akademischen Grad an einer anderen Universität oder Hochschuleinrichtung zu erlangen ohne der Zustimmung der Universität Stuttgart.

Unterschrift:
Stuttgart, den

Declaration:

I certify that this work contains no material which has been accepted for the award of any other degree or diploma in my name, in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. In addition, I certify that no part of this work will, in the future, be used in a submission in my name for any other degree or diploma in any university or other tertiary institution without the prior approval of the University of Stuttgart.

Signature:
Stuttgart,

Abstract

Deep Neural Networks (DNNs) have gained importance and are used in various fields for its ability to learn features and draw mathematical models on its own. However, with the increase in usage of the DNN, there have been a rise in attacks on the network which could sabotage the operations, especially in real-time scenarios. These attacks known as Adversarial attacks specifically target the input image to make minor changes to pixel values which hamper the performance of DNN and causes a threat especially in safety critical applications. Stochastic Computing (SC) is a field which has the ability to perform arithmetic operations using simple logical circuits which results in lower power consumption, smaller hardware area and error correction. In this thesis, the randomness of SC is used in DNN by replacing arithmetic blocks with stochastic computing blocks for computations and determine the robustness of the DNN against Adversarial attacks. Different DNNs are trained on MNIST, Fashion MNIST, CIFAR-10 datasets and specific layers in each DNNs are replaced with Stochastic Layers. Before and after the replacement of binary arithmetic layers with SC layers, the attack is performed on the network and a comparison of robustness between the standard DNN and Stochastic Computing Neural Network (SCNN) is noted. Analysis of various optimizers used in the DNN and their performance in SC domain along with analysis of perturbed images in correct classification of a SCNN. The Stochastic Network proves its ability to handle minor perturbations by the use of randomness in the circuit. This resistance to adversarial attack helps in the real-world scenarios of attacks and error-prone environment.

Contents

Acronyms

AI Artificial Intelligence.

ALU Arithmetic Logic Unit.

APC Approximate Parallel Counter.

CNN Convolutional Neural Network.

DL Deep Learning.

DNN Deep Neural Network.

FC Fully Connected.

FSM Finite State Machine.

LFSR Linear Feedback Shift Register.

MUX Multiplexer.

NN Neural Network.

ReLU Rectified Linear Unit.

SC Stochastic Computation.

SCNN Stochastic Convolutional Neural Network.

SN Stochastic Number.

SSIM Structural Similarity Index Measure.

Stanh Stochastic Hyperbolic Tangent.

Tanh Hyperbolic Tangent.

Chapter 1

Introduction

Over the last decade, with the increase in funding for Artificial Intelligence (AI) projects the so called AI winter [1] has finally ended. The main reason for renewal of AI also known as AI Spring [1] is due to the increase in computation power with incredible hardware at significantly low cost compared to previous decades. There has been exponential growth in the collection of data which has added to importance of AI. With the increase in internet consumption and accessibility, the world is moving into new period of connected devices from phones to cars.

The introduction of information theory by Shannon [2] gave a new outlook for computation of numbers and also handle errors. The concepts of Information theory helped in opening a new paradigm of sending and receiving data over noisy channels. The received data had the ability to correct the noise introduced by various channels. These concepts have been widely even since and thus gave an alternate way of computation using probability. Currently, there is a significant use of binary computation in circuits which consume more power and more area but also provide accurate results. This accuracy is not necessary in every scenario instead at places where accuracy isn't significant, probabilistic based computing known as Stochastic Computing can be introduced.

Stochastic Computing [3] [4] addresses the problems of power consumption and area since the computation is performed using basic logic gates and circuits for a stream of random bits. The randomness of the bits helps in handling minor bit flips due to errors, attacks or environmental changes. Since DNNs have high number of computations, stochastic computing would significantly reduce the power consumption and area with a small trade-off in accuracy. This is useful when the neural network needs to be deployed onto a

embedded device such as a mobile phone where battery power consumption plays an important role.

Deep Learning (DL) is a sub area in AI which deals with learning of mathematical patterns from dataset without the need of forming mathematical models in prior. The networks in DL are called Deep Neural Networks(DNNs) since the networks usually have several layers between the input and output before producing results. Some of the most widely used data set for DNNs are MNIST [5], Fashion MNIST[6] and CIFAR-10[7]. In this Thesis, the networks trained with MNIST data sets are quite small for a very high accuracy of approximately 98% whereas CIFAR-10 requires more number of layers to obtain a test accuracy of 85%. Fashion MNIST dataset relatively lies between MNIST and CIFAR-10 in terms of complexity. Such Trained DNNs can be fooled to misclassify the new images with minor perturbations using adversarial attack such as Deep Fool Attack which severely drop the accuracy to near zero. This factor is worrying since these networks are deployed in real world scenarios and also in safety critical systems such as autonomous driving cars.

Stochastic computing due to it's randomness [8] and error handling capabilities can withstand such adversarial attacks. When the DNNs are replaced with SC layers, they tend to reduce the effects of adversarial attacks by introducing randomness in the network but this should be done in controlled manner since there are several parameters that can be changed in accordance to stochastic layers. The SC layers should have comparable accuracy and timeliness of output else it cannot be used in real world scenarios.

Deep Fool Attack[9] is an un-targeted attack which learns about the model and assumes the network is a linear classifier. The Attack then makes minor perturbations to the image until the model is able to misclassify the label to the nearest neighbour. The attack calculates the l_2-distance to the next nearest label and then pushes the image to the other side of the label boundary. The modified image is called as perturbed image.

Depending on the depth and number of computations, one to several layers of the DNNs are replaced with Stochastic Layers. This replaced layers DNN is known as Stochastic Convolutional Neural Network (SCNN). The perturbed images are passed through the SCNN which provides better accuracy compared to the binary computation blocks of the Conventional Convolutional Neural Network (CNN). The standard CNN gets significantly affected by the perturbed images and misclassifies the input, whereas the SCNN is able to

nullify the perturbations to provide better results.

The structure of the Master thesis is as follows. In Chapter 2, the fundamentals of CNNs, SC and the adversarial attacks are discussed. Subsequently Chapter 3 provides the experimental setup of the different datasets used and the attacks on the trained CNN. Following which the results and analysis of the outcomes of the results are provided in Chapter 4 and 5. The final part of the Thesis would conclude and discuss the future work of the thesis in chapter 6.

Chapter 2

Background

2.1 Deep Neural Network

The First Neural Network was a small model called as Perceptron in 1957 designed by Rosenblatt [10]. Perceptron is similar to a Logistic Regression model. Logistic Regression(LR) is basically a statistical model which uses a logistic curve/Sigmoid (Fig 2.1) in order to distinguish between two types of data. LR develops a hyperplane which can perform binary classification. The main inspiration behind the development of Neural Networks was the network of the axons and dendrites in the brain of many animals and humans. There is a huge similarity between the biological neuron and the artificial neural node. With the idea of the perceptron and the idea of back-propagation [11] introduced in 1986, networks of perceptron connected to each other which learnt the features from the input data was introduced. Due to lack of computational power in 1986, the idea of Neural Networks (NNs) subsided. In the last decade, NNs gained more relevance due to high computation and cheap resource cost availability. A sub category of Neural Networks is Convolutional Neural Networks (CNN). The inspiration behind the idea of CNN was the visual cortex in mammals, which consists of layers of sensors which splits an image into different layers where each layer detects different features. Each layer in the visual cortex recognises motion, stereo, color, face, attention and many more. This layered splitting of features increased the development of convolutional layers.

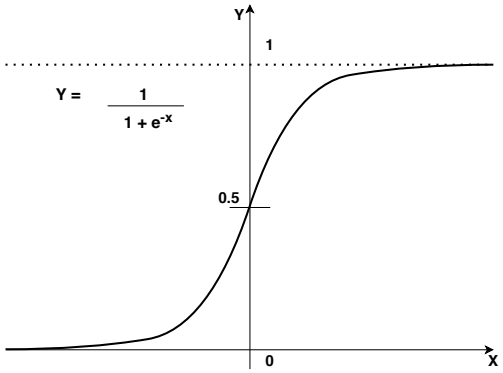


Figure 2.1: Logistic Function also known as Sigmoid function

2.1.1 Convolutional Neural Network

CNN is a sub section of NN which mainly deals with classification of images. A CNN consists of a input image matrix, followed by one to many convolutional layers, sometimes in combination with max pooling, dropout, batch normalisation and finally ending with fully connected layers before the output layer. One such example is shown in Fig 2.2.

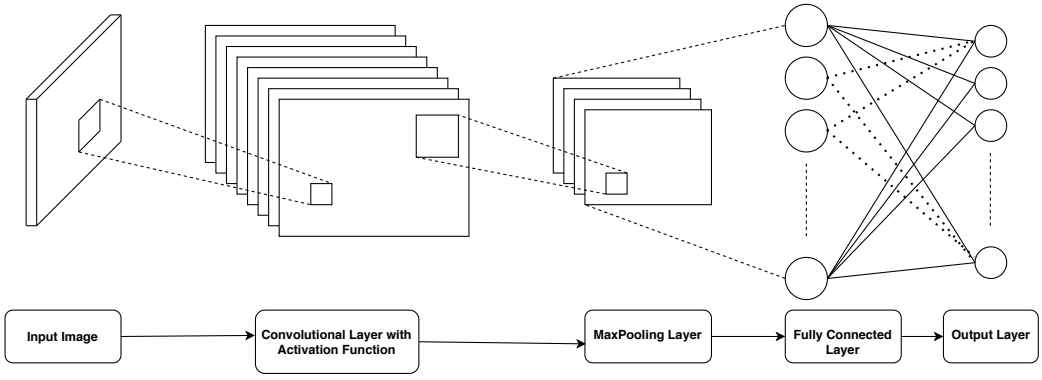


Figure 2.2: Simple Convolutional Neural Network Architecture: The input data is passed from one stage to the other where several features of the image are learnt.

Convolution Layer

A convolutional layer consists of two matrices, first the input matrix and second the weight matrix. The input matrix convolves over the weight matrix

to obtain the result matrix. The weight matrix in the CNN is learnt over iterations to obtain the best weight matrix that produces the least loss.

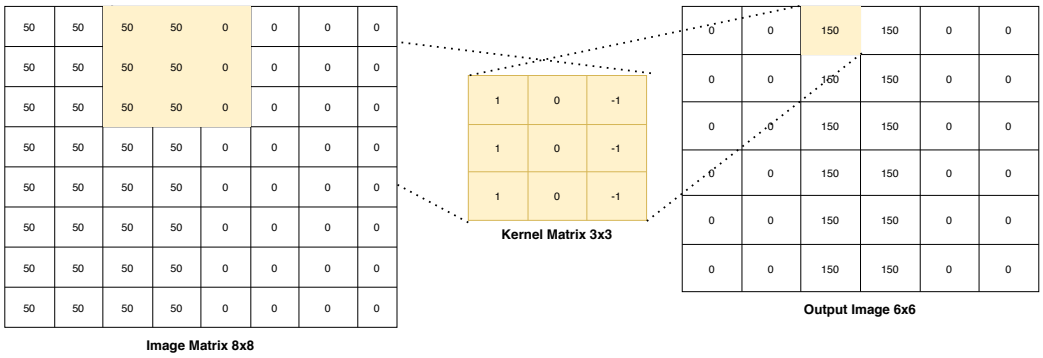


Figure 2.3: Convolution for Edge Detection from an Image: The Input image has half image with bright pixels and other with dark pixels. The Kernel matrix detects the line of transition between bright to dark.

The weight matrix also known as kernel matrix shown in the Fig 2.3 is an example of an edge detector kernel. The input image consists of a transition from brighter pixels to darker pixels. The brighter pixels are represented as 50 and the dark pixels as 0. Pixels can take values from 0 to 255, where 0 being black and 255 being white. The kernel matrix moves over the input image with element wise multiplication and addition which results in the output image. The movement of the kernel over the input image is known as striding. Stride of 1 is the movement of kernel matrix over the input image by 1 in the particular direction. Consider the highlighted area in the image matrix and the kernel matrix. Both the matrix are multiplied element wise meaning such as $\sum_{i=1}^{3}\sum_{j=1}^{3} Image_{ij} * K_{ij} = 150$ where i, j are row and column number under consideration. The input image can be padded with extra zeros over the matrix edges to result in a matrix of required size which is known as padding. The output image finally consists of the detected edge. The different weights in the kernel matrix determine the features that is detected. The output size depends on the padding, striding and kernel size. The output depth depends on the number of layers in the kernel. The depth dimension of the kernel is a hyper-parameter which can be tuned during the training and testing process. The kernel weights provides features like edge detection, blurring, sharpening, unsharpening etc.

The general mathematical notation of the convolution is as follows.

$$x_{i,j,k} = (\sum_{n=0}^{c-1} \sum_{m=0}^{p} \sum_{l=0}^{q} w_{l,m,n} * a_{k+n,j+m,i+l}) + b \tag{2.1}$$

Assume, from the Fig 2.4 the input image is of the size (h, w, d) and the kernel matrix is of the size (k_h, k_w, d), then the output matrix is of the size $(h - k_h + 1, w - k_w + 1, d)$.

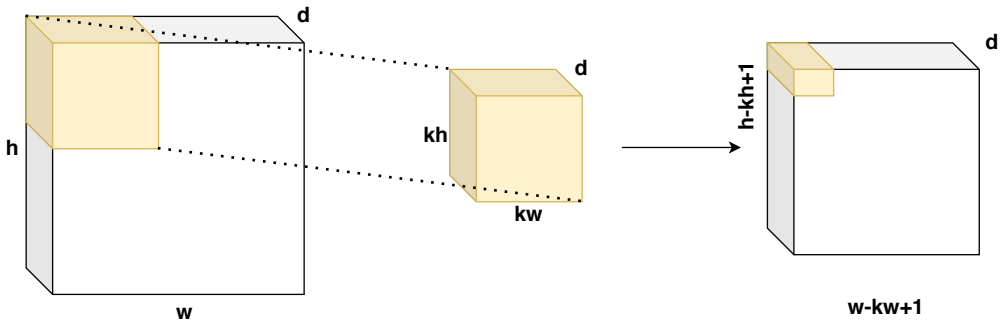


Figure 2.4: Convolution Input matrix, Kernel and Output sizes: Kernel matrix is element-wise multiplied to highlighted input matrix to obtain part of output matrix.

Activation Layer

Activation layers consists of non-linear functions that map each value from the output of the convolutional layer to required range. There are several functions such as ReLU, Tanh, Sigmoid, Leaky ReLU etc. Each of the functions have their own purpose. The Sigmoid function and Tanh functions are known as squashing functions as they map input values to $[0, 1]$ and $[-1, 1]$ respectively. Each function has their advantage and disadvantages. The Advantage of sigmoid and Tanh (Fig 2.5) is their the mapping range of [-1, 1] where as the disadvantage is the problem of vanishing gradient where the weight tends to stop learning in course of time due to very small values. This disadvantage is over come by the ReLU activation function (Fig 2.6). The ReLU solves the problem of vanishing gradient by removing negatives values altogether but introduces the problem of non differentiability at zero. This is in turn solved by the Leaky ReLU. Since the Thesis focuses on Stochastic numbers which lie between [-1, 1], sigmoid and Tanh are used over ReLU.

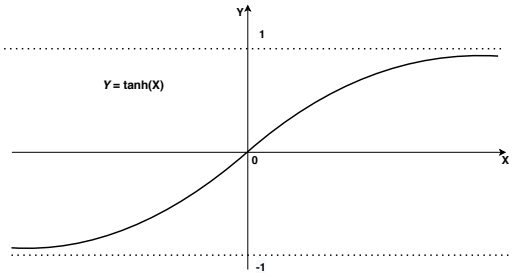


Figure 2.5: Hyperbolic Tangent Function: Tanh activation function used in the activation layer of a NN

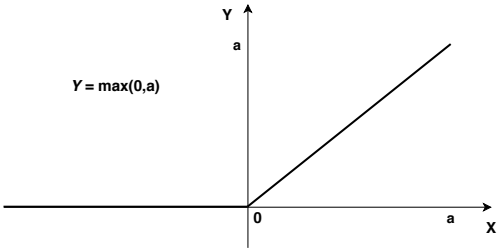


Figure 2.6: ReLU Activation Function: Substitute for Tanh activation function to prevent vanishing gradient.

Max Pooling Layer

The main objective of Max Pooling layer is down sampling of the output from activation layer to the required size for the next the convolutional layer or fully connected layer. Max pooling function picks the maximum pixel value among the given window size. For example, consider a 2 x 2 window size and input of 32 x 32. For every 2 x 2 sub matrix in the input, the maximum value pixel is passed onto the output. The resultant matrix has a size of 16x16. The Max pooling layer (Fig 2.7) has an added advantage of location invariance, rotational invariance and scale invariance. For example, wherever a face is located in an image, irrespective of its inclination and size, the face is detected. Max pooling layer provides features of selection of brighter pixel, contrast, edge highlight.

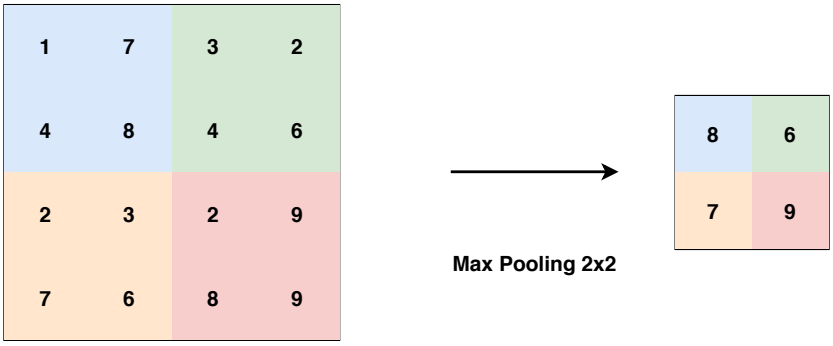


Figure 2.7: Max Pooling 2x2 Function: Same coloured sub matrix are mapped to output that contains the highest value.

Dropout Layer

Dropout is a regularisation technique which prevents a neural network from over-fitting by dropping connections between adjacent layer neurons during training. Over-fitting of a network is when the network memorizes the data instead of learning the underlying relation which results in high variance and low bias. The dropout layer (Fig 2.8) helps in lowering the dependency between neurons and results in lowering of over fit values. A dropout value 0.5 means dropping 50% of the connections to the next layer neurons. Starting layers should have lesser dropout rate and deep layers should have higher dropout rates.

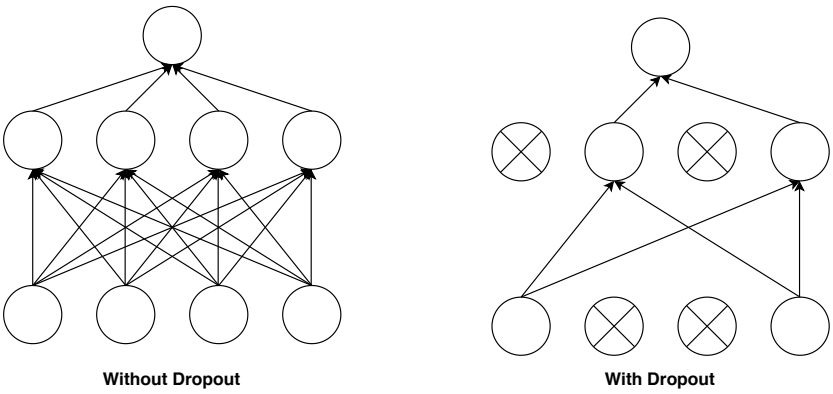


Figure 2.8: Dropout Layer: At a dropout rate of 0.5, 50% connections and nodes are dropped between the two layers.

Fully Connected Layers

Fully Connected (FC) Layer is a type of NN layer where each neuron in the previous layer is connected to each neuron in the next layer. This is also known as densely connected layer. In a CNN, the last convolutional layer output is flattened and fed to the FC layers. The FC layers along with activation function predicts the final output matrix or class. The final FC layer consists of neurons which equal the number of output classes. The output class is determined by applying softmax function to each neuron in order to predict the output class. The softmax function basically returns the probability of each class. The class which has the highest probability is the output class of the complete CNN.

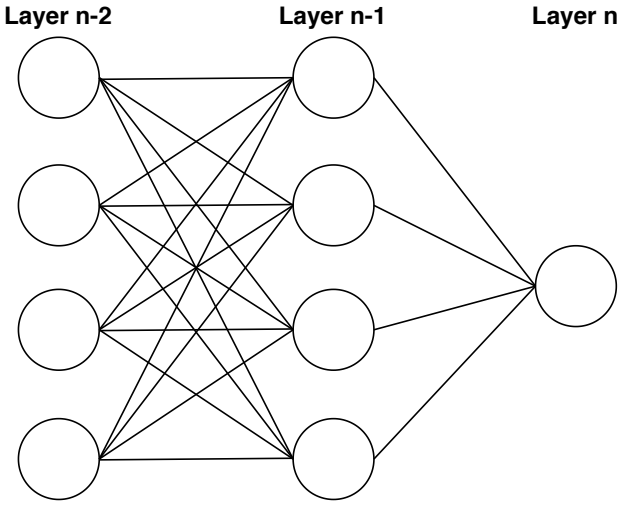


Figure 2.9: Fully Connected Layers: Nodes in one layer connected to every node in the next layer.

Learning and Back-propagation

The DNN learns the data using a concept called Back-propagation. For each input image, the output value predicted by the DNN for each class is compared to the actual value of all labels. The squared difference between the actual and predicted values is the error per class. The partial derivative of the squared error is taken in order to obtain the slope towards the minimum loss. This loss is multiplied with a learning rate η. The learning rate is a hyper-parameter which can be tuned. If the learning rate is high then the network keeps oscillating between the losses and if the learning rate is too small then the network does not learn. In both cases the minimum loss is not

obtained, thus the learning rate should be optimum. Once the slope of the error is multiplied with learning rate, the current weights are subtracted with the loss in order to obtain new weights (equation. 2.4). This new weights have converged one step towards an optimum point where the DNN can provide better results.

$$error = (Y_{true} - Y_{pred})^2 \tag{2.2}$$

$$L = 2 * (Y_{true} - Y_{pred}) \tag{2.3}$$

$$W_{new} = W_{old} - \eta * (\partial L / \partial W_{old}) \tag{2.4}$$

CNN Overview

All the above sub sections discussed each block used in the CNN which are combined in various combinations in order to obtain a network that is able to learn various features from the dataset. The usual arrangement of the above discussed block are as follows. Consider only shallow network, which has only one convolutional block, activation function, max-pooling layer, dropout layer, FC layer and finally a output softmax layer. This shallow network is an example of a complete CNN network. If the dataset used to train this shallow network is of binary classification, then the network mainly learns an optimum hyper-plane which is able to differentiate between the two classes. The weights are learnt using backpropagation. The hyperplane learnt by the network usually is an affine function (equation. 2.5).

$$Y(x) = f(w^T x + b) \tag{2.5}$$

In equation 2.5, Y is the output, f is the activation function like Sigmoid, ReLU, Tanh etc., w are the weights, b is the bias.

This affine function which is an equation of line separates the two classes. The weights when multiplied with the input and addition of bias mainly provides us the distance of the point from the hyperplane. If the point lies above the hyperplane, its value is positive else if the point is below the hyper-plane then its value is negative. The binary classification can be extended to multi-class classification, which results in several hyperplanes in order to correctly classify images.

There are high number computations in each layer of the CNN. These computations are performed using complex Arithmetic logic units (ALUs) and consume significant amount of time and power. This problem can be solved

by using Stochastic Computing (SC), where the ALUs can be replaced with simple logic circuits like AND gates, MUX in order to perform multiplication and addition respectively. These stochastic circuits consume much less power and area compared to an ALU. Before integrating DNN with SC, it is important to understand the working of SC blocks with its advantages and limitations. The next section discusses and introduces all the required SC blocks in order to replace the ALU functions.

2.2 Stochastic Computation

Stochastic Computing (SC) was introduced in 1960s [4] as an alternative to binary computing. In the past, the usage of SC had declined due to its long computation time and low accuracy. With advancement in computation power at affordable rates, SC has come into lime light due to its low power consumption, small area and error handling capabilities. The time complexity for performing arithmetic operations is significantly reduced from the binary circuits but with a trade-off in accuracy, which is an viable option when it comes to DNN due to high number of computations. Table 2.1 provides the timeline of the advancement in SC.

Dates	**Developments**	**References**
1956	Fundamental concept of probabilistic logic design.	[von Neumann 1956][12]
1960-79	Definition of SC and introduction to basic concepts. Construction of general-purpose SC computers.	[Gaines][4] [Poppelbaum][13]
1980-99	Advances in the theory of SC. Studies of specialized applications of SC, including artificial neural networks and hybrid controllers.	[Jeavons et al.1994][14] [Kim and Shanblatt 1995][15] [Toral et al. 1999][16]
2000-present	Application of efficient decoding of error correcting codes. New general-purpose architecture.	[Gaudet and Rapley 2003][17] [Qian et al. 2011][18]

Table 2.1: Timeline for the Development of Stochastic Computation.

In SC, there are two ways of representing a number, mainly a) Unipolar which ranges from [0, 1] and b) Bipolar ranging from [-1, 1]. The numbers are represented as streams of bits converted from their respective probability values. Let $p = 0.75$, then a bit-stream S contains 75% 1s and 25% 0s. S can be represented as 1110, 0111, 1011 and so on. This is an example for Unipolar stochastic number.

The length of bit stream in the above example is known as Stochastic number (SN) length which is 4. Depending on the required accuracy, the SN length can be increased or decreased. If $p = 0.125$, the SN length should be 8 since with SN length 4, we can have a resolution of 0.25. Probability p depends on the number of 1s in the bit stream and not on the position of the 1s and 0s.

SNs provide an advantage of performing basic arithmetic computations such as multiplication, additions with simple gates such as AND gate, MUX respectively. If two SN bit stream S_1 and S_2 with probability of occurrence of 1s are p_1 and p_2, then the multiplication $p_1 * p_2$ can performed using a logical AND gate. The main requirement for performing such arithmetic computations in Stochastic paradigm is that the numbers are to be uncorrelated or independent. Let S_1 and S_2 be two n-bit binary sequences. The sequences are said to be uncorrelated if and only if

$$\sum_{i=1}^{n} S_1(i)S_2(i) = \frac{\sum_{i=1}^{n} S_1(i) * \sum_{i=1}^{n} S_2(i)}{n} \tag{2.6}$$

where $S_1(i)$ and $S_2(i)$ are the i^{th} bit of sequence S_1 and S_2.

There are two ways of representing a Stochastic Number - Unipolar and Bipolar.

1. Unipolar stochastic number is represented as n_1/n

2. Bipolar stochastic number is represented as $(n_1 - n_0)/n$

where n_1 is the number of 1s in SN, n_0 is the number of zeros and n is the length of SN. The range of Unipolar and Bipolar SN is [0, 1] and [-1, 1] respectively. Unipolar can be converted to Bipolar and vice versa by equation 2.7.

$$P_{bipolar} = 2 * P_{unipolar} - 1 \tag{2.7}$$

Let Y be an SN with length 8, represented as 1100 0000. In Unipolar, the SN becomes $p(y = 1) = 2/8 = 0.25$ and in bipolar format it is represented as $p(y)_{bipolar} = (2 - 6)/8 = -0.5$. If there is single bit flip in the stochastic number say 1110 0000 or 1100 0010 from the original, the unipolar number deviates to $3/8 = 0.375$ in unipolar format and -0.25 in bipolar format. But in binary 0.25 is represented as $(0.01)_2$ and when a bit flip occurs, the number changes to $(0.11)_2$ which equals 0.75. The magnitude of error in binary is 0.5 where as in SN is 0.125. With increase in SN length the error handling capability increases as the resolution tends to become smaller which proves the point of SNs being more error tolerant.

Stochastic Computing Circuits

Multiplication: Multiplication in the Stochastic domain is computed by using a simple AND gate for Unipolar numbers and XNOR for bipolar numbers. Consider two SNs of length 8, where $x = 2/8$ and $y = 4/8$. Since SNs can have multiple representations of the same number due to the different positions of 1s, the best way to represent numbers are independent and uncorrelated to each other. The output of multiplication of the numbers are $(2/8) * (4/8) = (8/64) = (1/8)$.In Fig 2.10 and Fig 2.11 the multiplication of uncorrelated numbers and correlated numbers can be observed. The correlated numbers tend to provide a more approximate solutions when compared to the exact solutions provided by uncorrelated numbers.

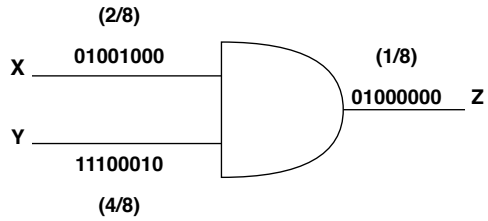


Figure 2.10: Stochastic multiplications for exact computation in Unipolar format

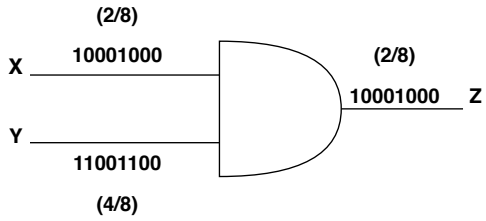


Figure 2.11: Stochastic multiplications for approximate computation for Unipolar numbers

Let us consider squaring of a SN using an AND gate. When a number is squared using an AND gate, the result obtained is the same number as the output instead of the square of the number i.e. $a*a = a$ and not a^2 (Fig 2.12). This is due to high correlation between the numbers. In order to obtain the square of a number, the two SNs generated for the same binary number need to be independent and uncorrelated which satisfy the condition provided in equation 2.6.

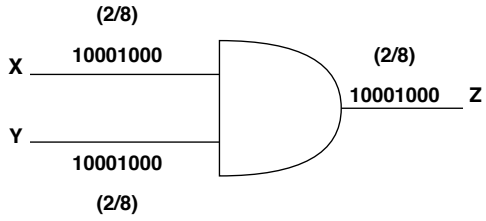


Figure 2.12: Problem of Squaring in Stochastic Multiplication

Addition: Addition of Stochastic numbers are performed using a n-way multiplexers where n being the number of inputs. Consider addition of two unipolar SNs having a range [0, 1]. The resultant number is in the range [0, 2] which is not in range for Stochastic numbers. Thus, the result needs to be scaled down in order to be in the range [0, 1]. In SC, the select input (S_{sel}) is used to scale down the resultant. $P(S_{sel}) = 0.5$ scales down the result into the required range. The addition of two stochastic numbers is performed using multiplexer as shown in Fig 2.13 using equation 2.8

$$P(S_{out}) = P(S_{sel}) * P(S_1) + (1 - P(S_{sel})) * P(S_2) = (P(S_1) + P(S_2))/2 \quad (2.8)$$

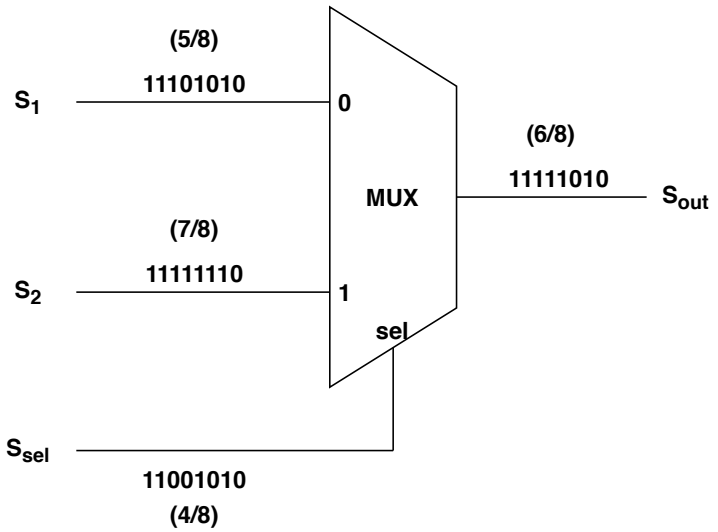


Figure 2.13: Addition in Stochastic Domain using Multiplexer: S_{sel} at 0.5 scales down the addition from [0,2] to [0,1]

Stochastic Number Generator (SNG): As discussed above, the conversion of binary probability values to SNs should be done such that the numbers generated are independent and uncorrelated with each other. Stochastic Number generator converts a binary number to a SN. One such generator is the Linear Shift Feedback Register (LFSR) which is the most commonly used Stochastic Number Generator (Fig 2.14). The SN can be converted back to binary by using a simple counter which counts the number of 1s in the bit stream (Fig 2.15)

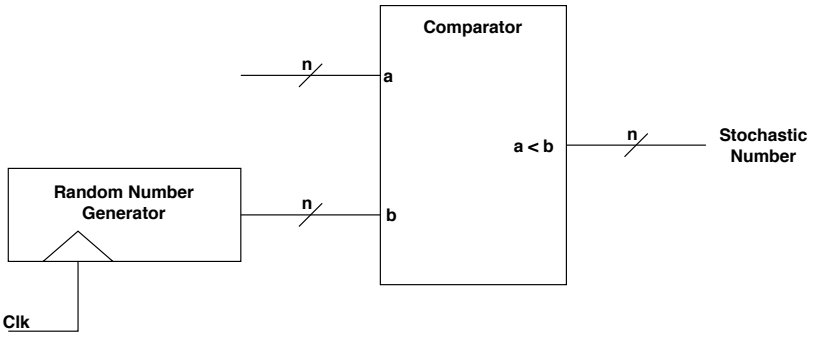


Figure 2.14: Stochastic Number Generator: Input number a compared with random number b to generate SN

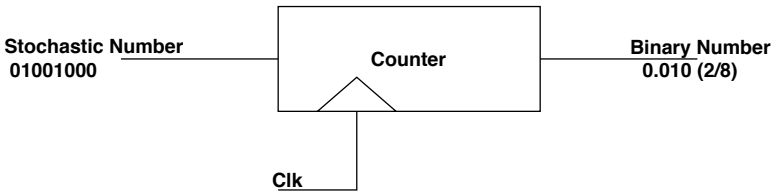


Figure 2.15: Stochastic to Binary Conversion using a simple counter to count number of 1s.

Hyperbolic Tangent function: One important sub-module of Neural Networks are the Activation functions. Stochastic computing mainly uses the hyperbolic Tangent function (Tanh) as the activation functions since the range of the function lies between [-1, 1] which is the same for Stochastic computation. There are several other activation functions like sigmoid, ReLU which can also be used as the activation function in SC. In this thesis, Tanh activation is used in the SC due to smaller hardware area. Activation function Tanh in SC in implemented using K-state finite state machine(FSM)[19] as shown in Fig 2.16.

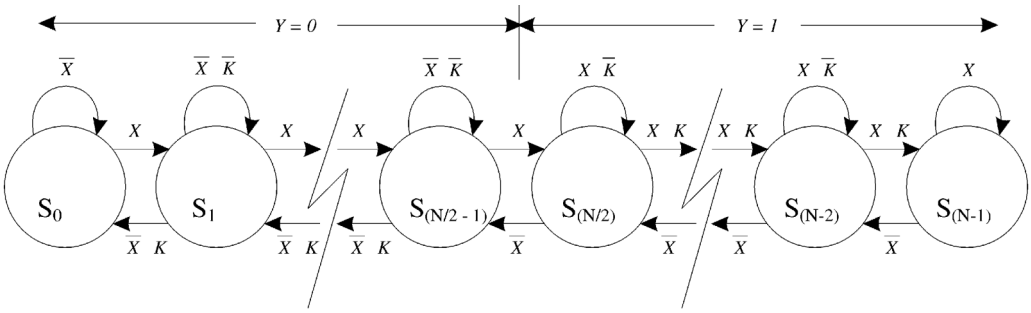


Figure 2.16: Stanh implementation using Finite State Machine [19]. N is number of states which should be even.

The working of the Tanh SC version is straight forward. Assuming N states in the FSM and N be even, the FSM fetches every SN bit from the bit stream. Assuming the current state of FSM is S, if the input is 1, the FSM moves to the next state $S + 1$ and if the input is 0, FSM movies to previous state $S - 1$. The FSM of N states is divided into two equal parts where first half states ($S < N/2$) results in 0 output and the second half states ($S > N/2$) outputs 1. The output of the FSM of Tanh is called Stochastic Hyperbolic tangent which is an approximation of Tanh function. The relation between

the hyperbolic tangent (Tanh) and the stochastic hyperbolic tangent (Stanh) is provided in the equation 2.9.

$$Stanh(N, x) \approx tanh(x * N/2) \tag{2.9}$$

The output states and the comparison graphs of the Tanh and Stanh is shown in Fig 2.17

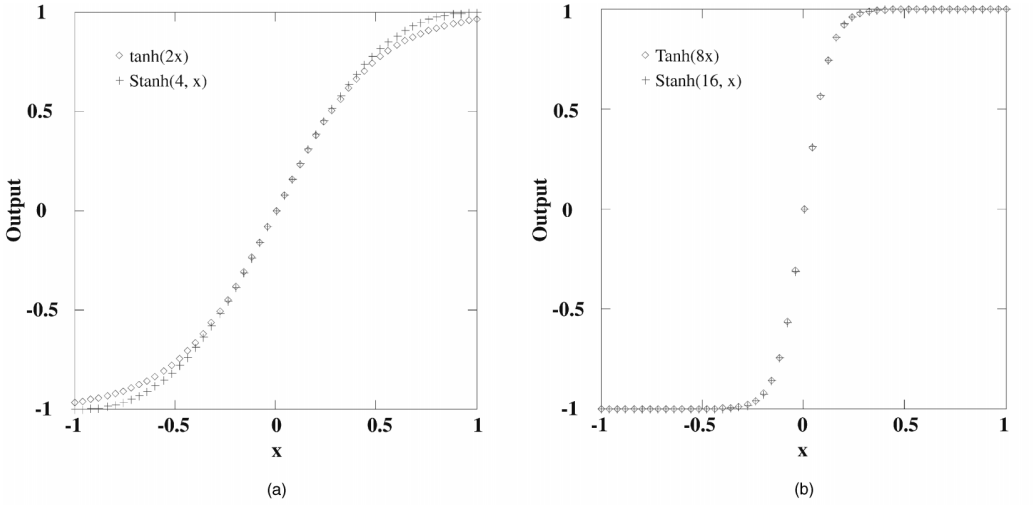


Figure 2.17: a) Stanh(4, x) from 4-state machine and tanh (2x) for comparison. (b) Stochastic activation function Stanh (16, x) from 16-state machine and Tanh (8x) for comparison [19].

Stochastic Computing Overview: With the understanding of the advantages, limitations and working of SC computing blocks, layers of the DNN can be replaced using the above discussed SC circuits. Each layer in DNN, especially the convolutional layers, consists of high number of multiplications and additions which can be replaced with XNOR gates and Multiplexers. Depending on the requirement of accuracy, the SN length can be increased or decreased. This allows the DNN to provide exact to approximate solution with a tradeoff between time and accuracy.

2.3 Stochastic Convolutional Neural Networks

Stochastic Convolutional Neural networks (SCNN) are Convolutional Neural Networks where the binary computation block like addition, multiplication, activation functions are replaced by SC blocks. One to several layers of the DNN are replaced with SC blocks depending on the accuracy required. Replacing all layers with SC blocks need not be the best solution in every case as this causes a significant increase in the SN length, which further increases the computation time thus losing the main purpose of SC. This section discusses the integration of the above sections of Neural Networks and Stochastic Computation.

Stochastic Inputs: The inputs to the Stochastic blocks in a SCNN are bit stream of the binary converted SN. Depending on the accuracy desired the length of the stochastic computing can be varied. For example, if the network requires m-bit for accuracy then the SN length is 2^m. The stochastic number length increases exponentially which in turn increases computational power and time consumption significantly. A trade off between the computation time and power with the accuracy is required in order to benefit the use of SNs. The conversion from binary to SN is provided in the equation 2.7.

Stochastic Convolutional Layer: In a Neural Network (NN), Convolutional operation requires multiplication and addition of elements. In stochastic domain, the multiplication is replaced by XNOR gate for bipolar SNs and the addition operation is replaced by scaled MUX (Fig 2.13). Consider two bipolar SNs SN_1 = 11000000 and SN_2 = 11101101. The SN_1 in binary is -1/2 and SN_2 is 1/2. The multiplication of the operations in binary should lead to -1/2 which is an approximate value to expected -1/4. The multiplication in Stochastic version is calculated as SN_1 XNOR SN_2. which results in $P(output) = -4/8 = -1/2$ as shown in Fig 2.18

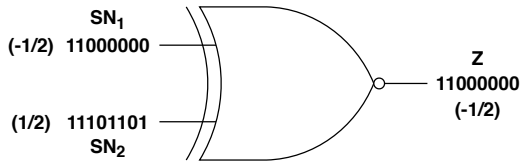


Figure 2.18: Multiplication of bipolar stochastic numbers using XNOR gate

Similarly, addition of the numbers is computed using the scaled down MUX. The multiplications in the Convolutional layers need to be added in order to obtain the output of the Convolutional layer. Consider addition of 4 numbers

x_1, x_2, x_3, x_4 which are the results of the multiplication of input and weights of NN using stochastic multiplication. The 4 way multiplexer with 2 control gates with probability of 0.5, scale down the addition by a factor of 4. Finally, the output of the 4 way MUX is an input to the next stage 2 way MUX for addition of bias. The select gate S_3 (Fig 2.19) has a probability of 4/5 for scale down addition in stochastic domain to provide equal weight-age to the 4 inputs from the 4-way MUX and 1 input from 2-way MUX.

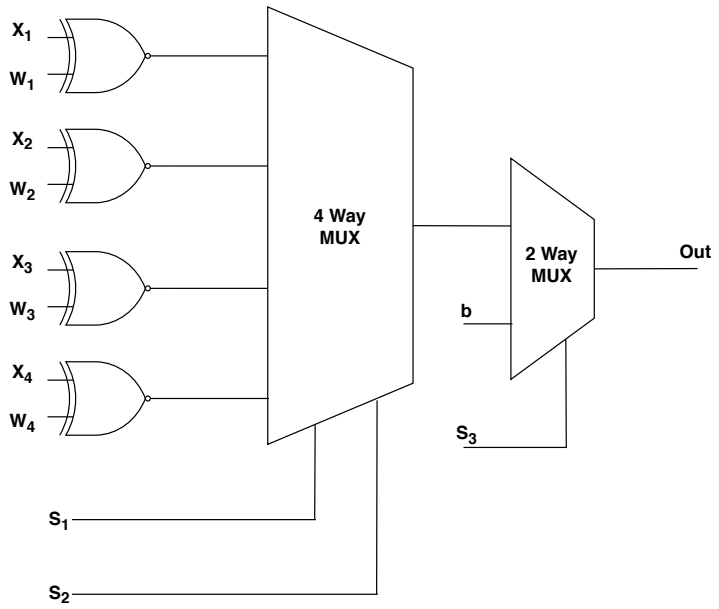


Figure 2.19: Stochastic Convolution using MUX and XNOR gate for arithmetic operations

Stochastic Activation function Tanh: The output from the Stochastic Convolutional layer is served as an input to the Stanh activation function. The Stanh is a FSM based Tanh function as shown in Fig 2.17. There is an alternate method of computing stochastic Tanh which is introduced in the paper [20]. This alternate method combines the addition of the convolutional layer along with Stanh functionality. The addition is performed in the binary domain and Tanh in the stochastic domain using Approximate parallel counters (APCs). The problem with stochastic addition is mainly of scaling down the values by huge factor thus reducing the available information. The alternate Tanh activation function is known as Btanh. Given n bit-streams having m bit length, the number of 1s is counter in each column by using a parallel counter. The counted value is fed into the up/down counter. The

resulting value from the counter is regarded as the state. At start, the initial state is at middle of the FSM as S_{half}. The number of 1s is counter for the m^{th} bit among all the n numbers are counted. The counted value is multiplied with 2 and then subtracted with n (number of bit streams). This value is added to the initialised state S. If the resultant value S after addition is greater than maximum state then S is assigned the value of the last maximum state of S_{max}. If the resultant S is negative then S is assigned 0. If the state S is greater than S_{half} the Y_i is 1 else 0. This entire functionality of Btanh is explained in the algorithm 1. n is the number of bit-streams. m is the length of each bit-stream. r is the number of states in up/down counter. t is the bipolar SC input, n x m size. t_i is the i^{th} column of t, t = $[t_1, \ldots, t_m]$.

Algorithm 1: Btanh(n, r, t)

Output: Y_i is the i^{th} stochastic output bit for Btanh.
$S_{max} = r - 1$
$S_{half} = r/2$
$S \leftarrow S_{half}$
for *i = 1 to m* **do**
    $V = Count(t_i) * 2 - n$ // bipolar 1s counting
    $S = S + V$
    if $S > S_{max}$ **then**
        $S \leftarrow S_{max}$
    end
    if $S < 0$ **then**
        $S \leftarrow 0$
    end
    if $S > S_{half}$ **then**
        $Y_i \leftarrow 1$
    else
        $Y_i \leftarrow 0$
    end
end

Stochastic Max Pooling Layer: Max pooling layers are used to down sample the results obtained from previous layers in the Neural Networks. The Stochastic version of the binary Max pooling layer is implemented using the NMax function presented by authors in [21]. The Pseudo code for the NMax Function is provided in the paper [21] which is as shown in algorithm 2. Both the for loops are calculated simultaneously.

Algorithm 2: Nmax Algorithm for Maximum of 3 Stochastic Numbers

for *each input SN i in cycle m* **do**
  if $Cnt_i = 0$ **then**
    Set $Cout_i = 0$
  else
    Set $Cout_i = 1$
  end
  Set $O_i = X_{i,m} \wedge \overline{Cout_i}$
end
for *each input SN i in cycle m* **do**
  Set $Inc_i = \overline{X_{l,m}} \wedge Z$
  Set $Dec_i = Cout_i \wedge X_{i,m} \wedge \overline{(V_{j \neq i} O_j)}$
end

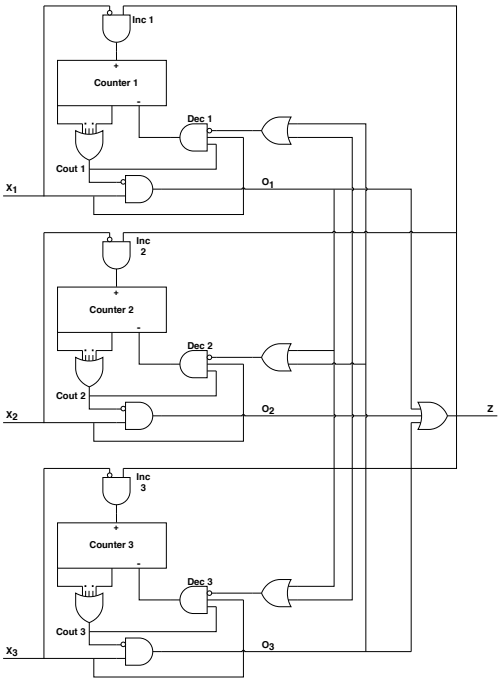


Figure 2.20: NMax Circuit Diagram [21]. NMax Circuit for Stochastic Max Pooling Layer to compute maximum value among the given stochastic bit streams.

Clock Cycle	1	2	3	4	5	6	7	8
X_1	1	1	0	1	0	1	1	1
X_1	1	0	0	1	0	0	1	0
X_1	0	0	1	0	0	0	0	1
Counter 1	0	0	0	0	0	0	0	0
Counter 2	0	1	1	1	1	2	2	3
Counter 3	1	2	1	2	2	3	4	4
Z	1	1	0	1	0	1	1	1

Table 2.2: Stochastic NMax Circuit Computation Example

The Stochastic NMax circuit shown in Fig 2.20 computes the maximum of 3 stochastic bit streams. The 3 inputs X_1, X_2, X_3 have respective counters which tracks the difference $Counter_i = max(CX_1^m, CX_2^m, CX_3^m) - CX_i^m$ where CX_i^m is the number 1s in the i_{th} Stochastic Number after m bits have been received. The output Z equals 1 if atleast one of the input is 1 in that clock cycle and atleast one counter is zero of those inputs that are 1, else the output Z is zero. For example, in Table 2.2 for clock cycle 3, the inputs X_1, X_2, X_3 are $0, 0, 1$ respectively where as the counter values (from previous clock cycle 2) are $0, 1, 2$ does not the satisfy the condition of Z, thus the output Z equals 0. The counter values of cycle 3 are not used in the clock cycle 3 for computing Z as the counter values are yet to be computed at the end of 3rd clock cycle. As seen in the Table 2.2 the final output Z is the maximum number among the three inputs. The arrangement 1s and 0s of Z need not be the same as the maximum number of the input, but Z contains same number of 1s and 0s from the input maximum number. The proposed NMax Circuit produces exact maximum and not approximate values, in addition it is insensitive to correlation.

Stochastic Fully Connected Layer: FC layers in a NN consists of Multiplication of inputs with weights and addition of bias. As discussed above, the multiplication in Stochastic can be performed using XNOR for bipolar numbers (Fig 2.19).

SCNN Overview: In this section, each layer in the CNN architecture is replaced with its counter part in the stochastic domain. With increase in stochastic blocks in a CNN specifically stochastic addition blocks, the SN length significantly increases in order to provide approximately the same accuracy as the non-SCNN. In the next section, the various attack types and specially Deep Fool Attack is discussed. Deep fool attack is used to attack the SCNN and non-SCNN to identify the benefits of introducing a

stochastic layer in a NN over a standard NN. This provides an insight into the robustness of the SCNN from attacks during real world scenarios.

2.4 Adversarial Attacks

With advancement and use of Neural networks in various safety critical applications like autonomous driving, face recognition etc., safety has become a top priority in such systems. These systems are vulnerable to adversarial attacks where the input images are slightly modified in order to fool the trained neural network. These type of attacks are very common and have been difficult to defend against. The modifications or perturbations in the image are so small which sometimes are not perceivable the human eye. Such type of attacks are called adversarial attacks.

The overview of the types of adversarial attacks are as follows:

1. **Targeted Attacks** : Attacks which generate perturbed images in order to make the NN classify it as the specific required class and not the original class.
2. **Non-Targeted Attacks**: Attacks which makes sure the Neural Network wrongly classifies the class other than original class.
3. **White Box attack**: Perturbed images are generated after knowing the entire CNN architecture, parameters etc.
4. **Black Box Attack**: Attacks that do not have any information about the model.

2.4.1 Deep Fool Attack

Deep Fool Attack [9] is an adversarial attack which computes the perturbations for an image and given model efficiently which fool the deep neural networks. Deep Fool Attack calculates the minimum perturbations by which the class predicted by the model changes compared to that of Fast Gradient Sign Method(FGSM) [9]. An example shown in the Fig 2.21 shows the difference in perturbations FGSM and Deep Fool.

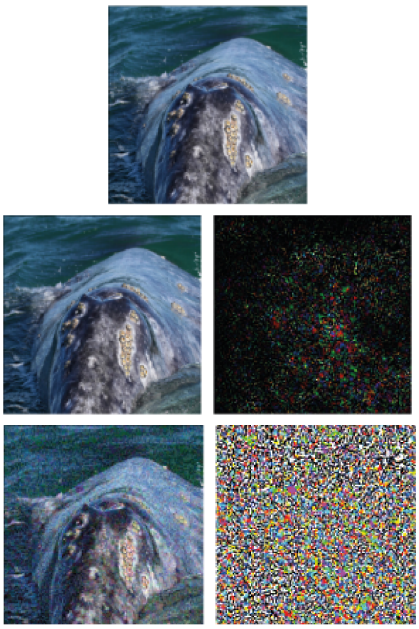


Figure 2.21: First row consist of the original image of whale. The second row consists of the perturbed image and perturbations generated by deep fool attack. Third row depicts the perturbations generated by Fast Gradient Sign Method(FGSM)[22]

Deep fool attack first identifies the output class of predicted by the model, following which the classifier of each class is identified using the straight line equation $f(x) = w^T x + b$. For example, consider there are three hyper planes for classification and point x_0 is inside the green area as shown in Fig 2.22. The Deep fool attack first identifies the least distance to the nearest hyperplane which here is F_2, and at each iteration tries to push the point x_0 on the other side of F_2 by making minor changes to the input image. Once the image is wrongly classified the loop stops and the perturbed image is obtained.

The affine classifier is given by $f(x) = w^T x + b$ where for a given weight w and bias b. $\hat{k}$ is the outcome of one vs all classification, then the minimum perturbation to fool the neural network is given as

$$\underset{r}{\operatorname{argmin}} ||r||^2 \quad s.t. \quad \exists k : w_k^T(x_0 + r) + b_k \geq w_{\hat{k}(x_0)}^T(x_0 + r) + b_{\hat{k}(x_0)} \tag{2.10}$$

where w_k is the k^{th} column of w. The above problem corresponds to the computation of a distance between x_0 and the complex polyhedron P. In the

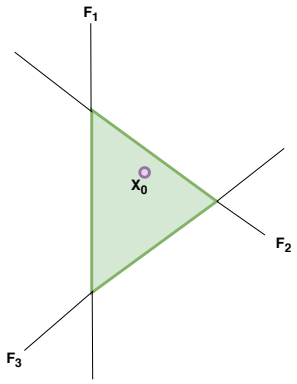


Figure 2.22: First row consist of the original image of whale. The second row consists of the perturbed image and perturbations generated by deep fool attack. Third row depicts the perturbations generated by Fast Gradient Sign Method(FGSM) [22]

Fig 2.22 the green area is the polyhedron P. The polyhedron P defines the region of space where f outputs the label $\hat{k}(x_0)$.

The solution to the above equation 2.10 can be computed as follows.

$$r_*(x_0) = \frac{|f_{\hat{l}x_0}(x_0) - f_{\hat{k}(x_0)}(x_0)|}{||w_{\hat{l}(x_0)}(x_0) - w_{\hat{k}(x_0)}(x_0)||_2^2}(w_{\hat{l}(x_0)}(x_0) - w_{\hat{k}(x_0)}(x_0)) \tag{2.11}$$

Where $l(x_0)$ is the closest hyperplane boundary to the classified point x_0. The minimum perturbation $r(x_0)$ is the vector that projects x_0 on the nearest hyperplane indicated by $\hat{l}(x_0)$ i.e. we find the closest plane to x_0. Basically $\hat{l}_0$ gives the distance to the nearest hyperplane where the output class $\hat{k}$ is not same as the original $\hat{k}(x_0)$.

$$\hat{l}(x_0) = \underset{k \neq \hat{k}(x_0)}{\operatorname{argmin}} \frac{|f_{\hat{k}}(x_0) - f_{\hat{k}(x_0)}(x_0)|}{||w_{\hat{k}} - w_{\hat{k}(x_0)}||_2} \tag{2.12}$$

Deep Fool Attack can also be extended to non-linear multiclass classifier. For multiclass non linear classifiers the set P describes the region where the classifiers outputs label $k(x_0)$ and the region is not polyhedron with non-linear boundaries. The algorithm for Deep fool: multi class classifier is shown in Algorithm 3.

Algorithm 3: Deep Fool: Multi Class Case

1 **input:** Image x, classifier f.
2 **output:** Perturbation $\hat{r}$
3
4 initialize $x_0 \leftarrow x, i \leftarrow 0$.
5 **while** $\hat{(}k)(x_i) = \hat{k}(x_0)$ **do**
6 **for** $k \neq \hat{k}(x_0)$ **do**
7 $w'_k \leftarrow \nabla f_k(x_i) - \nabla f_{\hat{k}(x_0)}(x_i)$
8 $f'_k \leftarrow f_k(x_i) - f'_{\hat{k}(x_0)(x_i)}$
9 **end**
10 $\hat{l} \leftarrow \text{argmin}_{k \neq \hat{k}(x_0)} \frac{|f'_k|}{||w'_k||_2}$
11 $r_i \leftarrow \frac{|f'_{\hat{l}}|}{||w'_{\hat{l}}||_2^2} w'_{\hat{l}}$
12 $x_{i+1} \leftarrow x_i + r_i$
13 $i \leftarrow i + 1$
14 **end**
15 **return** $\hat{r} = \sum_i r_i$

The algorithm 3 at line 4 initialises the input x to x_0 and iteration variable $i = 0$. Line 5 compares the current iterated output label $\hat{k}(x_i)$ of the network to the initial stored output label $\hat{k}(x_0)$, if they are same then more perturbations are added over the input image. Line 6 is for loop which loops for every class other than the original class. Line 7 and 8 modifies the weights of each class by using the affine function of that class to calculate the distance between the point and the hyperplane. This is done for every plane present in the network which can misclassify the network. Line 10 with the help of the change in weights and affine function values in the line 8, the loss $\hat{l}$ is computed which provides the hyperplane which has the least distance among all the hyperplanes. Line 11 updates the perturbation value with respect to the new hyperplane using which has the least distance from the point from line 10. The ratio of f and $||w||_2^2$ basically gives the distance in the direction perpendicular to the hyperplane. Once the perturbation is computed, it is added to the input image on line 12 and the loop value i is incremented on line 13.

Chapter 3

Experiment

This chapter explains the experimental setup that is used in this thesis. There are 3 datasets which have their own respective DNN for classification. Depending on the dataset, the network complexity varies. The experiment uses Tensorflow 2.0 [23] for training various neural networks and NumPy [24] for development of SCNN. There are several permutation and combinations of the activation function, SC layers, SN length, adversarial attack parameter that are performed in order to test multiple scenarios. With complex NN only one layer at time is replaced with SC blocks whereas for shallow NN, all the layers are replaced to SC blocks.

3.1 Dataset

3.1.1 MNIST Dataset

MNIST (Modified National Institute of Standards and Technology database) dataset [5] is database consisting of handwritten digits from 0 to 9 by different people. Each image in dataset is of size 28x28. There are two sets namely the training set consists of 60000 images and test set consisting of 10000 images. The dataset also contains the output label of each image describing the digit written in the each image. Thus the dataset consists of overall 70000 images. Sample images from the MNIST dataset is shown in the Fig 3.1

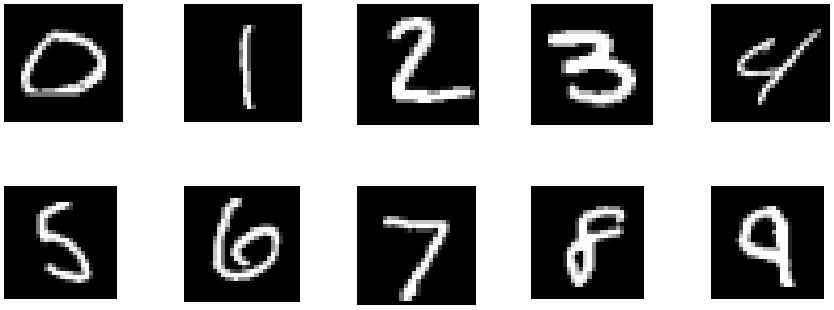


Figure 3.1: MNIST sample images from each class

3.1.2 Fashion MNIST Dataset

Fashion MNIST dataset [6] is a collection of Zalando's article images. This dataset is similar to MNIST dataset but each consists of different apparels of fashion and slightly more complex to the MNIST dataset. Fashion MNIST contains more details and features over the simple handwritten MNIST. The dataset consists of 60000 training images and 10000 test images. Each image is of size 28x28. The sample images from Fashion MNIST dataset is shown in the Fig 3.2

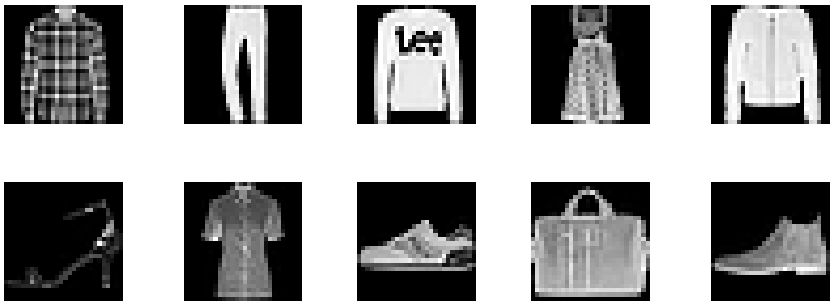


Figure 3.2: Fashion MNIST sample images from each class

3.1.3 CIFAR-10 Dataset

CIFAR-10 (Canadian Institute For Advanced Research) [7] is a large dataset that consists of images 10 different classes namely airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. CIFAR-10 dataset is far more complex compared to MNIST and Fashion MNIST due to presence of RGB colours and wide variety of classes which almost have no resemblance with each other. The dataset consists of 60000 images with each class having 6000 images. Each image in the dataset is a colored RGB image with size 32x32x3

where 3 is the number of channels representing RGB. The sample images from CIFAR-10 dataset is shown in the Fig 3.3

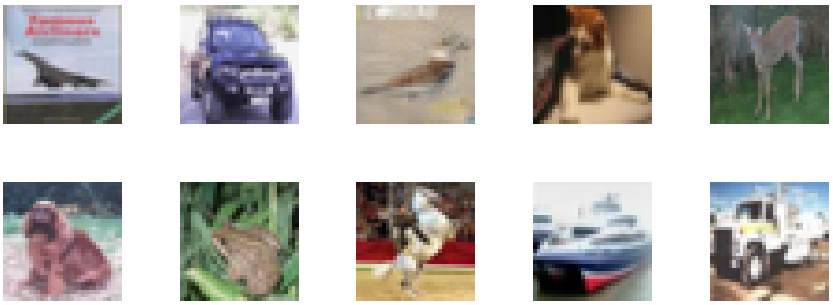


Figure 3.3: CIFAR-10 sample images from each class

3.2 CNN Architecture

3.2.1 MNIST CNN Architecture

The CNN architecture of MNIST consists of single Convolution layer, custom activation layer, max pooling layer, fully connected layer and finally the softmax output layer. The reason for the network being simple is due to dataset being non complex. The CNN architecture for MNIST is as shown in Fig 3.4.

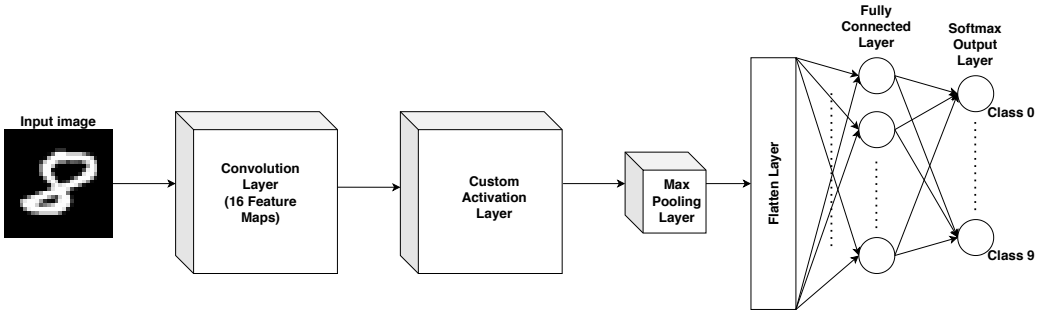


Figure 3.4: MNIST CNN Architecture with Convolutional layer, activation layer, maxpooling layer and FC layer

The input images are of the size 28 x 28 and grayscale. Since the images are grayscale there is only one channel. Each image ranges from 0-255 where 0 is black and 255 being white. The pixel values of each image are normalized between [0, 1] since SNs operate between the same range.

In the standard binary version of the CNN architecture, the network is trained using the input image of size 28 x 28 of MNIST dataset. The Input image is fed to the convolutional layer followed by custom activation function in equation 2.9. The activation function is scaled by a value 0.4 in order to have the same affect as activation function value during stochastic computation. Further the output is passed through the 2 x 2 Max pooling layer which reduces the size of the output by 75%. The output of max pooling layer is flattened and provided as input to the fully connected (FC) layer. In the FC layer, multiplication of the input with the weights is performed and the output is provided to the final softmax layer which decides the class of the input image. The error generated in classification is the propagated to all the layers in order to learn the weights during the training phase.

The Stochastic Convolutional layer consists of the 28 x 28 input matrix and 3x3 kernel. The image of size 28 x 28 is converted to SNs where each number is passed through SN Generator to obtain bipolar SNs. Let n_{len} be the SN length, the size of the image changes from 28 x 28 to 28 x 28 x n_{len}. The kernel is passed through SN generator to obtain the kernel with size 3 x 3 x n_{len}. The two inputs are then multiplied with each other using XNOR operation and finally added using MUX based operation.

The custom Tanh activation layer is upscaled by a value of 4 and downscaled by a value of 10. The constant divider is due to the upscaling factor by stochastic Tanh and the downscaling factor from stochastic addition. The stochastic Tanh has 8 states hence provides an upscaling value of 4 and the stochastic additions has 10 additions where 9 additions of 3 x 3 kernel and 1 bias thus providing a downscaling factor of 10. The activation function is multiplied with constant in order to incorporate the scaling during the training using Tensorflow [23]. The binary Tanh is multiplied with a constant upscale factor/downscale factor as in equation 3.1.

$$A = tanh(x * (upscalefactor/downscalefactor))$$

$$A = tanh(x * 4/10) = tanh(x * 0.4) \tag{3.1}$$

The Output of the activation layers is fed as input to the Max pooling layer which is implemented using the NMax Function discussed in the Section 2.3. The output of the Max pooling layer is flattened and provided as an input the Fully Connected layer. The Fully Connected layer multiplies the input with corresponding weights and provides the output to the next layer with

Layer Name	Parameters	Output Shape
Convolutional Layer	Kernel Size = 3 x 3 x 16	26 x 26 x 16
Custom Activation Layer	tanh (x / 2.5)	26 x 26 x 16
Max Pooling Layer	2 x 2	13 x 13 x 16
Flatten Layer	-	(None, 2704)
Fully Connected Layer	Number of Neurons = 2704	(None, 2704)
Softmax Output Layer	Number of Neurons = 10	(None, 10)

Table 3.1: Architecture Summary of CNN used for MNIST dataset

10 nodes. The output class is determined by the maximum value of the 10 neurons at the final layer. The final layer is known as softmax layer which provides a probability value for each particular class. For example, if neuron 2 in the last layer has a value 0.83, it suggests that the input is classified as handwritten digit 2 with 83% probability/confidence. The Multiplication in the Fully Connected network for stochastic version is implemented using XNOR operation. The architecture table of the CNN configuration used for MNIST dataset is given in Table 3.1

3.2.2 Fashion MNIST CNN Architecture

Fashion MNIST dataset is similar to handwritten MNIST digit dataset but with more complex features. The MNIST CNN does not provide sufficient accuracy when trained for Fashion MNIST dataset. The Fashion MNIST CNN Architecture consists of 2 Convolutional Blocks where each block consists of one convolutional layer, activation function layer, max pooling layer and dropout layer. The first convolutional block has a 32 feature map convolutional layer, Tanh activation, 2 x 2 maxpooling layer and a dropout rate of 0.2. In the second convolutional block, the convolutional layer has 64 feature maps and a dropout rate of 0.3 and rest being the same. The two convolutional block is followed by a flattening layer and an output layer consisting of a softmax activation function to determine the final class. The CNN is shown in the Fig 3.5. The architecture details of the CNN used in Fashion MNIST training is provided in the Table 3.2.

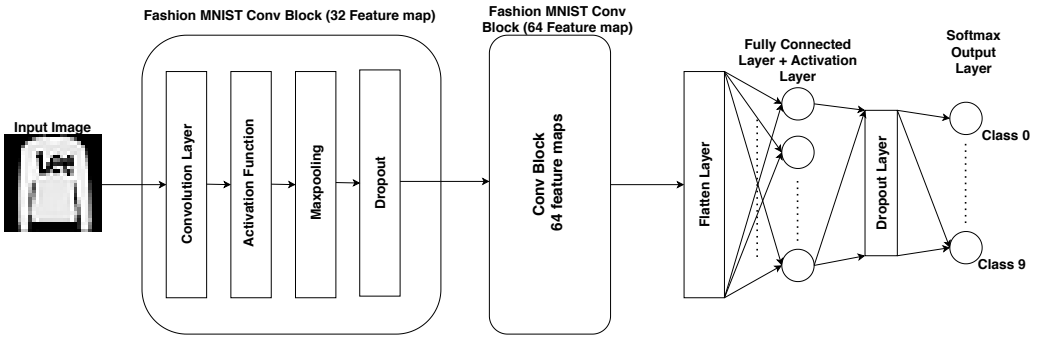


Figure 3.5: Architecture of Fashion MNIST CNN

Layer Name	**Parameters**	**Output Shape**
Convolutional Layer	Kernel Size = 3 x 3 x 32	28 x 28 x 32
Activation Layer	tanh/ReLU	28 x 28 x 32
Max Pooling Layer	2 x 2	14 x 14 x 32
Dropout	Dropout rate = 0.2	14 x 14 x 32
Convolutional Layer	Kernel Size = 3 x 3 x 64	14 x 14 x 64
Activation Layer	tanh/ReLU	14 x 14 x 64
Max Pooling Layer	2 x 2	7 x 7 x 64
Dropout	Dropout rate = 0.2	7 x 7 x 64
Flatten Layer	-	(None, 3136)
Fully Connected Layer	Number of Neurons = 64	(None, 64)
Activation Layer	tanh/ReLU	(None, 64)
Softmax Output Layer	Number of Neurons = 10	(None, 10)

Table 3.2: Architecture Summary of CNN used for Fashion MNIST dataset

3.2.3 CIFAR-10 CNN Architecture

CIFAR-10 being a colored image and more complex dataset compared to MNIST. The CIFAR-10 CNN requires more layers in order to capture different features. The CNN architecture of CIFAR-10 dataset has 3 Convolutional blocks, 1 Fully connected layer and softmax layer at the output end. Each convolutional block has 2 convolutional layer, 2 activation functions, one max pooling layer and one dropout arranged as shown in the Fig 3.6. The overall structure of the CNN has 6 convolutional layers with each pair of layers having same feature maps as in Fig 3.7.

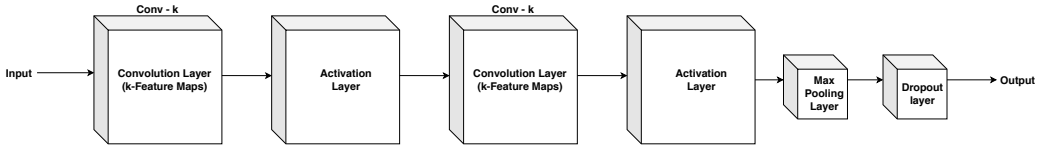


Figure 3.6: Architecture of Single Convolutional block

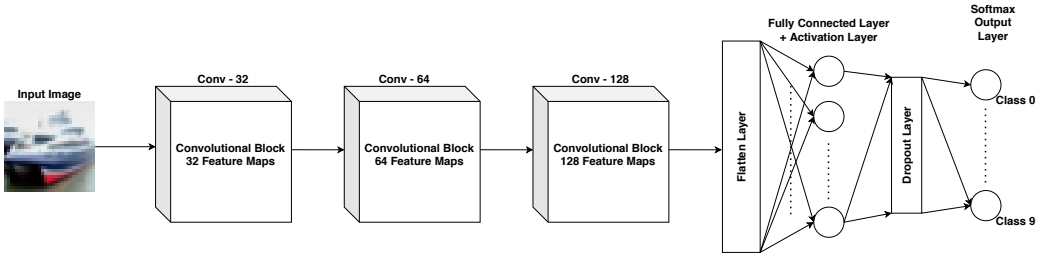


Figure 3.7: Architecture of CNN Architecture used for CIFAR-10 Dataset

The input image from the CIFAR dataset is applied to the first convolutional block which has a kernel size of 3 x 3 x 32 i.e the first block has 32 feature maps. The input is padded with zeros before input to each convolutional layer to maintain the size of the image. The second convolutional block has a kernel matrix of size 3 x 3 x 64 where as the third convolutional block has a kernel size matrix of 3 x 3 x 128. After each convolutional block a drop out layer is introduced in order to avoid overfitting. As the number of layers increase, there is high probability of overfitting. The dropout rate should increase as the depth of the network increases. This is followed by a fully connected layer where the input from the third convolutional block is flattened before it is applied to fully connected layer. The FC layer is followed by a dropout layer which then is connected to the final output softmax layer. The architecture details of the CNN is provided in the Table 3.3. The rows without the horizontal line belong to one convolutional block as shown in Fig 3.7

Since the number of layers are many, replacement of all blocks with stochastic blocks will take significant amount of computation time, power and requires a higher stochastic length due to multiple scale down by addition which loses the purpose of using stochastic blocks. In this thesis, we replace one convolution layer of one block with stochastic convolutional layer and after which the stochastic numbers are converted back to binary number and passed through

Layer Name	Parameters	Output Shape
Convolution Layer 1	Kernel Size = (3 x 3 x 32)	32 x 32 x 32
Activation Layer 1	tanh/ReLU	32 x 32 x 32
Convolution Layer 2	Kernel Size = (3 x 3 x 32)	32 x 32 x 32
Activation Layer 2	tanh/ReLU	32 x 32 x 32
Max Pooling Layer 1	Pooling Size = 2 x 2	16 x 16 x 32
Dropout Layer 1	Dropout rate = 0.2	16 x 16 x 32
Convolution Layer 3	Kernel Size = (3 x 3 x 64)	16 x 16 x 64
Activation Layer 3	tanh/ReLU	16 x 16 x 64
Convolution Layer 4	Kernel Size = (3 x 3 x 64)	16 x 16 x 64
Activation Layer 4	tanh/ReLU	16 x 16 x 64
Max Pooling Layer 2	Pooling Size = 2 x 2	8 x 8 x 64
Dropout Layer 2	Dropout rate = 0.3	8 x 8 x 64
Convolution Layer 5	Kernel Size = (3 x 3 x 128)	8 x 8 x 128
Activation Layer 5	tanh/ReLU	8 x 8 x 128
Convolution Layer 6	Kernel Size = (3 x 3 x 128)	8 x 8 x 128
Activation Layer 6	tanh/ReLU	8 x 8 x 128
Max Pooling Layer 3	Pooling Size = 2 x 2	4 x 4 x 128
Dropout Layer 3	Dropout rate = 0.3	4 x 4 x 128
Flatten Layer	-	(None, 2048)
Fully Connected Layer	Number of Neurons = 128	(None, 128)
Activation Layer 7	tanh/ReLU	(None, 128)
Dropout Layer 4	Dropout rate = 0.4	(None, 128)
Softmax Output Layer	Number of Neurons = 10	(None, 10)

Table 3.3: Architecture Summary of CNN used for CIFAR-10 dataset

the standard CNN layers. This is known as a hybrid circuit which is a combination of binary and SCNN.

Due to large number of additions which will affect the accuracy, addition is processed as binary addition and multiplication is alone executed in stochastic. However there are multiple tried and tested combinations in order to provide the best results. The main intention of choosing the initial convolutional layers to be executed in stochastic is to nullify the perturbations of the input image at starting stages instead of magnifying the errors due to addition and multiplication after passing through multiple layers. Several combinations of activation functions such as Tanh, ReLU are tested in stochastic domain. In the experiment, each convolutional layer is replaced with stochastic convolutional layers one at a time. The results for each convolutional layer when replaced with stochastic layers are provided in the result section 4.4. Several other combinations of the stochastic versions are experimented with such as replacing only maxpooling block with the NMax function.

Training and evaluation of SCNN

Architecture 1 of MNIST: The CNN for simple data MNIST is trained using Tensorflow 2.0. The weights of the layers to be converted to stochastic layers are scaled between [-1, 1] after every batch end. The labels of the train and test data set are converted to one hot encoding. For example, for image label of 3 out of 10 classes, the one hot coding value is [0,0,1,0,0,0,0,0,0,0]. The CNN is trained using Adadelta optimizer with a learning rate of 1.0 with 12 epochs and batch size of 64. The weights are initialised having Random Normal distribution with mean 0.0 and standard deviation of 0.05. The number of epochs can be increased to 20 for MNIST. The NN is trained over 50 epochs to provide a train accuracy of 93.61% and test accuracy of 93.37% with Tanh as activation function.

The trained network weight are exported in order to build a stochastic network with the same weights for testing the stochastic CNN compared to the standard CNN. The stochastic layers are built using NumPy. The Stochastic CNN is tested for first 500 images of the MNIST dataset.

Architecture 2 of Fashion MNIST: The Fashion MNIST CNN shown in the Fig 3.5 is built using Tensorflow 2.0. The NN is trained over 50 epochs to provide a train accuracy of 94.02% and test accuracy of 91.17%. This shows that the model works well without overfitting. The addition of dropout layers

prevents the model from overfitting. Similar to the MNIST digit dataset, the weights are scaled between the value [-1, 1] for the layers where stochastic computation is to be applied. The main optimizer used is Adadelta with a learning rate of 1. The trained network weights are then exported to the SCNN in order to test the network against attacks.

Architecture 3 of CIFAR-10: The preprocessing of input for CIFAR-10 is similar to MNIST dataset. The major difference is the number of epochs required for CIFAR-10 is 200 to 250. The CNN model of CIFAR-10 tends to overfit a little with training accuracy of 85.5% and test accuracy of 81.3%. The activation function used is Tanh along with different dropout values for each convolutional block. Similarly, ReLU activation function along with He-Normal weight initialization for the same network provides train accuracy of 85.94% and test accuracy of 85.93%.

Similar to MNIST model, the weights of the CNN of CIFAR-10 are exported to stochastic CNN for further testing. The custom activation function is not used here since the addition and activation function are in binary version where as only multiplication takes place in stochastic domain which does not upscale or downscale the values. Since the number of feature maps are significantly high compared to MNIST CNN, the SN length required tends to increase for obtaining an acceptable accuracy. Each convolutional layer is replaced with stochastic convolution with only stochastic multiplication and addition computed in binary domain. When there is one stochastic convolutional layer in the network, the rest of the convolutional layers are in non-stochastic domain. This helps determine which layer provides the best robustness when attacked by an adversarial attack. However the stochastic block is not only restricted to convolutional layers but each max pooling layer is replaced with NMax function in the stochastic domain. Each max pooling layer is replaced with NMax function one at a time which helps in providing a clear picture about the security the stochastic layer provides against adversarial attack.

Chapter 4

Results

This chapter encompasses the various results of all the permutation and combination of SCNN and CNN with Deep fool adversarial attacks performed over them. The results obtained showcase the robustness provided by the SCNN over a normal CNN. In general, when perturbed images generated by Deep fool using FoolBox [25] library is passed onto CNN and SCNN, the attack is able completely misclassify the images in CNN with accuracy dropping to 0% whereas SCNN is able to withhold from 10-30% depending on the dataset and complexity of the network.

4.1 Evaluation Criteria

The robustness of the networks is determined using accuracy before the attack, accuracy after the attack and success rate of the attack for the CNN and SCNN.

Accuracy: Accuracy of NN is defined as the ratio of the number of correctly classified images over total number of images into 100. The formula is as shown below in equation. 4.1

$$Accuracy = \frac{number\ of\ correctly\ classified\ images}{total\ number\ of\ images}\ .\ 100 \qquad (4.1)$$

Success rate of the attack: It is defined as

$$Successrate = \frac{accuracy\ before\ attack\ -\ accuracy\ after\ attack}{accuracy\ before\ attack}.100 \quad (4.2)$$

4.2 Deep Fool attack on MNIST

The MNIST SCNN network for Deep Fool attack consists of 1 convolutional layer with stochastic multiplication and addition followed by Stanh activation function and NMax stochastic max pooling layer and finally one fully connected layer implemented with stochastic adder and multipliers. The perturbed image sample of the MNIST dataset after the Deepfool attack are shown in Fig 4.1. The perturbations on the images are very small and sometimes non perceivable to the human eye but the CNN misclassifies the labels even for such small perturbations.

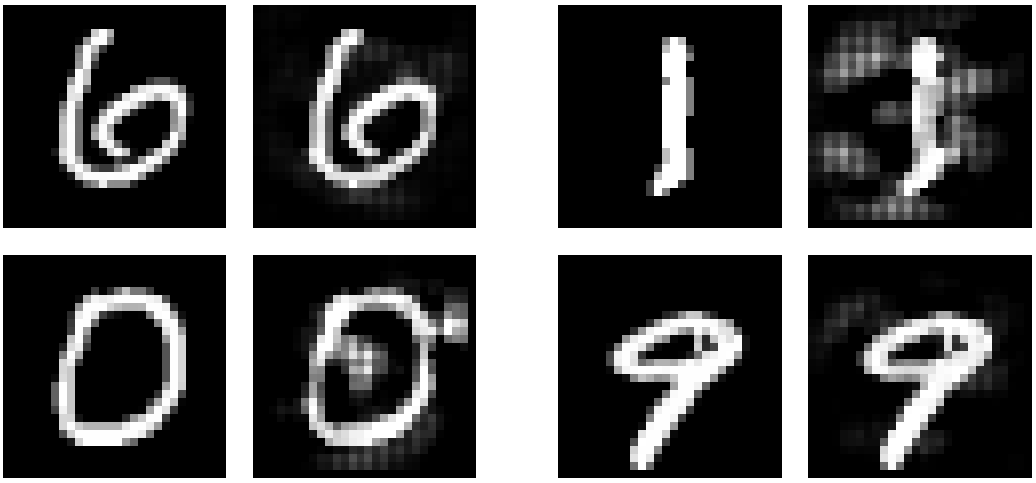


Figure 4.1: MNIST images perturbed by Deepfool Attack at 0.1 overshooting parameter.

Each attack has a overshooting parameter. For example, in Fig 4.2, the overshoot parameter is 0.01 which means the image is perturbed until the data point crosses the hyperplane to classify as another class by a l_2 distance of 0.01. From the below results we see that the binary network completely fails due to perturbations whereas the SCNN is able to retain upto 13% accuracy even after deep fool attack. At perturbations of 0.01 and 0.001 which is close to the hyperplane, we tend to see better accuracy compared to the other results. This is due to the fact that there are very small perturbations added to the input image since the overshoot value is just 0.01 and 0.001. The smaller the SN length we tend to see better robustness since the computations becomes approximate. The results for this case is shown from Fig 4.2 to Fig 4.6

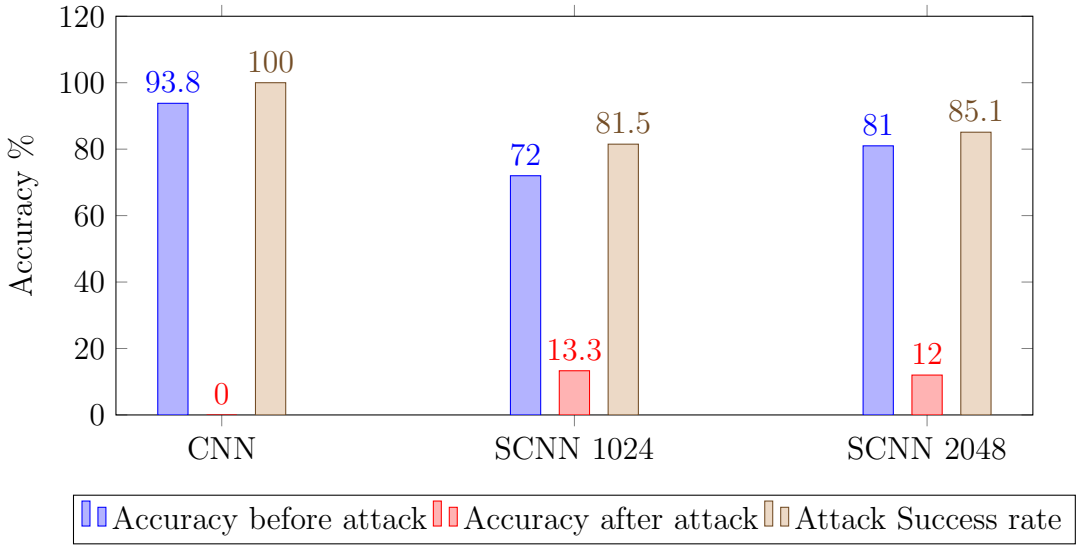


Figure 4.2: MNIST fully stochastic network - Attack at 0.01

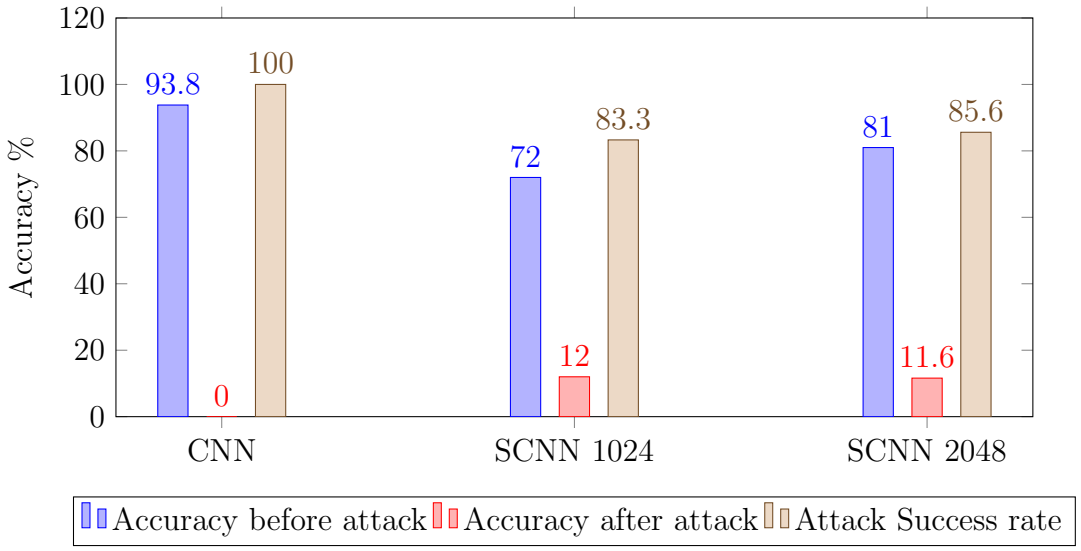


Figure 4.3: MNIST fully stochastic network - Attack at 0.001

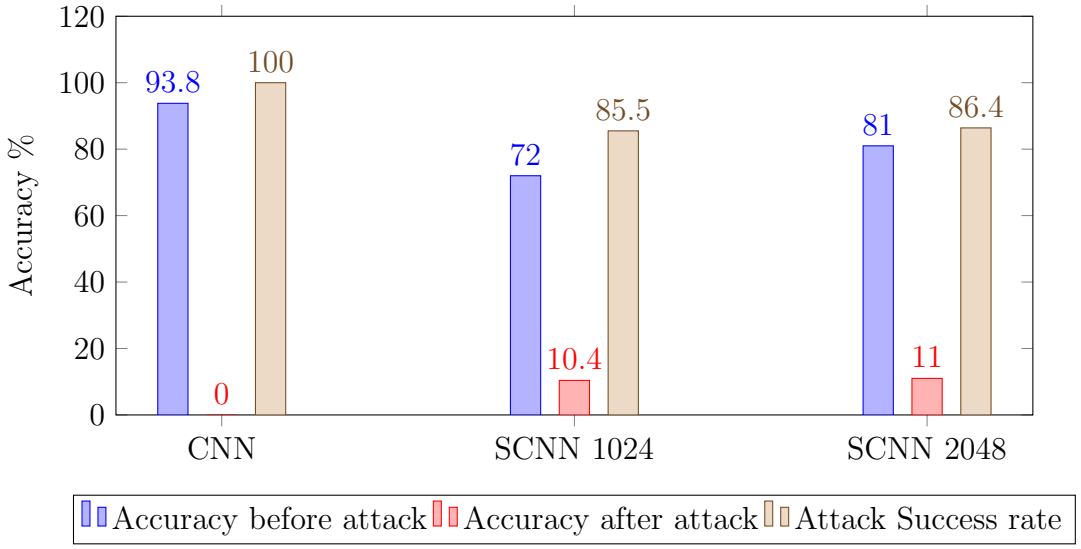


Figure 4.4: MNIST fully stochastic network - Attack at 0.0

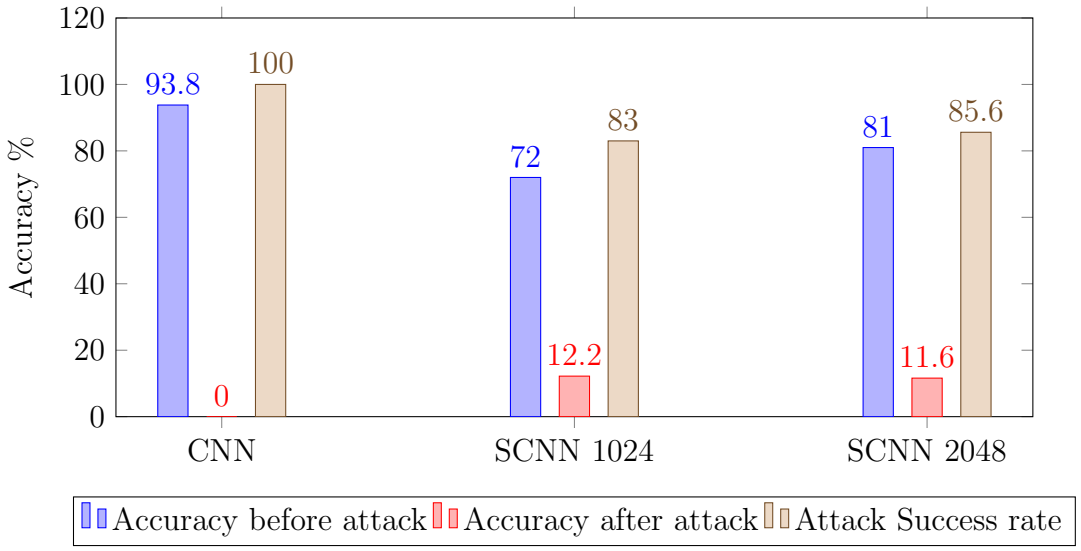


Figure 4.5: MNIST fully stochastic network - Attack at 0.1

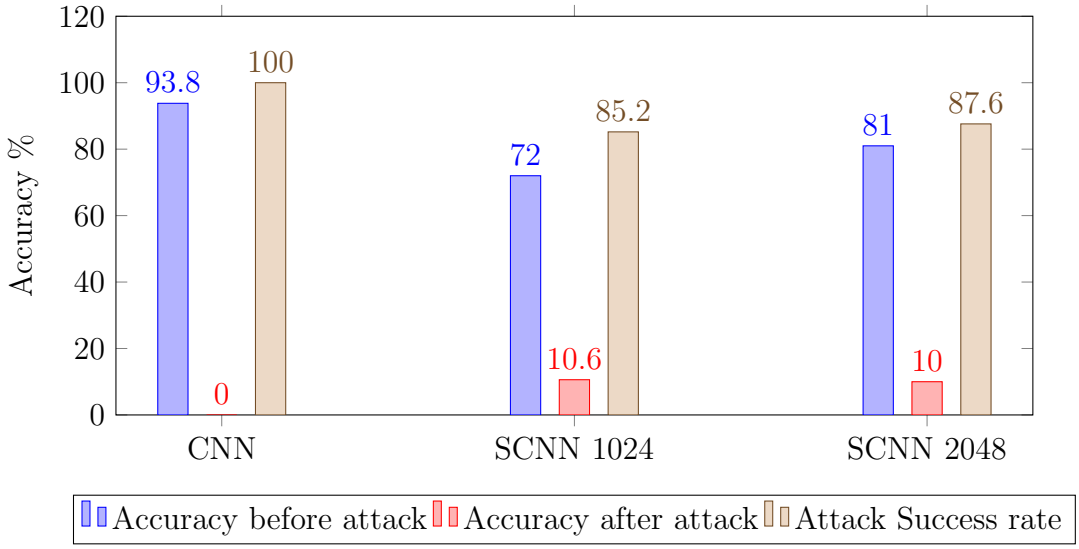


Figure 4.6: MNIST fully stochastic network - Attack at 0.5

The accuracy of the network in the first experiment is similar at 12-13% especially when the attack overshooting parameter is at 0.01 and 0.001. When the image is perturbed beyond the two parameters we see a drop in accuracy to 10% this is due to the boundary being away by a higher margin.

The second type of MNIST network being tested is similar to the above fully stochastic network but the addition and activation function in the convolutional layer are replaced by the Btanh activation function. This function combines the operation of addition and activation function. The addition is computed in binary domain and activation function in the stochastic domain. This type of network when attacked with the Deep fool, the network does not provide much robustness due the introduction of exact computations. We see that after the attack as usual the binary CNN completely fails whereas the SCNN with Btanh is able to withhold 8-10% accuracy. Here since the addition is not scaled there is an increase in accuracy in the stochastic network and comes close to the binary network. In the Btanh SCNN, we are able see better accuracy for the same overshooting values of 0.001 and 0.01 as the previous fully stochastic network. The results for Btanh SCNN is shown from Fig 4.7 to 4.10.

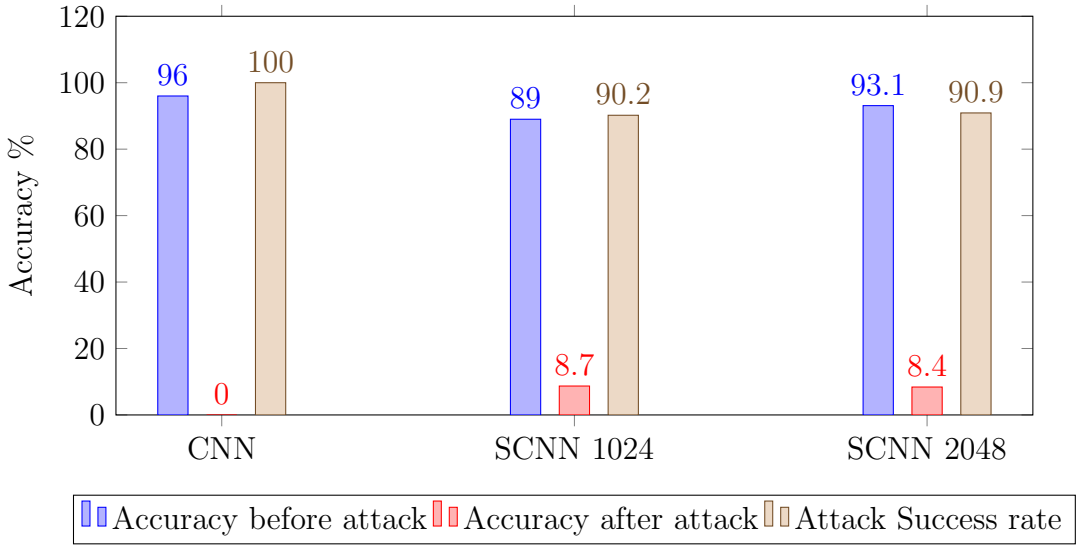


Figure 4.7: MNIST fully stochastic network with Btanh - Attack at 0.01

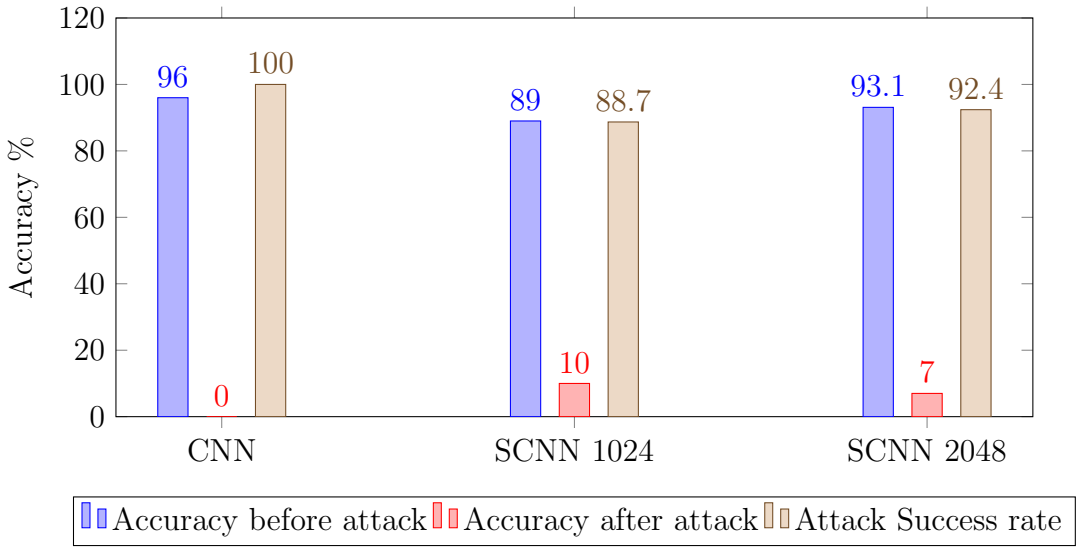


Figure 4.8: MNIST fully stochastic network with Btanh - Attack at 0.001

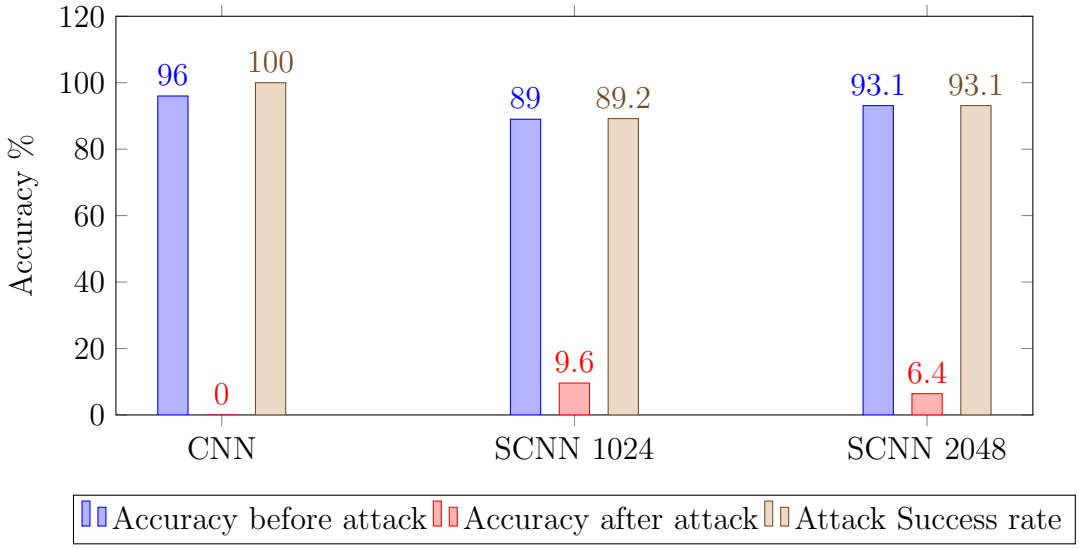


Figure 4.9: MNIST fully stochastic network with Btanh - Attack at 0.1

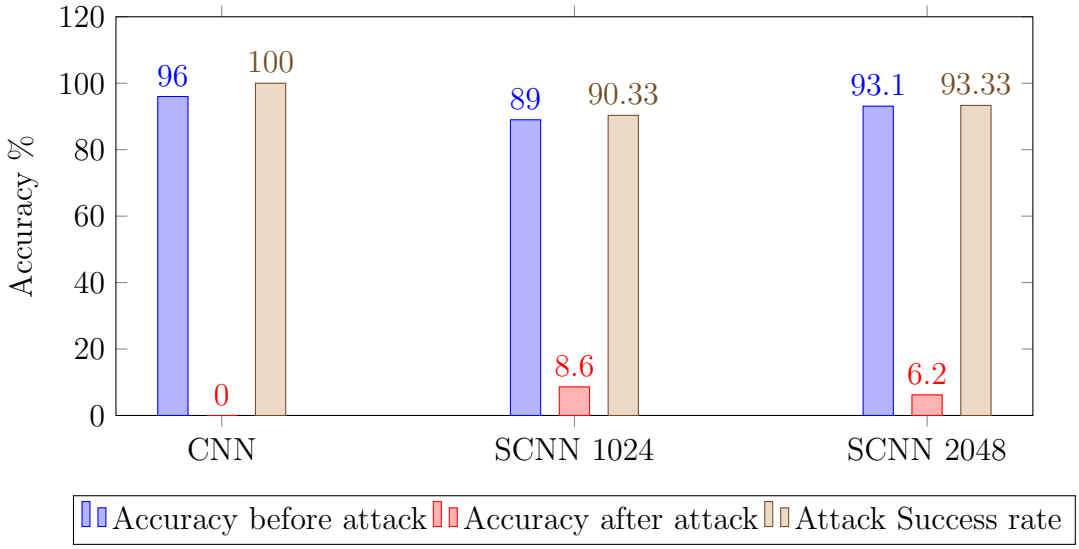


Figure 4.10: MNIST fully stochastic network with Btanh - Attack at 0.5

The second experiment results do not provide significant resistance to the attack. Since the Btanh function is performing a binary addition instead of the stochastic addition seen in the first experiment of MNIST, the randomness features provided at the addition layer decreases and thus the overall

accuracy at each of the overshooting parameter is similar to 8.5 to 10%.

The third type of MNIST network under consideration is the fully stochastic network with ReLU activation function instead of the Stanh function. The Stochastic ReLU can be obtained using the NMax function. Since the NMax function provides the maximum of n stochastic numbers, it can be used to provide maximum of (0, input value) thus implementing ReLU in stochastic domain. The MNIST SCNN with ReLU provides the best robustness compared to Stanh and Btanh activation functions. As usual, the binary network completely fails with an accuracy of 0% after the deep fool attack. The SCNN is able to provide an accuracy of 25-27% after the attack. The attack was performed only for the overshoot parameter of 0.01 and 0.001 as the resistance is significantly higher compared to other overshooting values. From all the three networks discussed, there is a general pattern of more robustness with lower SN length. Since the resolution for smaller SN length is lesser, it is an added advantage during computation since finer values tend get approximated to nearest resolution value. The results plot in the graphs in Fig 4.11 and Fig 4.12.

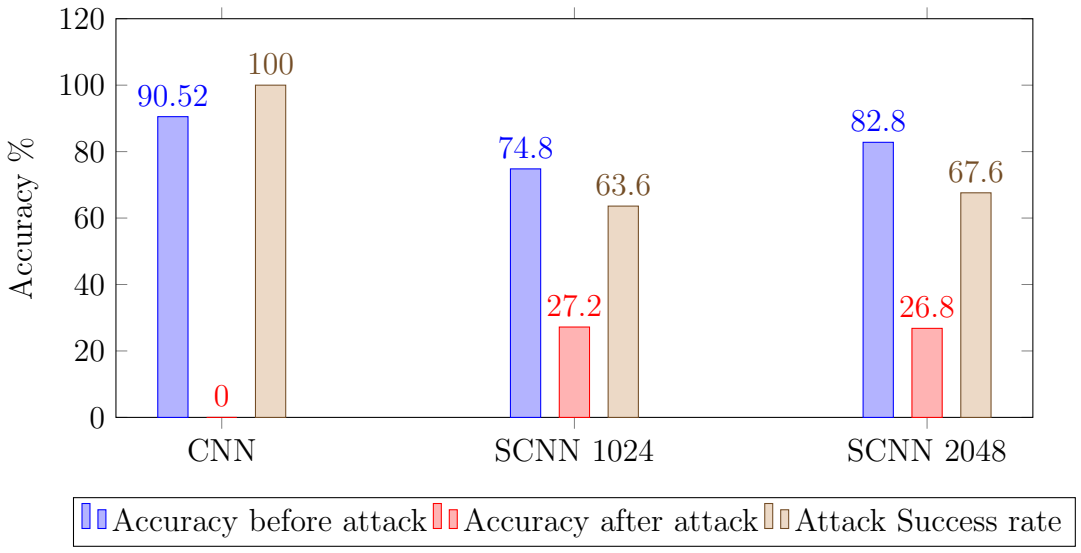


Figure 4.11: MNIST fully stochastic network with Stochastic ReLU - Attack at 0.01

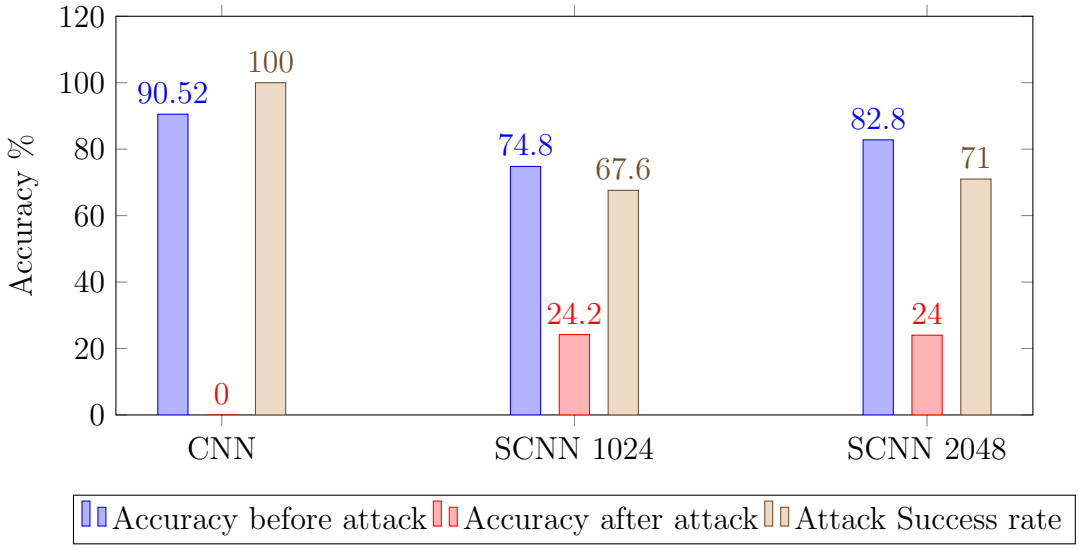


Figure 4.12: MNIST Fully stochastic network with Stochastic ReLU - Attack at 0.001

The third experiment of MNIST shows a different picture in terms of accuracy post deep fool attack. The stochastic ReLU is able to provide significant resistance to the MNIST dataset at 25-27%. In the binary version of CNN, ReLU activation function is known for preventing vanishing gradients and provide better non-linearity during learning. It also helps prevents the weights to be distributed in a narrow band. The benefits of ReLU and SN provide very good robustness for perturbed MNIST dataset.

4.3 Deep Fool attack on Fashion MNIST

The Fashion MNIST SCNN for Deep fool attack consists of 2 Convolutional blocks where each block consists of one convolutional layer, activation layer, max pooling layer and dropout layer. The 2 convolutional blocks are followed by flatten layers, fully connected layers and output softmax layer. The experiment for Fashion MNIST was performed in two ways namely (a) Performing stochastic multiplication only in the convolutional layer while keeping rest of the network in binary domain and (b) entire convolutional block in stochastic domain. Fig 4.13 sample images of the original and perturbed images with Deep fool attack at 0.1 overshooting values

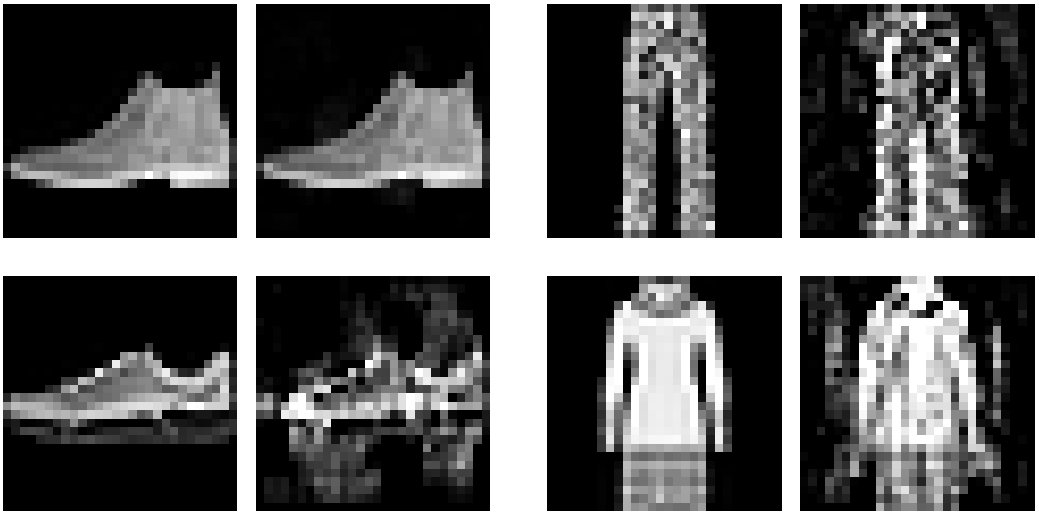


Figure 4.13: Fashion MNIST images perturbed by Deepfool Attack at 0.1 overshooting parameter.

The first experiment is performed on Fashion MNIST dataset by keeping the first convolution layer with stochastic multiplication only and retaining the rest of the network in binary domain. The attack is performed with two different overshooting parameters 0.01 (Fig 4.14) and 0.001 (Fig 4.15). The binary network gets misclassified completely whereas the SCNN performs slightly better with around 9% accuracy after attack. The lower SN length numbers usually tends to perform better after attacks compared to high SN length.

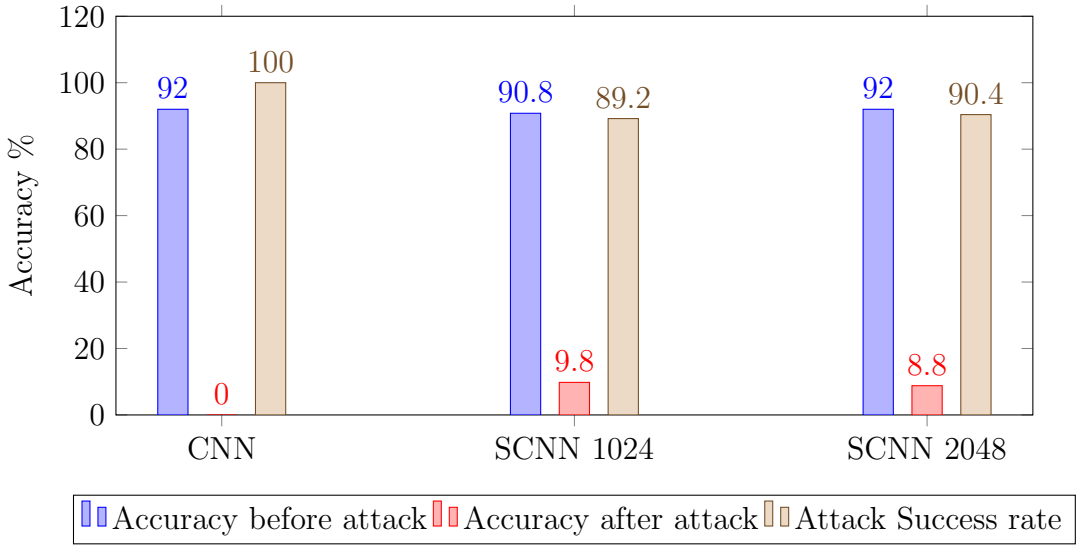


Figure 4.14: Fashion MNIST 1st layer Convolution with stochastic multiplication only - Attack at 0.01

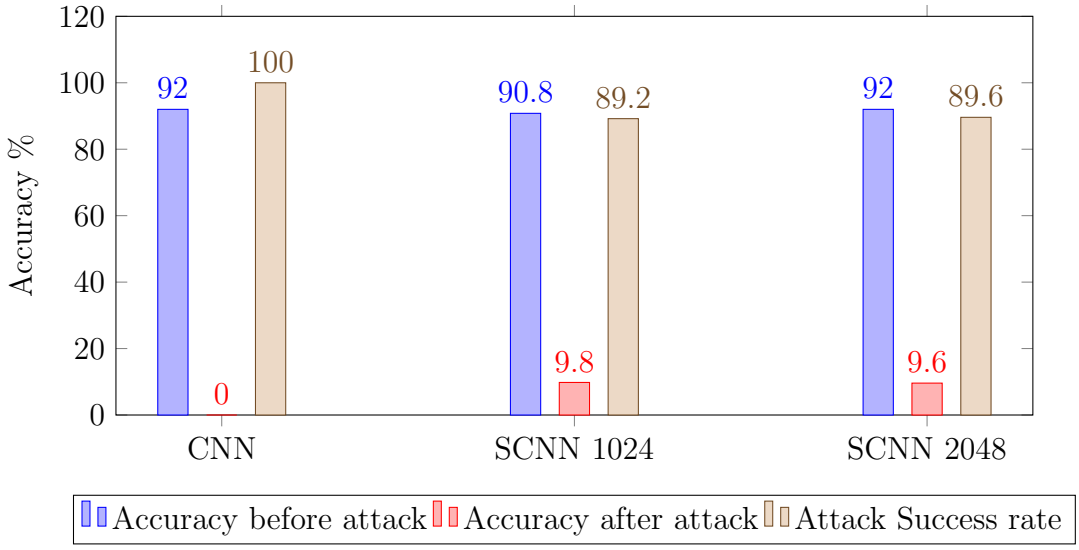


Figure 4.15: Fashion MNIST 1st layer Convolution with stochastic multiplication only - Attack at 0.001

The accuracy of the first experiment of Fashion MNIST is low in terms of resistance towards perturbed images. The dataset being slightly more complex

compared to MNIST dataset does reduce the accuracy compared to MNIST. The MNIST with similar scenario provided 10-13% accuracy in the first layer.

The second experiment is performed with the 2nd Convolutional layer having only multiplication performed in stochastic domain and keeping the rest of the network in the binary domain. Here too the usual CNN fails to classify even a single image where as the SCNN are able to classify around 10-11% of the images correctly after attack. The second layer is attacked with two parameters mainly 0.01 and 0.001 as shown in Fig 4.16 and Fig 4.17 respectively.

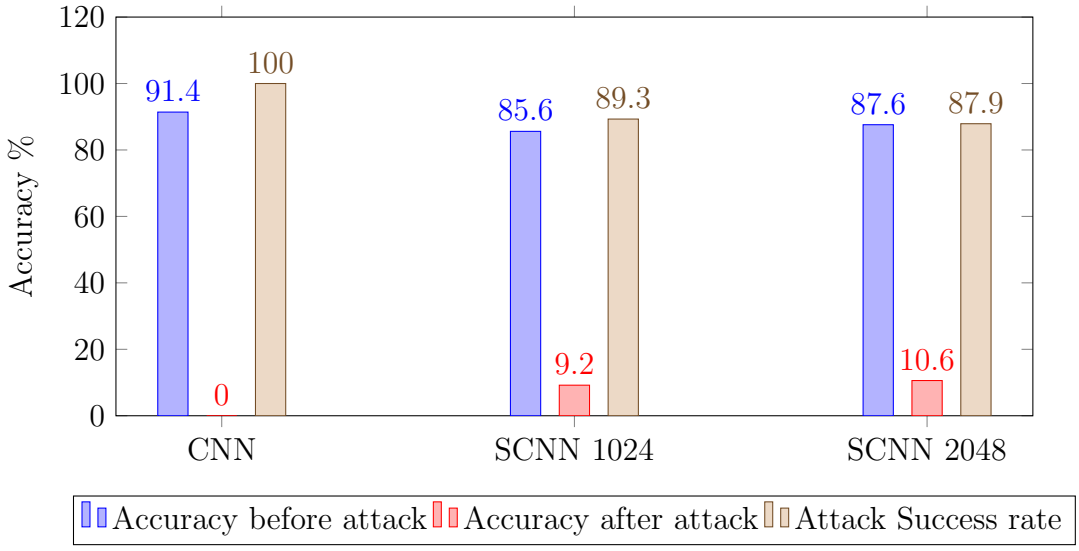


Figure 4.16: Fashion MNIST 2nd layer Convolution with stochastic multiplication only - Attack at 0.01

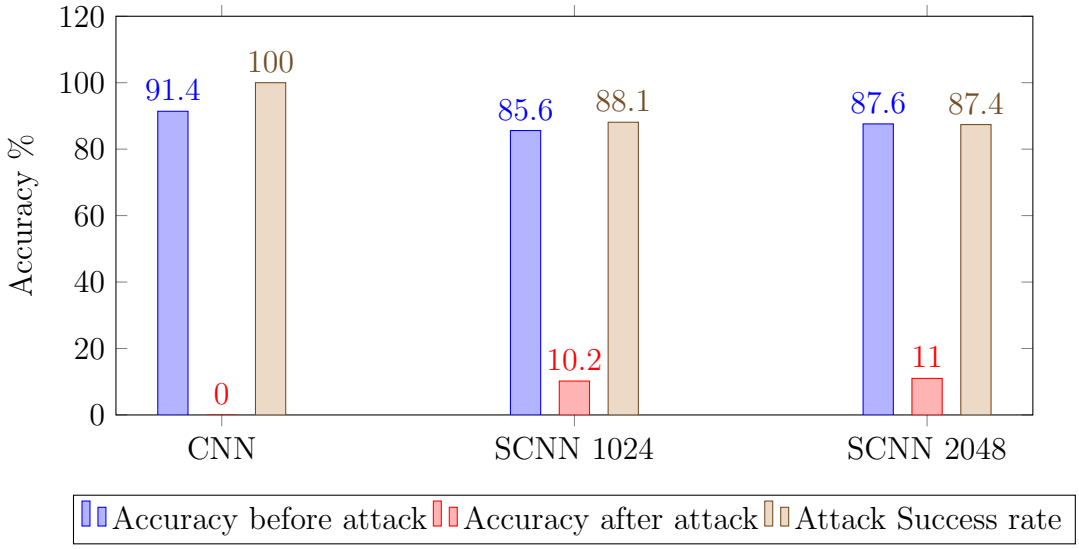


Figure 4.17: Fashion MNIST 2nd layer Convolution with stochastic multiplication only - Attack at 0.001

The second experiment of Fashion MNIST provides similar result to the first experiment since only multiplication in performed in the stochastic domain. The results tend be in the range of 9-10%. The resistance provided by the stochastic multiplication for the fashion MNIST dataset is marginal.

The third experiment in the Fashion MNIST dataset is performed by converting the entire first convolutional block into stochastic domain where the convolution including multiplication and addition are computed in stochastic domain. The activation function used is Stanh and NMax as the stochastic max pooling layer. Since addition is performed in stochastic domain, the accuracy tends to drop significantly which is why higher SN lengths are considered in this experiment. After the attack with 0.01 and 0.001 we see that the SCNN is able to withhold the measure of approximately 10% accuracy even after using stochastic addition. The experiment results are shown in the Fig 4.18 and Fig 4.19.

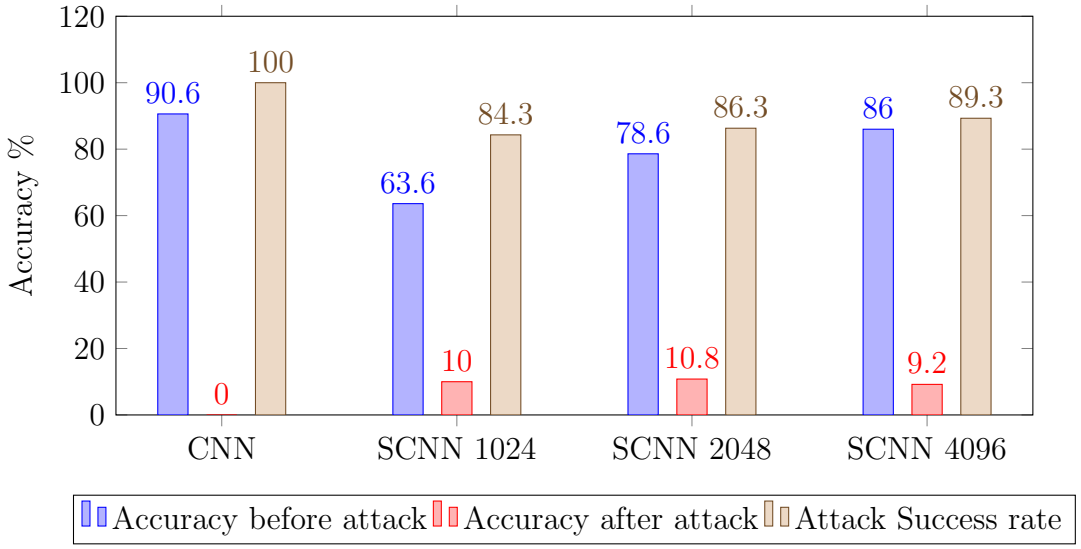


Figure 4.18: Fashion MNIST 1st Convolution layer Fully Stochastic - Attack at 0.01

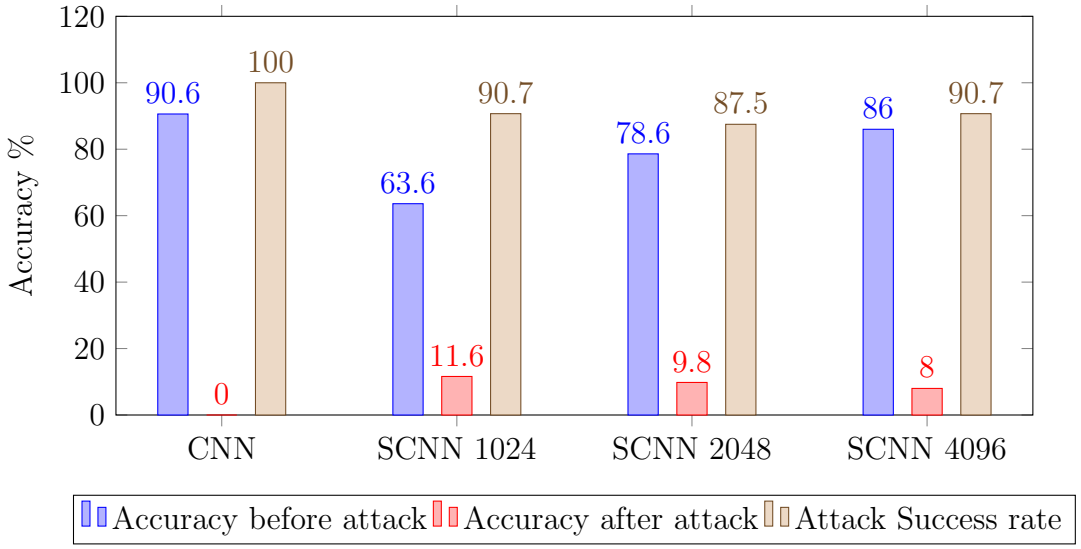


Figure 4.19: Fashion MNIST 1st Convolution layer Fully Stochastic - Attack at 0.001

The results of the third experiment where the complete first convolutional block is stochastic increases the accuracy of the network by a very small mar-

gin of 1% over that of the first experiment which consisted of only stochastic multiplication. The stochastic addition however decreases the accuracy for the non perturbed images significantly without providing much robustness in terms of perturbed images.

The fourth experiment in the Fashion MNIST dataset is similar to the third but the 2nd convolutional block is converted to stochastic domain to obtain the attack results for 0.01 and 0.001 overshoot parameters of deep fool attack. The 2nd convolutional block is able to provide slightly better accuracy compared to the 1st convolutional stochastic block with an average accuracy of 13-16%. But there is far significant reduction in accuracy for the original non adversarial dataset due to high number of stochastic addition which scales down the value. The results for the fourth experiment are shown in Fig 4.20 and Fig 4.21

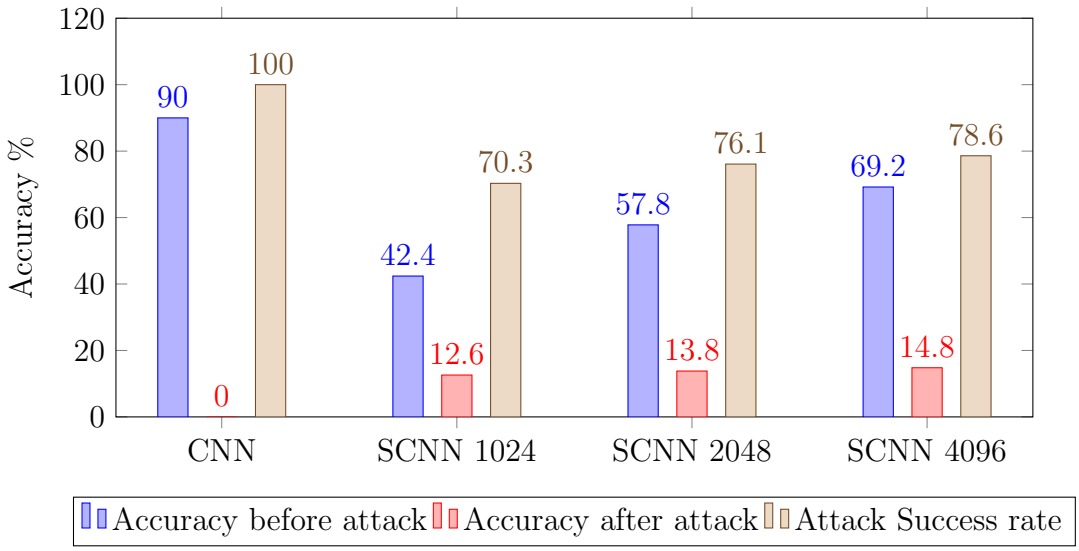


Figure 4.20: Fashion MNIST 2nd Convolution layer Fully Stochastic - Attack at 0.01

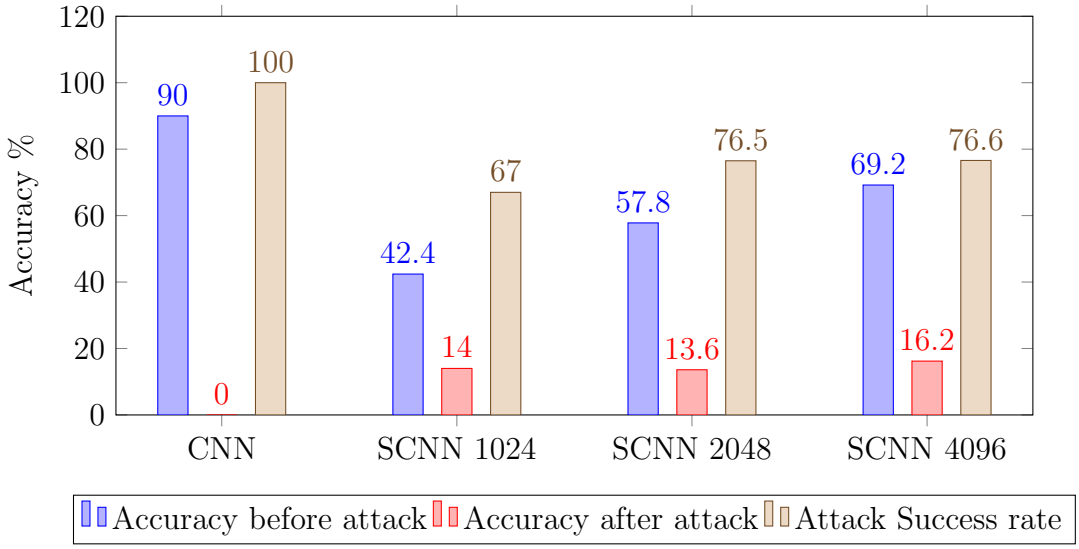


Figure 4.21: Fashion MNIST 2nd Convolution layer Fully Stochastic - Attack at 0.001

The final experiment of the Fashion MNIST dataset with entire second convolutional block in stochastic domain increases the accuracy especially for SN length of 4096. The only draw back being that lower SN length values provide poor accuracy in terms of non perturbed images which can be seen in the Fig 4.21 standing at 42.4% for SN length of 1024. The better accuracy can be attributed to the randomness provided by the addition and max pooling layers over the already existing 10% accuracy provided by stochastic multiplication layer.

4.4 Deep Fool attack on CIFAR-10

The first experiment in CIFAR-10 SCNN network for Deep Fool attack consists of 1 convolutional layer with stochastic multiplication only and the rest of the network is computed are non-stochastic. The perturbed image sample of the CIFAR-10 dataset after the deep fool attack are shown in Fig 4.22. Clearly, we can see that some images the perturbation are visible but others the perturbations are not visible to human eye but the CNN classifies the images incorrectly. The results of the these network before and after attack using Deep fool attack is shown from Fig 4.23 to Fig 4.26.

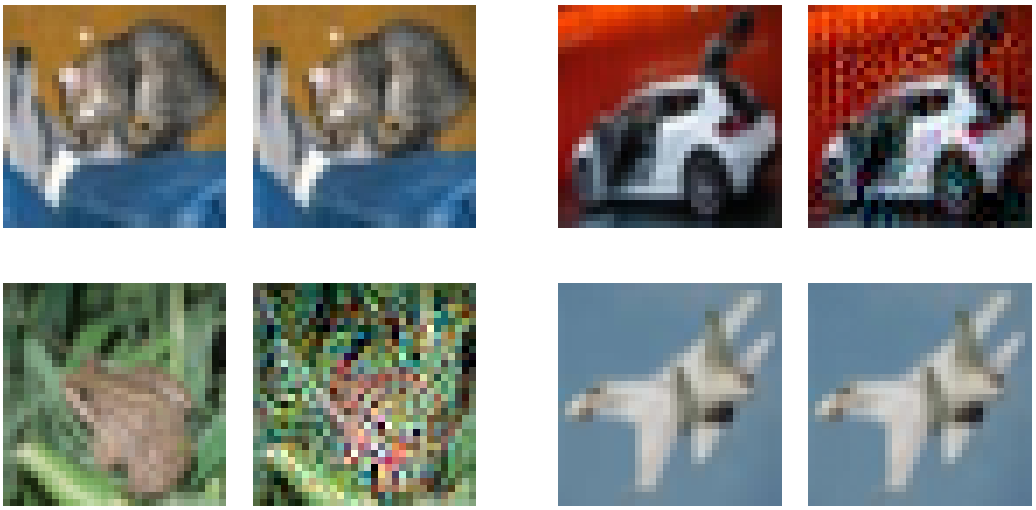


Figure 4.22: CIFAR-10 images perturbed by Deepfool Attack at 0.001 overshooting parameter

Here too the binary network is completely broken and provides no correct classification. The SCNN is able to nullify the perturbations and provide an accuracy of 25%. The SCNN is able to provide such accuracy only with multiplication performed in stochastic in the first convolutional layer. This shows that the randomness in the stochastic multiplication is able to nullify the small perturbations and provide better results over the standard CNN. Here almost all SCNNs are able to provide the same results irrespective the overshooting values from the deep fool attack.

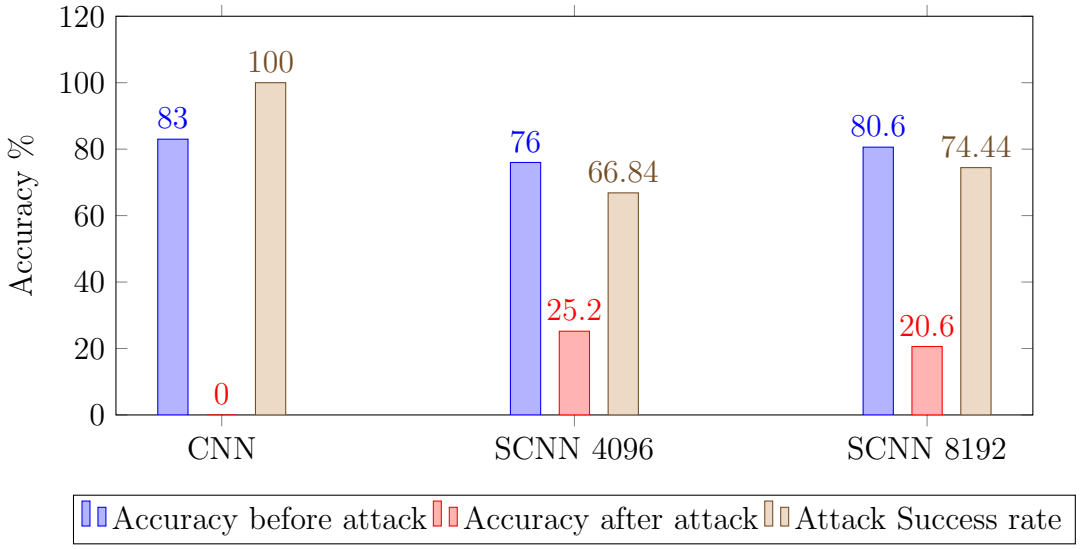


Figure 4.23: CIFAR-10 1st layer Convolution with stochastic multiplication only - Attack at 0.01

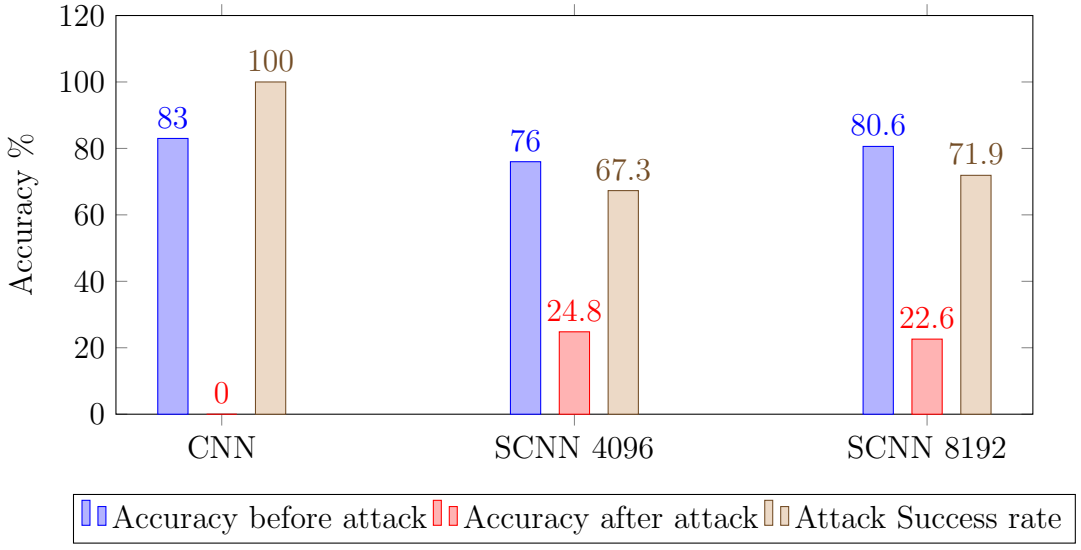


Figure 4.24: CIFAR-10 1st layer Convolution with stochastic multiplication only - Attack at 0.001

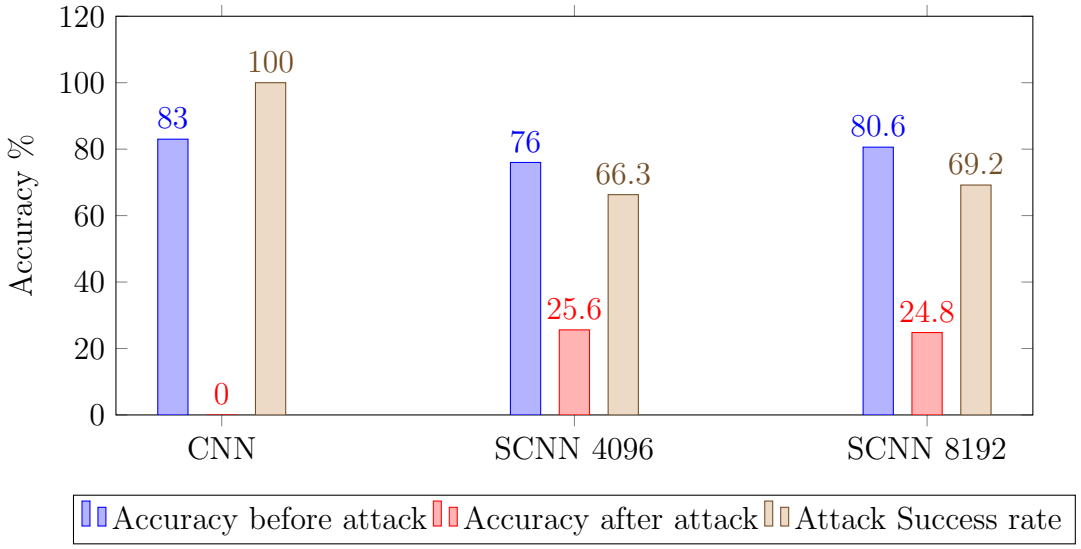


Figure 4.25: CIFAR-10 1st layer Convolution with stochastic multiplication only - Attack at 0.1

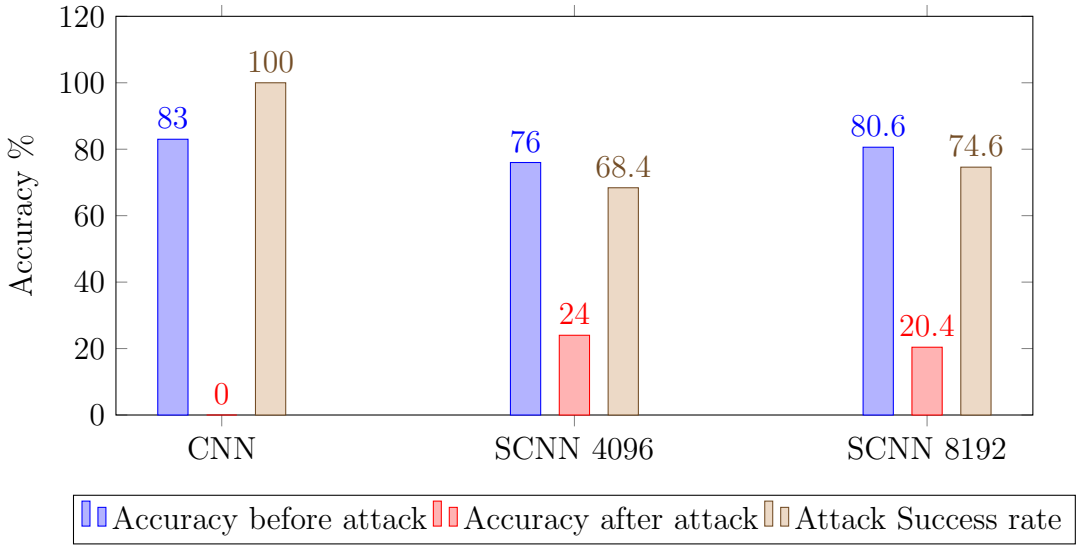


Figure 4.26: CIFAR-10 1st layer Convolution with stochastic multiplication only - Attack at 0.5

The stochastic multiplication from the first layer provides very good accuracy for the perturbed images. The perturbations are mainly spread among

the 3 channels of the input images and with the randomness provided by the stochastic numbers, the NN is able to nullify few of the minor perturbations. The accuracy CIFAR-10 NN is similar for any overshooting attack parameter with minor variations of 1%. Lower SN length tends to perform better against perturbations mainly due to lower resolution and the random quantization error acting in favour of the NN classification.

In the second experiment of the CIFAR-10 SCNN, only the second convolutional layers multiplication is performed in stochastic where as the rest of computations in the binary domain. The binary network completely fails and the SCNN is able to provide an accuracy of 21-23%. The accuracy after attack in current case is reduced compared to the accuracy after attack in the previous SCNN of 1st convolutional layer stochastic multiplication as the perturbations tend to increase at every layer since multiplication and additions are performed. These operations performed in the 1st layer tend to spread out the error and also increase the error in the network which thus results in lower accuracy in the SC in 2nd Convolutional layer. The results for 2nd Convolutional stochastic multiplication are shown in Fig 4.27 and Fig 4.28

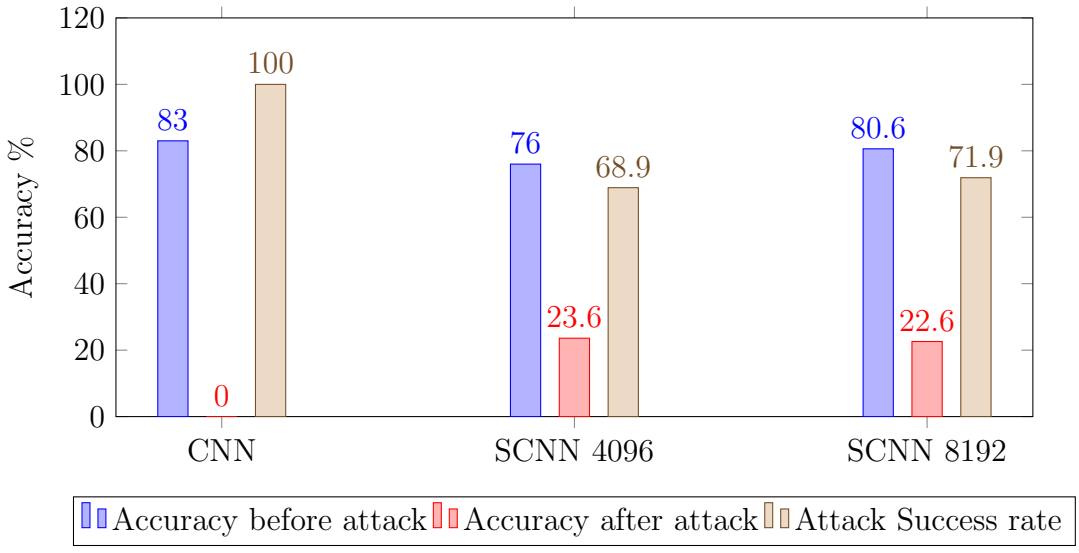


Figure 4.27: CIFAR-10 2nd layer Convolution with stochastic multiplication only - Attack at 0.01

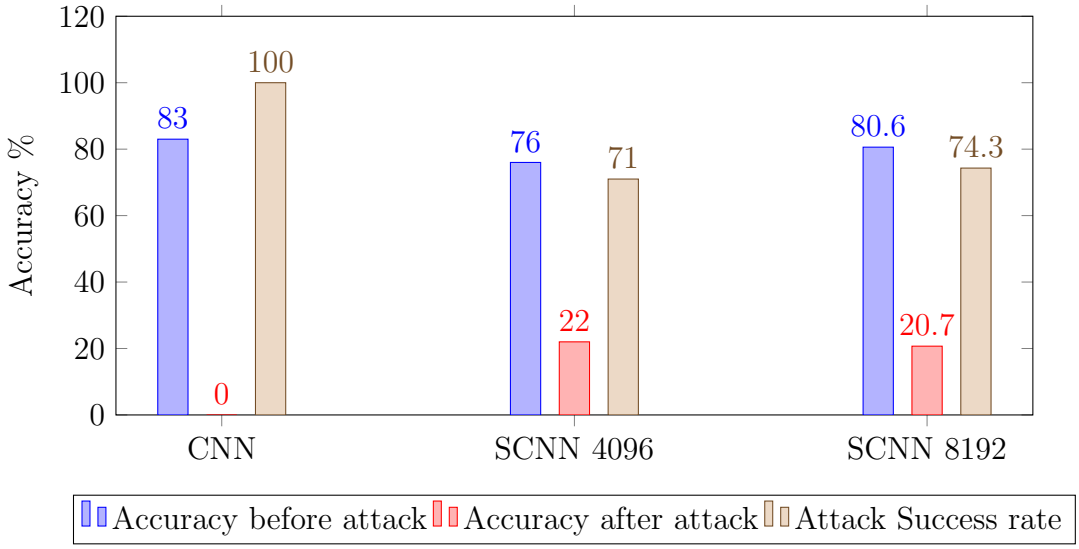


Figure 4.28: CIFAR-10 2nd layer Convolution with stochastic multiplication only - Attack at 0.001

The accuracy of second experiment of CIFAR-10 reduces slightly due to the higher number of layers of the input with 32 feature maps and spreading of the perturbations due to arithmetic operations from previous layer. The accuracy drops approximately 2% from the first experiment.

In the third experiment of CIFAR-10, the third convolutional layer multiplication is performed in stochastic domain which provides a similar accuracy of 17% after the adversarial attack. All other layers and operations are performed in the non stochastic domain. The robustness shown by the CIFAR-10 dataset is better compared to that of the MNIST and Fashion MNIST since the MNIST dataset have an average accuracy of 10-15% post adversarial attack. The results for the 3rd convolutional layer in stochastic domain is shown in Fig 4.29 and Fig 4.30 for overshoot parameters of 0.01 and 0.001 respectively.

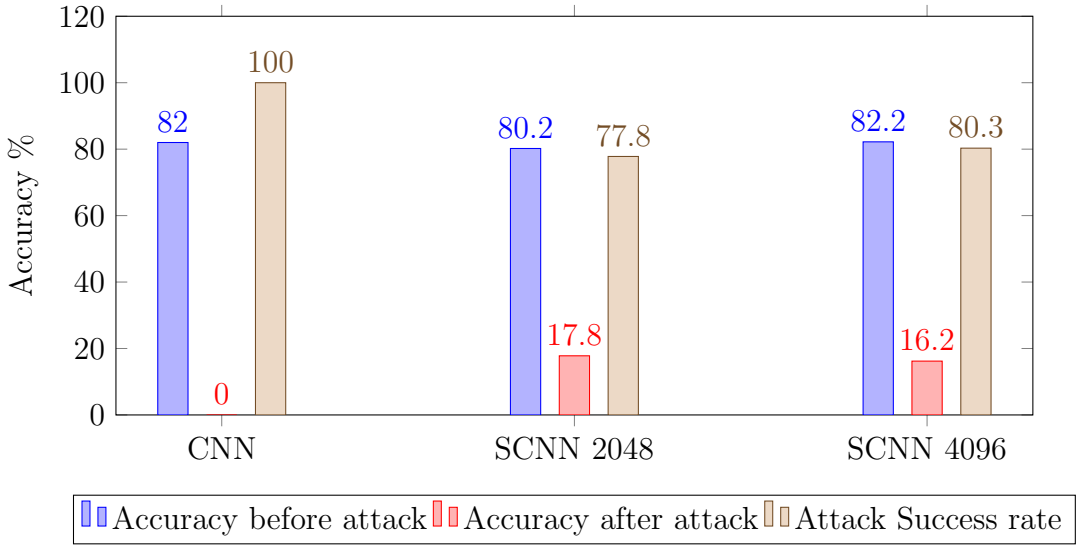


Figure 4.29: CIFAR-10 3rd layer Convolution with stochastic multiplication only - Attack at 0.01

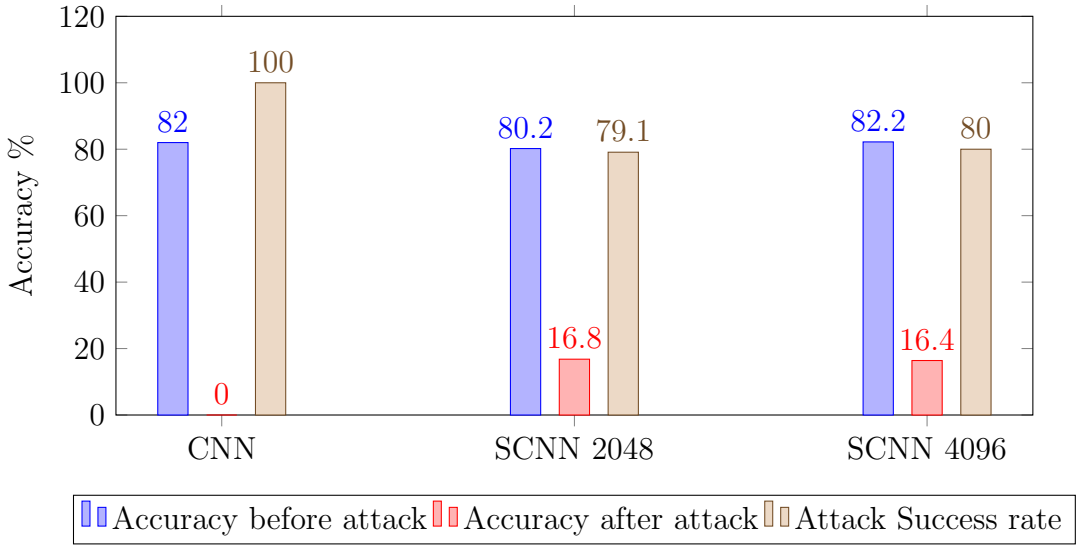


Figure 4.30: CIFAR-10 3rd layer Convolution with stochastic multiplication only - Attack at 0.001

Since the number of feature maps increase from 32 to 64, the accuracy further drops but by a higher margin of 4% compared to second experiment.

The reason for the drop in accuracy is attributed to the higher number of computations and error spread from previous layers and difficulty in nullifying errors at all feature maps.

The fourth experiment in CIFAR-10 dataset is performed by attacking the SCNN where the fourth convolution layer multiplication is calculated in the stochastic domain keeping rest of the NN in the usual binary CNN form. The attack once again is performed for 0.01 and 0.001 overshooting values of the deep fool attack. These two values are chosen since they tend to provide more robustness and they are the least overshooting values by which the output label predicted by the CNN can be changed. The 4th convolutional layer with multiplication in stochastic domain provides a post adversarial attack accuracy of 17% which is shown in Fig 4.31 and Fig 4.32.

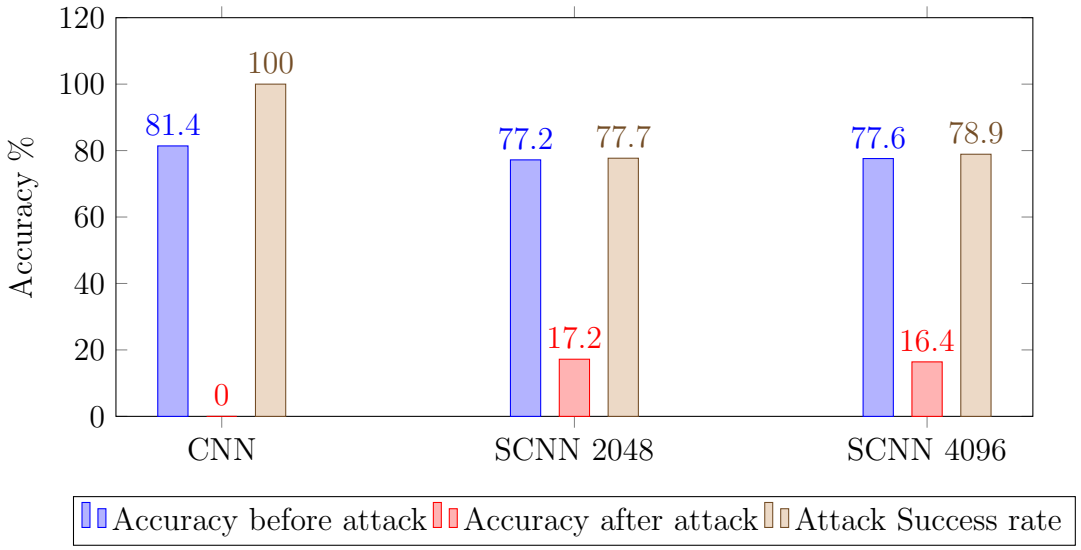


Figure 4.31: CIFAR-10 4th layer Convolution with stochastic multiplication only - Attack at 0.01

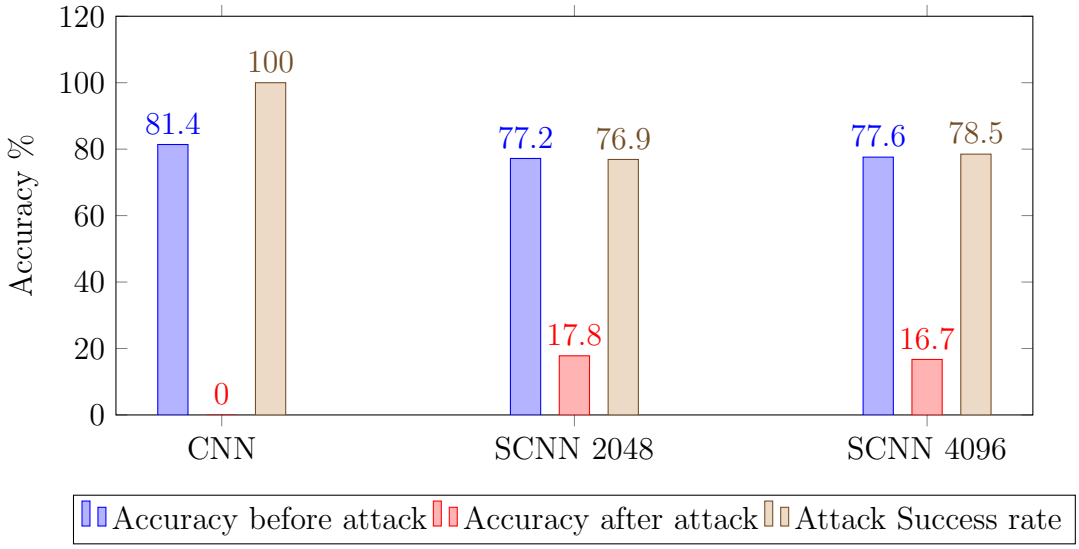


Figure 4.32: CIFAR-10 4th layer Convolution with stochastic multiplication only - Attack at 0.001

The fourth experiment performs similar to the results from the third experiment of CIFAR-10 due to same number feature maps of 64. The layers with similar feature maps provide similar results with minor change in the accuracy which can be seen in the first two experiments of CIFAR-10.

The fifth and sixth experiment is performed by replacing the multiplication in 5th and 6th convolutional layer to stochastic multiplication respectively. The experiments are exclusive to each other meaning when the 5th convolutional layer multiplication is in stochastic domain, the 6th convolutional layer multiplication is in the binary domain and vice versa. The fifth experiment provided an accuracy of 17-19% depending on the SN length which is shown in the Fig 4.33 and Fig 4.34. The sixth experiment provides an accuracy of 10-13% depending on the overshoot value and SN length as shown in Fig 4.35 and Fig 4.36. Since overshoot 0.001 value indicated the distance beyond the boundary(class label boundary) the image is pushed to, there is increase in accuracy after the attack when the image is more closer to the boundary. The accuracy of SCNN after attack with an overshoot value 0.001 provides better results than 0.01 which can be used to conclude that when the image perturbations are smaller, SCNN provides better resistance to adversaries.

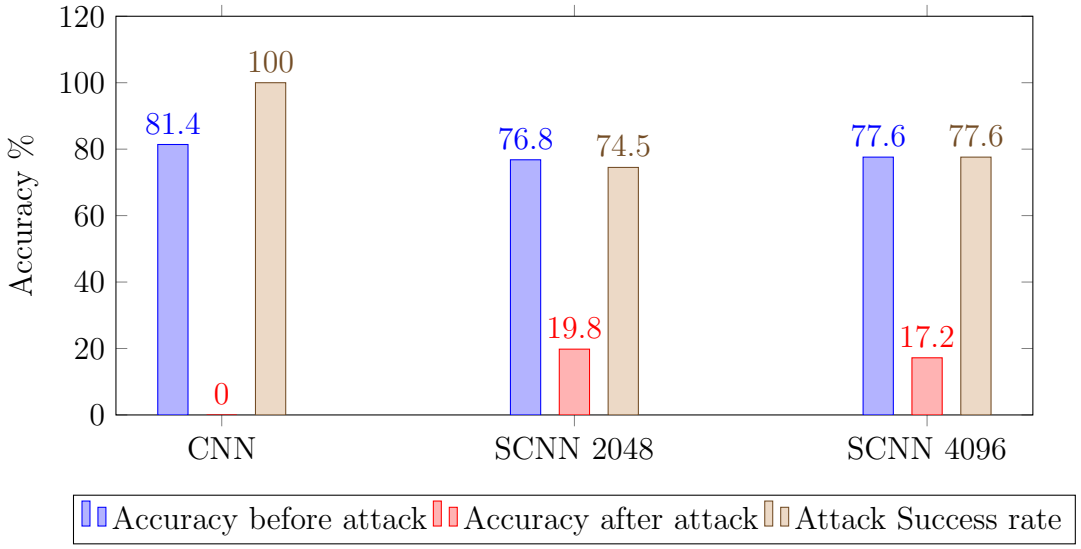


Figure 4.33: CIFAR-10 5th layer Convolution with stochastic multiplication only - Attack at 0.01

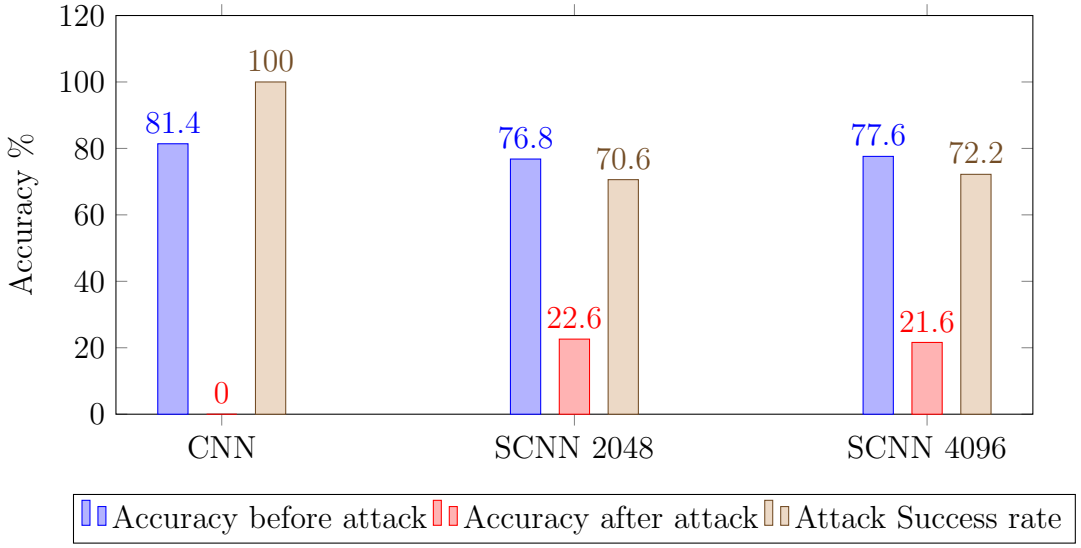


Figure 4.34: CIFAR-10 5th layer Convolution with stochastic multiplication only - Attack at 0.001

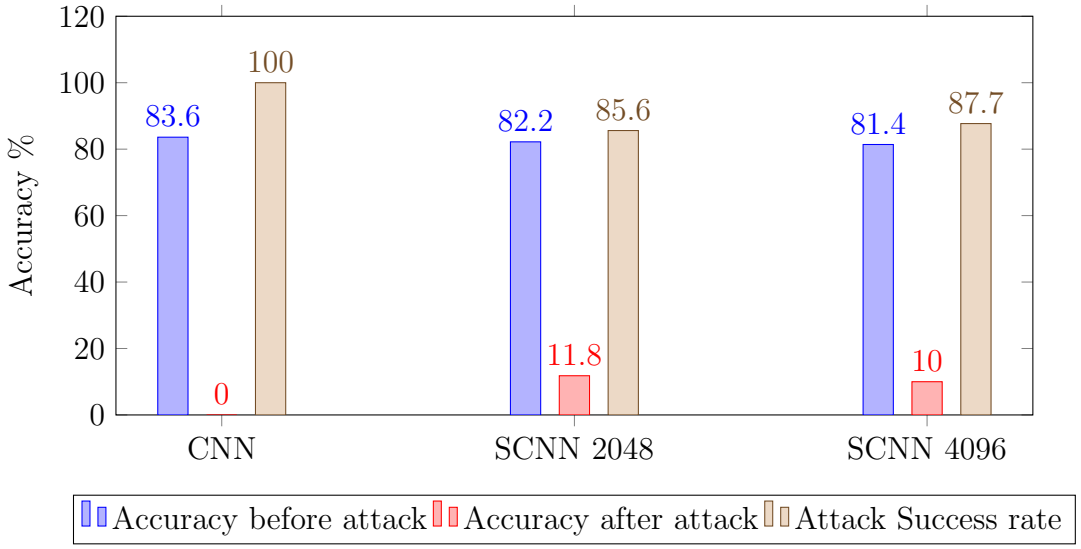


Figure 4.35: CIFAR-10 6th layer Convolution with stochastic multiplication only - Attack at 0.01

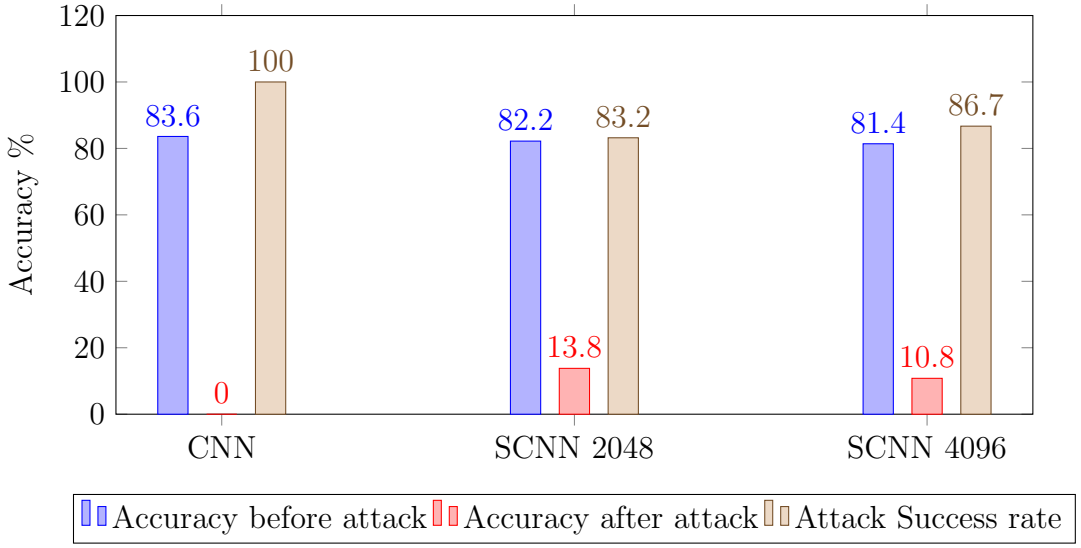


Figure 4.36: CIFAR-10 6th layer Convolution with stochastic multiplication only - Attack at 0.001

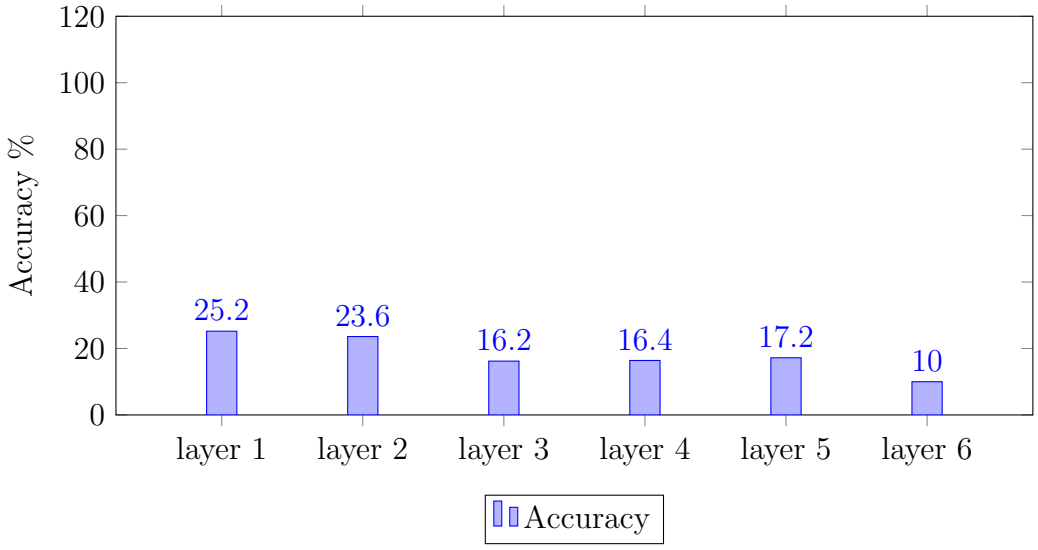


Figure 4.37: CIFAR-10 accuracy at each Convolution layer - Attack at 0.01

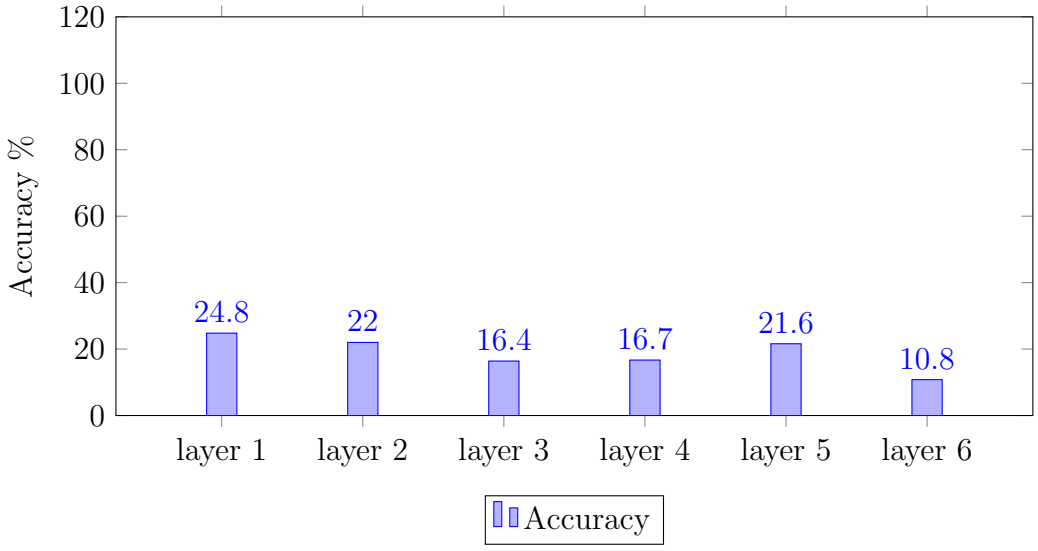


Figure 4.38: CIFAR-10 accuracy at each Convolution layer - Attack at 0.001

The fifth and sixth experiments moves by the same pattern of low accuracy and high number of feature maps. The fifth experiment how ever provides similar results to fourth experiment due to lower multiplications when moving from 64 feature maps to 128 where as the sixth experiment input contains

128 feature maps. The SCNN performs much better for overshooting parameter of 0.001 due to very small number of perturbations which SN can rectify. Fig 4.37 and Fig 4.38 provides the overall accuracy for each layer with attack parameter at 0.01 and 0.001 respectively, which shows a general trend of decrease in accuracy for increase in depth and number of feature maps.

The seventh experiment on the CIFAR-10 SCNN is performed by replacing Tanh function with ReLU function. The multiplication in the 1st convolutional layer is the only stochastic computation in the entire network. The binary network does not completely fail here compared to the previous networks. The binary CNN is able to provide upto 10% accuracy after the deep fool attack. The SCNN however provides 20-22% which shows that this type of network is less robust compared to the one discussed above. The reason for performing only multiplication in the CIFAR-10 SCNN in stochastic is due to huge scale down factors ranging upto 500 which causes significant loss in accuracy. To increase the accuracy we tend to increase the SN length, which increases computation time significantly thus losing the purpose of SC layers. The results of the seventh experiment are plot in the Fig 4.39 and Fig 4.40.

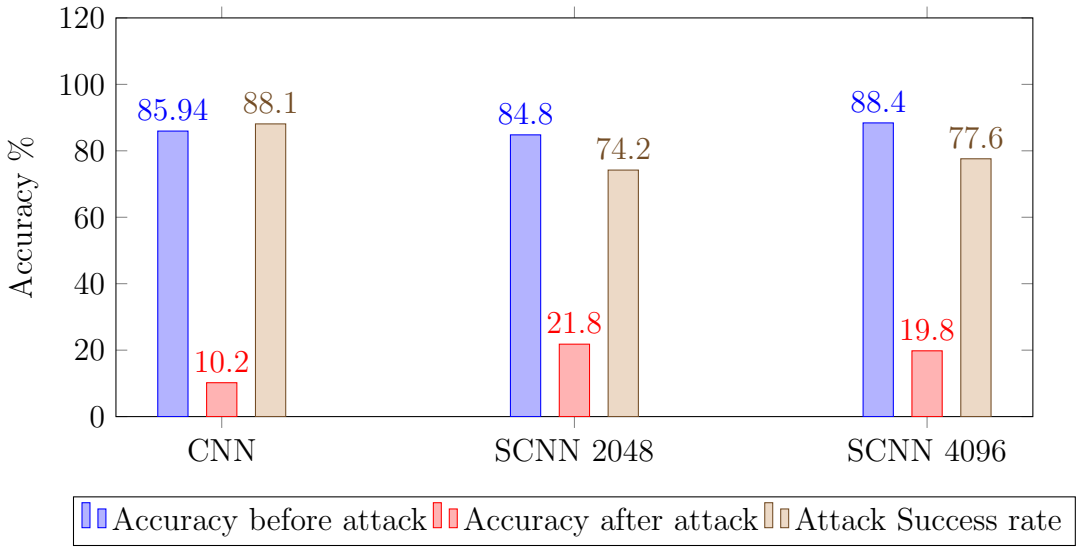


Figure 4.39: CIFAR-10 1st layer Convolution with ReLU with stochastic multiplication only - Attack at 0.01

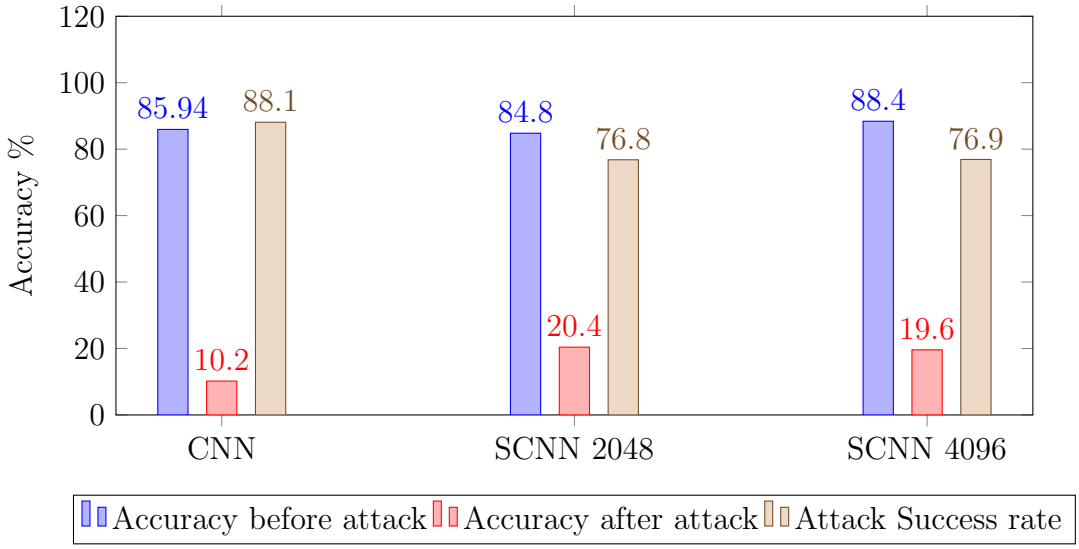


Figure 4.40: CIFAR-10 1st layer Convolution with ReLU with stochastic multiplication only - Attack at 0.001

The accuracy of the seventh experiment has similar to first experiment of CIFAR-10 with the only change of activation function from Tanh to ReLU. The binary network itself provided an accuracy of 10% where as the SCNN performed relatively better with 20%. With the Tanh activation, large perturbations are handled more effectively due the squashing of values to near 1 and -1 where as with ReLU large perturbations on the positive side are not clipped which spreads the error in subsequent layers.

Stochastic Max Pooling Layer Experiments

The following experiments are performed by converting only the max pooling layer into a stochastic max pooling layer using the mentioned NMax implementation and keeping all other layers in standard binary computation domain. The experiment helps us understand the amount of resistance the max pool layers provide against the Deep fool adversarial attack compared to the multiplication layers that is experimented above. The neural network was trained using Adadelta optimizer, Tanh activation, batch size of 64 and for 200 epochs provided a train accuracy of 86% and test accuracy of 81.6%. The following figures from Fig 4.41 to Fig 4.46 showcase the results of each of the max pool stochastic version. Each max pool layer is attack with two different overshooting Deep fool attack parameter of 0.01 and 0.001. As the depth increases the resistance provided to the attack by the NMax function

decreases i.e. at the first max pool layer the accuracy after attack is at 5.8% (Fig 4.41) where as at 3rd layer it drops to 1.8% (Fig 4.46) which is negligible.

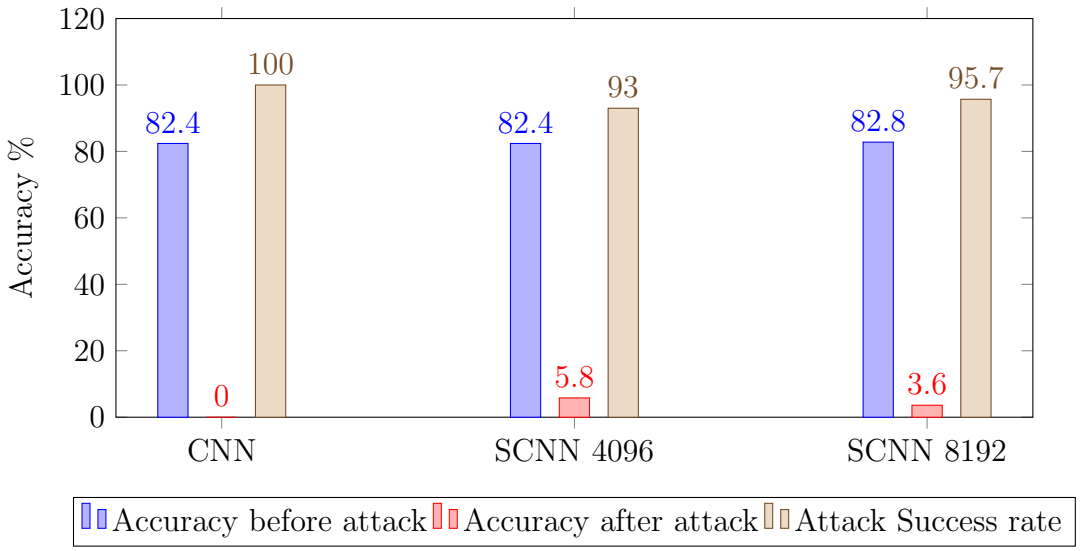


Figure 4.41: CIFAR-10 1st max pool layer as stochastic - Attack at 0.01

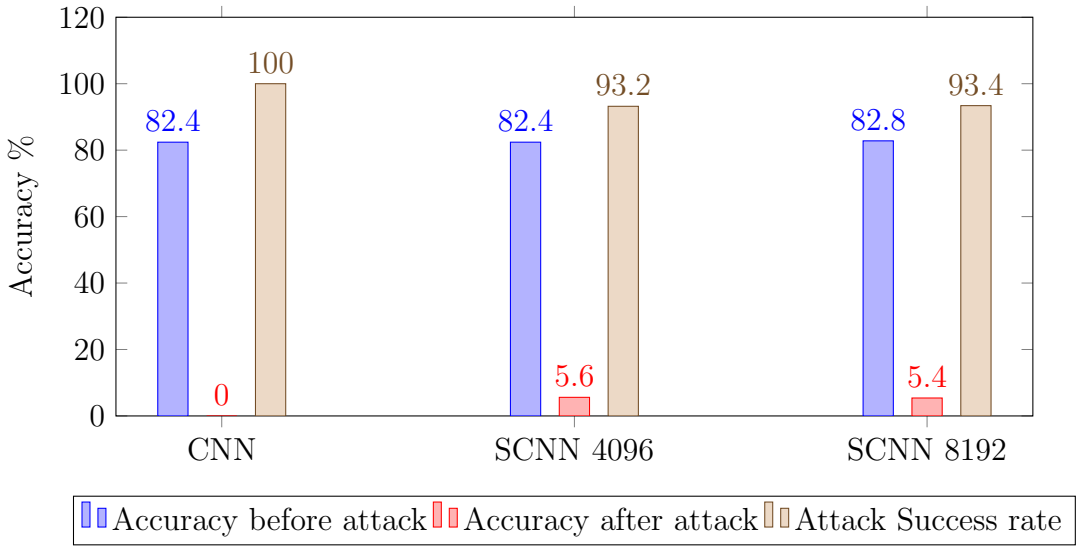


Figure 4.42: CIFAR-10 1st max pool layer as stochastic - Attack at 0.001

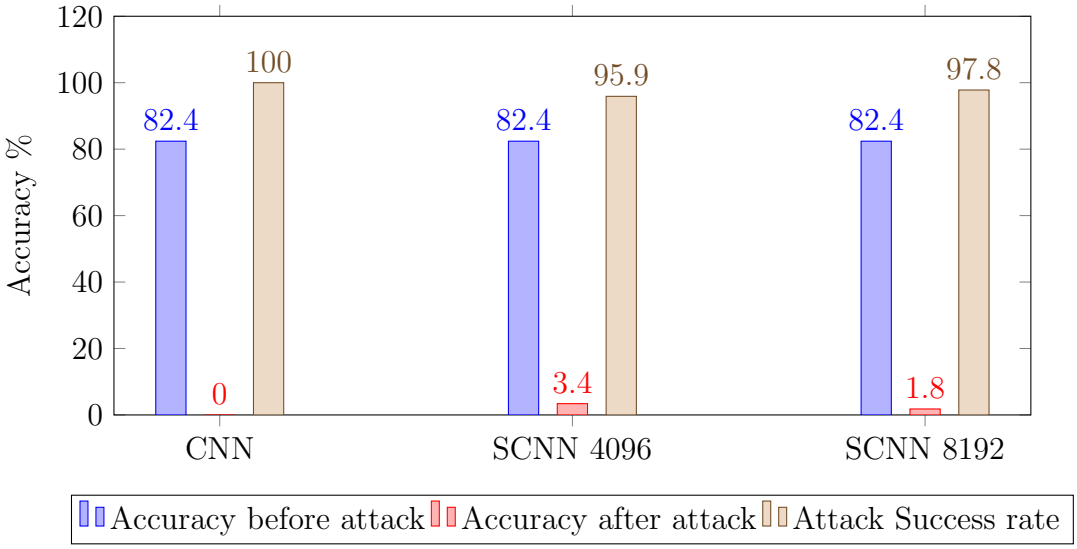


Figure 4.43: CIFAR-10 2nd max pool layer as stochastic - Attack at 0.01

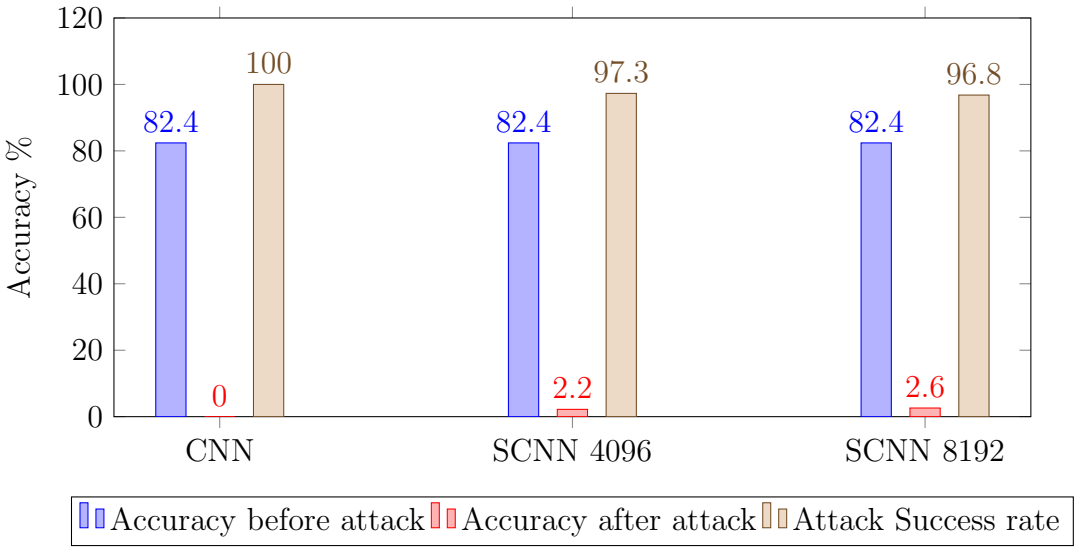


Figure 4.44: CIFAR-10 2nd max pool layer as stochastic - Attack at 0.001

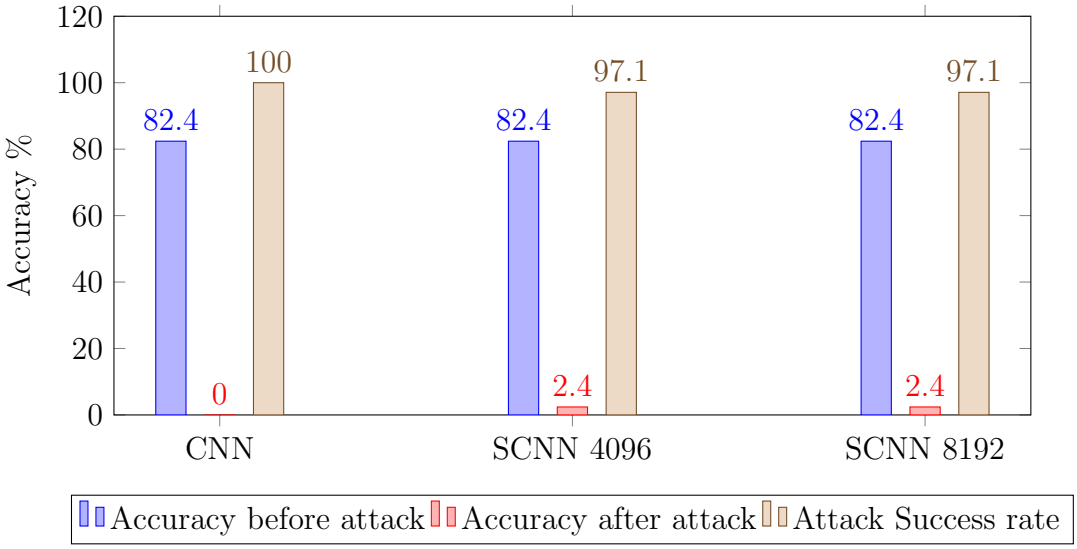


Figure 4.45: CIFAR-10 3rd max pool layer as stochastic - Attack at 0.01

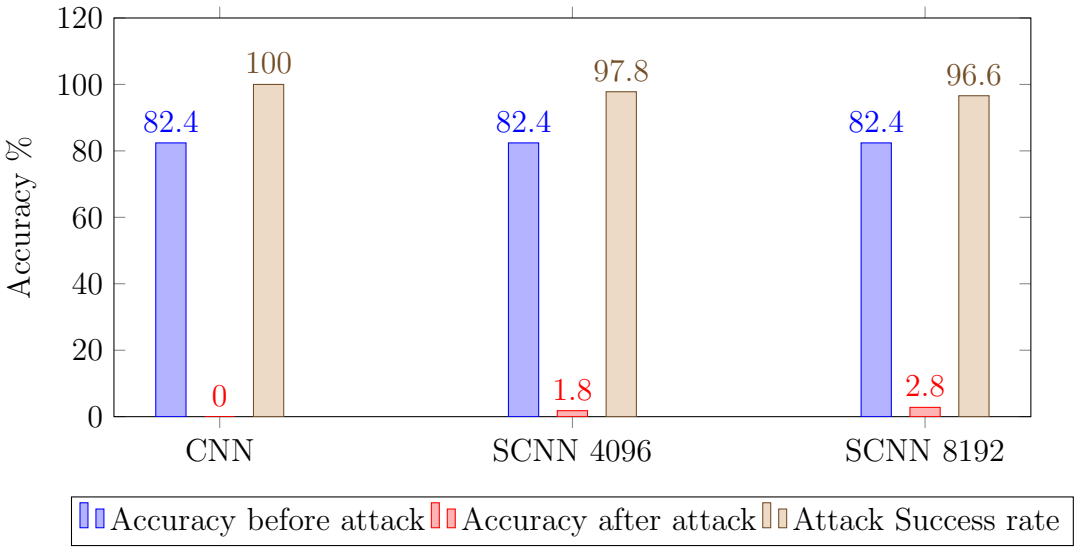


Figure 4.46: CIFAR-10 3rd max pool layer as stochastic - Attack at 0.001

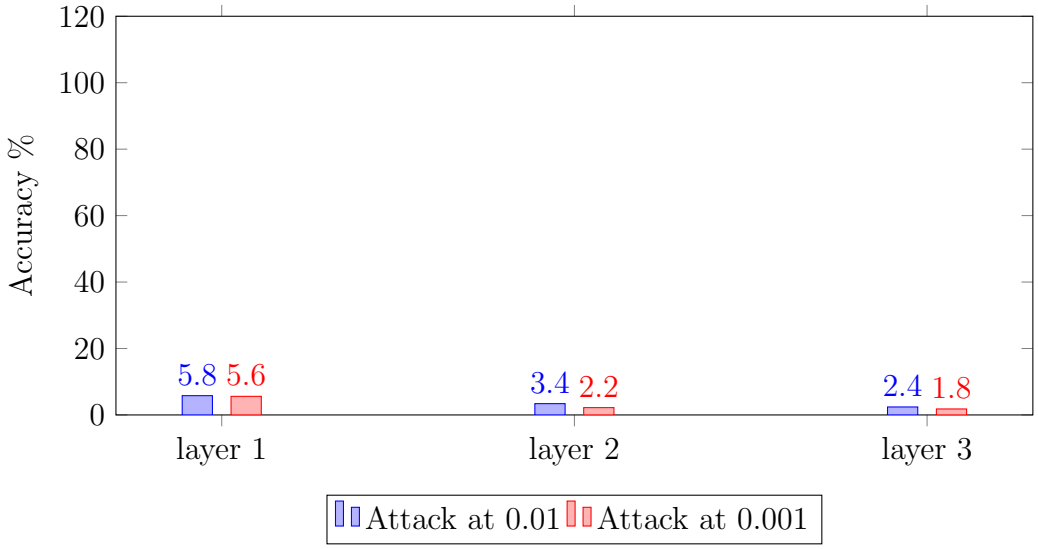


Figure 4.47: CIFAR-10 accuracy for each max pool layer with SN length 4096 - Attack at 0.01 and 0.001

The NMax function from the max pool layers in the stochastic domain provide lower resistance compared to the stochastic multiplication. The accuracy decreases at every layer and SCNN performs better for higher perturbations for max pooling layer since SCNN with attack parameter of 0.01 performs slightly better over 0.001 as seen in Fig 4.47.

Observations at the end of all experiments are as follows. As the depth increases, replacing deeper layers with stochastic layers does not provide much robustness due to the spread of errors in deeper layers. The initial layers tend to provide the best resistance to the deep fool adversarial attacks by nullifying the minor perturbation in the beginning. SCNN provides best results when the images are not severely perturbed. SCNN provides a small accuracy of 5-10% for highly perturbed images. The use of stochastic multiplication tends to have greater positive impact compared to other stochastic computational blocks.

Chapter 5

Extended Analysis

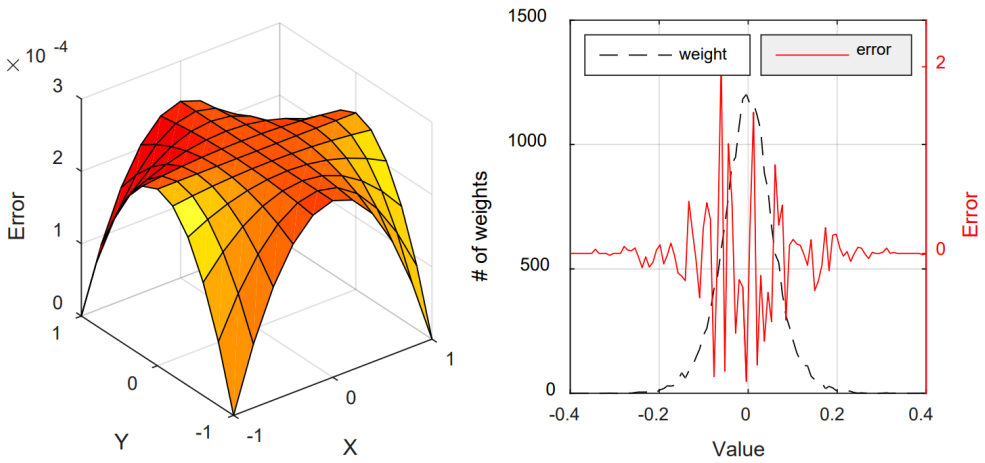
5.1 Optimizer Analysis for Stochastic Computation

Various optimizers such as RMSprop, Adadelta, AdaGrad and Adam were used to train the binary neural network. Although Adam performed the fastest convergence among other optimizers with least loss and higher test accuracy, the same was not reflected in the SCNN. Among the 4 mentioned optimizers the best performance in SCNN was provided by Adadelta with good accuracy in both binary as well as stochastic domain. The results of the each of the optimizers are mentioned in the Table 5.1 for MNIST dataset.

Optimizer	**CNN**	**SCNN (SN length = 2048)**
RMSprop	96.2%	57.6%
Adadelta	94.8%	87.4%
AdaGrad	95.4%	58.4%
Adam	97%	79.8%

Table 5.1: Optimizer Accuracy Table

One of the reason for such variation of accuracy is the distribution of weights of each optimizer. As per the authors in [20], if the weights are very close to zero, then stochastic multiplication operations causes random errors as shown in Fig 5.1. We see that as the weights tend towards zero the error increases. The error distribution is similar to a Gaussian distribution thus helping us calculating error at each weight without plotting the graph by using properties of Gaussian distribution.



(a) Error of XNOR gate (b) Error of near-zero weights

Figure 5.1: Random error problem of SC in DNN [20]. (a) Random error of XNOR gate in 1024-bit stream as absolute value. (b) 20000 weights distribution in 200 x 100 networks (left Y axis) and error multiplying by near zero values (right Y axis)

The histogram distribution of the weights of each of the optimizer provides a clearer picture. The weight distribution of RMSprop and Adam are provided in the Fig. 5.2. The distributions of RMSprop mainly is focus between -0.4 to 0.4 and majorly concentrated near zero. Same goes for the Adam where it is mainly distributed between from -0.2 to 0.2 which again is close to zero. Due their distributions being close to zero the errors tend to increase during stochastic multiplication which results in lower accuracy in stochastic domain.

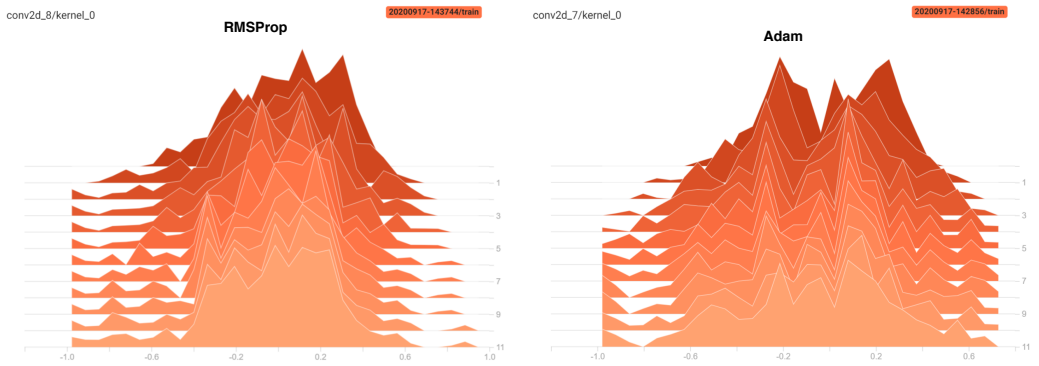


Figure 5.2: Histogram distribution of weights of optimizers (a) on the left RMSprop (b) and right Adam

Fig 5.3 provides the histogram distribution of weights of Adadelta and Adagrad. Adadelta can be seen with a fair distribution of weights from -1 to 1 and also if observed precisely the weights are not concentrated near zero. Adagrad weights seem more better distributed but the weights are concentrated near zero where as Adadelta weights tend to be more concentrated around 0.2. This provides Adadelta better performance in terms of multiplication with lower errors in the stochastic domain.

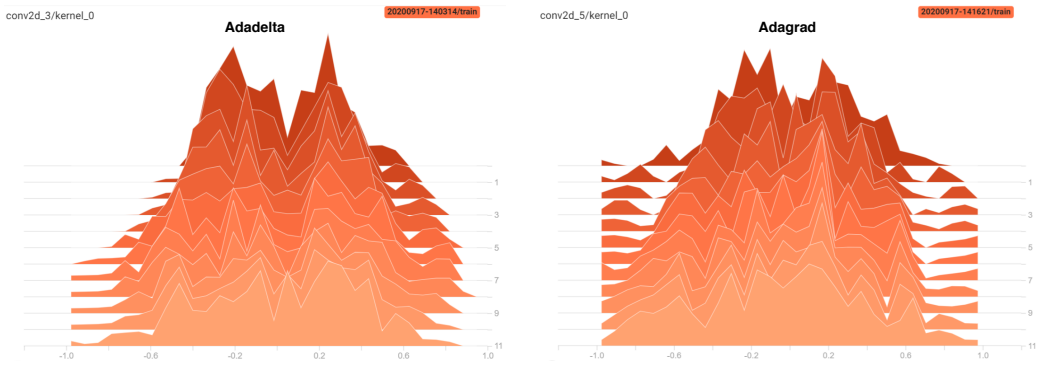


Figure 5.3: Histogram distribution of weights of optimizers (a) on left Adadelta (b) and right Adagrad

5.2 Structural Analysis of Perturbed Images for SCNN

The deep fool attack makes several adversarial changes to the input image in order to fool the network. It is important to identify the similarity between the original and perturbed images and take into account the when the SCNN provides correct classification even after the adversarial attack. The Structural Similarity Index (SSIM) is the metric used to compare the two images. SSIM quantifies image degradation during compression and data transmission. The formula for SSIM is provided in the equation 5.1

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_x^2 + c_2)} \tag{5.1}$$

where μ_x is mean of x, μ_y is mean of y, σ_x^2 is variance of x, σ_y^2 is variance of y, σ_{xy} is co-variance of x and y, $c_1 = (k_1 L)^2$ and $c_2 = (k_2 L)^2$ are variables to stabilises the denominator, L dynamic range of pixel values, $k_1 = 0.01$, $k_2 = 0.03$ by default.

A SSIM of value 1 means perfect similarity. The SSIM values of the original and perturbed input for correctly classified images had a SSIM value close to 1. Most of the correctly classified SSIM had a value ranging from 0.95 to 1.0. There were few SSIM which were very dissimilar having a value of less than 0.1 yet correctly classified but few in number. It is interesting to note that even with high similarity the binary CNN completely failed to correctly classify where as the SCNN provided a decent accuracy of 20-25% in CIFAR-10 network. Fig 5.4 show the perturbed images with different SSIM values.

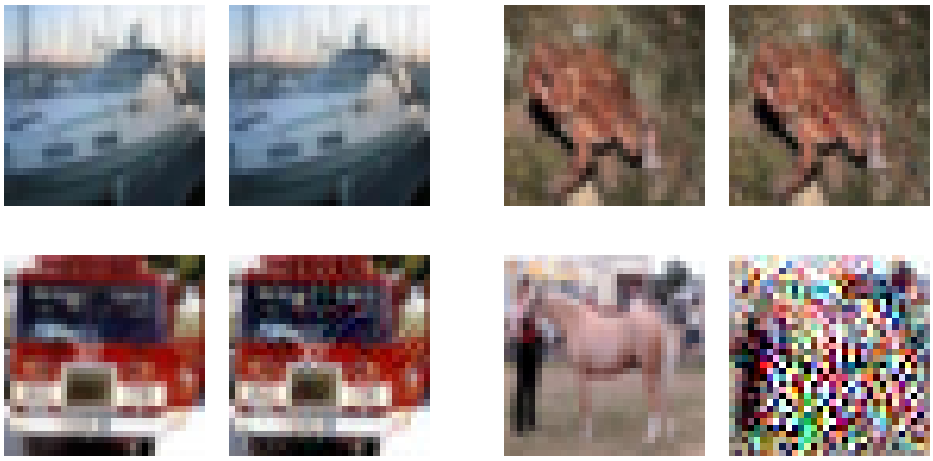


Figure 5.4: (a) Correctly classified images (top row) (b) Misclassified images (bottom row)

There are 4 sets of images in the Fig 5.4 and row contains two different sets of images. Each set contains two images with left being original and right being perturbed. The top row contains the correctly classified images with SSIM value of left set of images being 0.9999683 and the right set being 0.9604659. The misclassified set of images are in the second row of Fig 5.4 with SSIM value of left set being 0.93969005 and right set being 0.25640938. With SSIM value being close to 1, images become non differential to a human eye where as with SSIM value less than 0.95 the images have visible changes.

Chapter 6

Conclusion and Future work

This master thesis evaluates the robustness of SCNN against the deep fool attack with various permutation and combination of parameters of CNN, SC blocks and Deep fool attack. The SCNN was able to resist the perturbation inserted by Deep fool attack on the input image where as the conventional CNN was completely unfit for usage. The success rate of the attack on the conventional CNN is almost 100% in every case where as the success rate of the attack on SCNN varied between 60-80% depending on the dataset and SC parameters. The MNIST dataset with ReLU activation function provided upto 27% accuracy after the attack where as its binary counter part stayed at 0%. MNIST dataset with Tanh or Btanh activation function in the stochastic domain did not provide the required amount of resistance towards the attack and managed to give 13% and 8% accuracy respectively. Fashion MNIST dataset with Stanh activation function provided an accuracy of 11-16% after the Deep fool adversarial attack. CIFAR-10 dataset being far more complex provided accuracy upto 28% after the images were perturbed with Deep fool attack. Almost for every combination of SCNN for CIFAR-10 provided an accuracy above 20%. CNNs where the initial layers computations are performed with stochastic blocks provided better robustness compared to stochastic layers very deep in the network. SCNN can handle minor perturbation significantly well especially when the SSIM value ranges between 0.95 to 1 which are non-perceivable to human eye. Among the optimizers, the best results were provided by Adadelta optimizer. Optimizers like RMSprop, Adadelta, AdaGrad and Adam were tried and tested on the MNIST CNN and SCNN. This is due to the fact that Adadeltas weight distribution are fairly distributed rather than moving weights very close to zero. Thus SCNN can be used in parallel with CNN in order to provide better results in friendly and non-friendly environment.

The thesis used 3 CNN architectures, one each for the three datasets of MNIST, Fashion MNIST and CIFAR-10. This can be extended to other datasets of CIFAR-100 and ImageNet. The SCNN can be extended to implement architectures like ResNet-50, LeNet, AlexNet. Implementation of batch normalization in the test phase of the SCNN will significantly increase the accuracy of the network. This thesis dealt with only Deep fool attack and can be extended to other attacks such as FGSM, one pixel attack, Jacobian based saliency attack. Various activation functions can also be incorporated other than the standard Tanh and ReLU. Current setup works for multi-class classification and can be extended for multi-label classification. The current experiments were performed only 500-1000 test images which can be extended to the entire dataset. Thus from the results of the SCNN compared to the Conventional CNN, the SCNN provides more robustness and resistance against adversarial attacks.

List of Tables

List of Figures

Bibliography

[1] Wikipedia. Ai winter and ai spring.

[2] C. E. SHANNON. A mathematical theory of communication. *Bell System Technical Journal*, 1948.

[3] A. Alaghi and J. Hayes. Survey of stochastic computing. *ACM Trans*, vol. 12:pp. 92:1–92:19, 2013.

[4] B.R. Gaines. Stochastic computing systems. *Advances in Information Systems Science*, vol. 2:pp. 37 – 172, 1969.

[5] C. Cortes Y. LeCun and C. Burges. Mnist handwritten digit database. *ATT Labs*, vol. 2, 2010.

[6] K. Rasul H. Xiao and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, vol. abs/1708.07747, 2017.

[7] Alex Krizhevsky. Learning multiple layers of features from tiny images. Chapter 3, 2009.

[8] P. Ting and J. P. Hayes. Exploiting randomness in stochastic computing. *2019 IEEE/ACM International Conference on Computer-Aided Design*, pages pp. 1–6, 2019.

[9] Pascal Frossard Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi. Deepfool: a simple and accurate method to fool deep neural networks. *Ecole Polytechnique F ed erale de Lausanne*, 2016.

[10] F. ROSENBLATT. The perceptron: A probabilistic model for information storage and organization in the brain. *Cornell Aeronautical Laboratory*, 1958.

[11] R. J. Williams D. Rumelhart, Geoffrey E. Hinton. Learning representations by back-propagating errors. *Letters to Nature*, 1986.

[12] J von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata Studies, Princeton University Press,*, 1956.

[13] M. Rubinoff W. Poppelbaum and Eds. Elsevier M. C. Yovits. Statistical processors. *Advances in Computers*, vol. 14:pp. 187 – 230, 1976.

[14] Cohen D. A. Jeavons, P. and J. Shawe-Taylor. Generating binary sequences for stochastic computing. *IEEE Trans*, vol. 40:pp. 716–720, 1994.

[15] Y-C. Kim and M. A. Shanblatt. Architecture and statistical model of a pulse-mode digital multilayer neural network. *IEEE Trans Neural Netw*, vol. 6:pp. 1109–1118, 1995.

[16] Quero J. M. Toral, S. L. and L. G. Franquelo. Stochastic pulse coded arithmetic. *Proceedings of ISCAS*, vol. 1:pp. 599–602, 1999.

[17] V. C. Gaudet and A. C. Rapley. Iterative decoding using stochastic computation. *Electron. Lett*, vol. 39:pp. 299–301, 2003.

[18] Li X. Riedel M. D. Bazargan K. Qian, W. and D. J. Lilja. An architecture for fault-tolerant computation with stochastic logic. *IEEE Trans. Comput.*, vol. 60:pp. 93–105, 2011.

[19] B. D. Brown and H. C. Card. Stochastic neural computation. i. computational elements. *IEEE Transactions on Computers*, vol. 50:pp. 891–905, 2001.

[20] Joonsang .Y Jungwoo .S Jongeun .L Kiyoung .C Kyounghoon .K, Jungki .K. Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks. *53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016.

[21] F. Neugebauer, I. Polian, and J. P. Hayes. On the maximum function in stochastic computing. *Proceedings of the 16th ACM International Conference on Computing Frontiers, ser. CF '19. New York, NY, USA: Association for Computing Machinery*, page pp. 59–66, 2019.

[22] J. Shlens I. J. Goodfellow and C. Szegedy. "explaining and harnessing adversarial examples. *International Conference on Learning Representations*, 2005.

[23] P. Barham E. Brevdo Z. Chen C. Citro G. S. Corrado A. Davis J. Dean M. Devin S. Ghemawat I. Goodfellow A. Harp G. Irving M. Isard Y. Jia R. Jozefowicz L. Kaiser M. Kudlur J. Levenberg D. Mane R. Monga S. Moore D. Murray C. Olah M. Schuster J. Shlens B. Steiner I. Sutskever K. Talwar P. Tucker V. Vanhoucke V. Vasudevan F. Viegas O. Vinyals P. Warden M. Wattenberg M. Wicke Y. Yu M. Abadi, A. Agarwal and X. Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. 2016.

[24] Travis Oliphant. Guide to numpy. 2006.

[25] Jonas Rauber, Wieland Brendel, and Matthias Bethge. Foolbox: A python toolbox to benchmark the robustness of machine learning models. 2017.