MODEL ATTACK ON CONVOLUTIONAL NEURAL NETWORKS

A Thesis

by

ABHILASH RAJENDRA BABU VALLAMKONDA

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Anxiao Jiang
Committee Members,	Ruihong Huang
	Tie Liu
Head of Department,	Dilma Da Silva

December 2019

Major Subject: Computer Science

Copyright 2019 Abhilash Rajendra Babu Vallamkonda

ABSTRACT

Deep learning is a machine learning technique that enables computers to learn directly from images, text, or sound in the same way that people do. It is a key technology which enables self-driving cars and speech recognition. In the past few years, deep learning has been successfully used in a wide range of applications and has demonstrated results beyond what computers were thought to be capable of. This new technology is poised to change the way we live.

Despite the successes, the exact working of deep learning models is not well-understood, and they can fail in several unintuitive ways. One such vulnerability is that small modifications to the input, which might not even be noticeable for humans, are enough to fool these models. This vulnerability has received significant attention from the research community and is a well-studied problem. Our focus is the scenario where the parameters of the model, rather than its inputs, are maliciously modified.

Deep learning models contain a large number of parameters that interact with each other in complex ways, so small perturbations to a large number of parameters can produce a cumulative effect, causing the model to misbehave. Further, noise inherent in practical systems can act as a camouflage for such malicious perturbations, making it difficult to detect them.

Even though deep learning models have produced amazing results, their vulnerabilities present a serious concern that must be overcome before they can be deployed in practical systems. In this work, we evaluate the threat of attackers maliciously modifying the model parameters to compromise the model. We demonstrate that small perturbations to the parameters are enough to compromise the model without significantly affecting its performance. We also study the characteristics of these malicious perturbations and devise a strategy to detect such an attack.

DEDICATION

This thesis is dedicated to my

Father and Mother,

whose unwavering support is the single biggest reason for my success,

whose love and encouragement kept me going when things got hard,

who have given me more than I can ever hope to repay.

ACKNOWLEDGMENTS

I would like to express my sincere thanks to my advisor, Dr. Anxiao Jiang, whose expertise and foresight made this research possible. I have learnt a lot by simply listening to him and following his suggestions. I am grateful to have had this opportunity to work with him.

I also had the pleasure of working with Dr. Zhangyang Wang. His insightful suggestions enabled many of the breakthroughs in my work.

Special thanks to Alexis from the University Writing Center for being so supportive and also offering well-thought-out suggestions as I was working on my thesis draft and preparing for my thesis defense.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by Dr. Anxiao Jiang (advisor) and Dr. Zhangyang Wang of the CSE Department.

This work was supervised by a thesis committee consisting of Dr. Anxiao Jiang (advisor) and Dr. Ruihong Huang of the CSE Department and Dr. Tie Liu of the ECEN department.

All work for the thesis was completed independently by the student.

Funding Sources

Graduate study was not supported by any outside funding sources.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

In 2012, Alex Krizhevesky and his group used deep neural networks to win the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [2]. In the object classification competition, they nearly halved the top-5 error rate compared to other teams who were using traditional approaches. Their extraordinary performance gained a lot of attention, and since then, deep neural networks have demonstrated impressive results in a wide range of applications. They have surpassed humans in object recognition [3] and produced state of the art results in machine translation [4, 5, 6] and speech recognition [7].

Deep neural networks have been incorporated into Google Translate, resulting in fluent sentences that are easier to read and understand [8]. Most major companies including Amazon, Apple, Google, and Microsoft now use deep neural networks in their speech recognition systems [9, 10, 11, 12]. Deep learning models have also been used for medical applications like discovering new drugs [13] and diagnosing skin cancer [14] as well as autism [15]. They have been used to beat the World Champion Go player [16] and mastered a number of other games [17, 18]. They have also been used to create realistic images [19] and artworks [20] as well as compose music [21].

Despite the widespread use of deep learning models in a diverse set of fields, research suggests that they are vulnerable to attacks in several unintuitive ways. One such vulnerability is that small perturbations added to the input, which might not even be noticeable for humans, are enough to fool the model. Further, the adversarial inputs can be designed so that the models have a high confidence in their incorrect prediction [1, 22, 23].

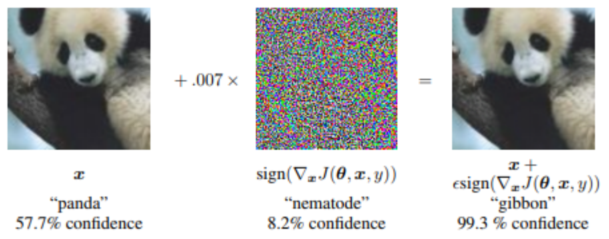

$\boldsymbol{x}$
"panda"
57.7% confidence
$+ .007 \times$
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$=$
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

Figure 1.1: A panda misclassified as a gibbon using FGSM [1] (Reprinted from [1])

We refer to this as the data attack since the model's input is being maliciously modified to fool the model. The data attack requires the addition of carefully crafted perturbations, which do not occur naturally. So, the threat posed by this type of an attack is limited to applications where the input is readily available to be modified and is not as severe for real-time applications like the self-driving car where the input is obtained directly from the real world. For such applications, modifying the parameters of the deployed model is a more effective attack strategy.

The way deep neural networks decide their classifications is not well-understood. Also, these models typically have a huge number of parameters that interact with each other in complex ways, so small perturbations to a large number of parameters can produce a cumulative effect, causing the model to misbehave. Noise inherent in practical systems can act as a camouflage for such malicious perturbations, making it difficult to detect such an attack. We refer to this as the model attack, and the goal of the model attack is to compromise the model by modifying its parameters.

Our goal is to study the threat posed by the model attack and find ways to mitigate the risk. The rest of the thesis is organized as follows: Chapter 2 summarizes the previous research on the model attack problem as well as the data attack problem. Chapter 3 formally defines the model attack problem. In Chapter 4, we describe our strategy to carry out the model attack and demonstrate that the model attack can be carried out without being easily detected. In Chapter 5, we examine the bit errors introduced due to the model attack and analyze the model-attack noise values using strip plots. Chapter 6 describes our strategy to detect the model attack. Chapter 7 describes our attempt to devise a strategy to detect the model attack by visualizing the noise values using t-SNE

[24]. In Chapter 8, we compare the model's predictions before and after the attack with the goal of understanding how the model attack affects the model's performance. We conclude the thesis and provide suggestions for future work in Chapter 9.

2. RELATED WORK

In this chapter, we discuss previous attempts to carry out the model attack and explain how our research relates to them. We view the model attack problem as the dual of the data attack problem, so we also review the existing research on the data attack.

2.1 Prior efforts to carry out the model attack

There has been some recent research focusing on the model attack problem. Several researchers have demonstrated that a model's classification ability can be compromised by perturbing its parameters.

- Liu et al. argue that fault injection attacks can be carried out on SRAMs to precisely modify any of the DNN weights stored in memory. They propose the Gradient Descent attack which causes the model to misclassify a given input pattern while minimizing the number of weights modified. They minimize the number of faults required to be injected by ignoring the modifications which are close to zero [25].

- Rakin et al. propose a strategy to identify the most vulnerable bits in the neural network. They demonstrate that flipping a small number of these bits can cause a large deterioration in performance [26].

- Qin et al. store weights as arrays of bits and each bit is flipped independently with a given probability. They examine how the probability of errors affects the model's performance. [27].

- Arechiga et al. introduce errors in the weight values using random bit flips and evaluate the performance of the network. They conclude that Multilayer Perceptrons are more robust than CNNs and that CNNs with larger kernels are more robust than those with smaller kernels [28].

All of the research mentioned above focuses on the security aspects of models implemented in conventional memories like the SRAMs.

2.2 Limitation of conventional systems and in-memory computing

Conventional systems use the von-Neumann architecture, so the data is stored separate from the processor. This necessitates the back and forth transfer of data between the processor and memory. This movement of data is time-consuming and energy-intensive, which limits the performance of these systems.

This limitation is unavoidable in conventional systems, which makes them unsuitable for applications using Internet of things (IoT) and machine learning, which are data-intensive, and yet, are required to be fast and energy-efficient. In order to make these applications practical, we need to avoid the overhead of transferring data from memory to the processor. For this purpose, novel memory systems, capable of performing matrix operations in-memory, are proposed.

Memristor is one such device, and it has been demonstrated to be more energy-efficient than CMOS devices. The resistance of a memristive device is determined by the amount of charge that has flowed through it, so its value can be varied by controlling the current flowing through the device. This property makes these devices well-suited to represent the trainable weights in a neural network. Further, the integrate and fire operation of the synapse is easily realizable using memristors making them a good choice to implement neural networks [29, 30]. Phase Change Memories (PCMs) also offer promise for neural network computations [31]. The different phases of the cells have different resistivities, and these provide stable resistance levels to store the weight values. These weights can be tuned in-situ during the learning process by adjusting the amount of heat applied to the cells. Research suggests that using these devices to implement neural networks will result in significant gains in speed and energy efficiency [32, 33]. In addition to being fast and energy-efficient, these devices are small and non-volatile, which makes them well-suited to store the large number of weights contained in a DNN. We envision that these devices will be instrumental while implementing DNNs in hardware.

Even though these devices have several useful properties, there is an inherent problem of noise associated with them [34]. This means that a system, which uses these devices, will perceive at least some noise even during normal operation. This noise can provide a camouflage for the attackers' perturbations to the model parameters, allowing them to compromise the model while the system only perceives an acceptable level of noise. Our work focuses on the possible security implications when DNN models are implemented using these devices.

2.3 Recently proposed strategy to detect the model attack

Recently, He et al. proposed a strategy to enable customers serving their models through cloud providers to verify the integrity of their deployed models. Since the actual parameters values are not easily available in this scenario, they propose a method to detect changes in the model parameters by means of specially crafted inputs. These inputs are designed so that any changes in the model's parameters will cause its prediction for the inputs to change [35]. Their strategy is effective in detecting changes to the model parameters; however, it is not suitable for detecting an attack in the scenario where the model parameters always contain some noise.

2.4 The data attack problem

A significant amount of research effort has gone into understanding the data attack problem and several strategies have been proposed to carry out the data attack. Goodfellow et al. hypothesize that the cause for adversarial examples is the linearity of the models. Based on this hypothesis, they propose the Fast-Gradient Sign Method which tries to maximize the change in the loss value by adding perturbations of magnitude ϵ to the pixels of the image [1]. This allows them to create a simple but effective strategy for generating adversarial examples.

$$x' \;=\; x + \epsilon sign(\nabla_x J(\theta, x, y))$$

where x is original image and x' is the adversarial image,

y is the model's label for x,

θ represents the model parameters and

$J(\theta, x, y)$ is the cost function used to train the model.

Szegedy et al. formulate the problem of finding adversarial inputs as follows [22]:

$$minimize \; \|x - x'\| + loss_{F,l}(x') \quad s.t \quad x' \,\epsilon\, [0, 1]^n \tag{2.1}$$

where x is the original image,

l is the desired incorrect label,

F is the trained neural network,

$loss_{F,l}(x')$ is the negative log probability (cross entropy) that the model assigns a label l to x'.

By minimizing $\|x - x'\| + loss_{F,l}(x')$, they find x', the closest image to x which is most likely to be misclassified as l by the network. Carlini et al. build on this approach by reformulating the box constraint in Eqn. 2.1 using the hyperbolic tangent function. This allows them to devise a more robust attack strategy using the Adam optimizer [23].

3. THE MODEL ATTACK PROBLEM

The model attack aims to compromise the model's classification ability without alerting the system. In this chapter, we identify an objective for the model attack and formally define the problem.

3.1 Objective of the model attack

Before we can define the problem, we need to identify the objective of the model attack. The data attack involves perturbing the input image so that the model is unable to classify it correctly. In the case of the model attack, instead of the input image, the parameters of the model are perturbed. So, the model attack problem can be viewed as the dual of the data attack problem. We use this observation to define the goal of the model attack as causing the model to misclassify a given input.

3.2 Problem Definition

We have defined the objective of the model attack as causing the model to classify a specific input with a specific incorrect label. However, simply meeting this objective is not enough for a successful attack. If the attack results in a significant change in the performance of the model or if the added perturbations are too large, then the system is likely to become aware of the attack and decide to reload the original weights, causing the model attack to fail. So, for a successful attack, we require that the added perturbations and the resulting change in model performance are small enough so that the system is unable to detect the attack.

We evaluate the change in the model's performance by comparing its accuracy on the test set before and after the attack. For the magnitude of the perturbations, we evaluate the SNR (Signal to Noise Ratio) of the weights considering their original values as the signal and the added perturbations as the noise.

Let $w_1, w_2, ..., w_n$ be the n weights in the original DNN and
$w'_1, w'_2, ..., w'_n$ be the corresponding weights in the compromised DNN.
Then w_i is the signal value, and $w'_i - w_i$ is the corresponding noise value.

The noise power and signal power are given by

$$NoisePower = \frac{1}{n}\sum_{i=1}^{n}(w_i - w_i')^2$$

$$SignalPower = \frac{1}{n}\sum_{i=1}^{n} w_i^2$$

$$SNR = \frac{SignalPower}{NoisePower}$$

Since $NoisePower$ is also equal to the MSE (Mean Square Error) of the weights, we can also define SNR as follows:

$$SNR = \frac{SignalPower}{MSE}$$

For a given model, the power of the signal is fixed, so the SNR only depends on the magnitude of the added perturbations. The means that minimizing the MSE value in the weights leads to a high SNR value.

Considering the objective of the model attack and the fact that the attack must minimize the noise added to the model parameters, we can formulate the model attack problem as follows:

$$minimize \; \|W - W'\| \quad s.t \quad C(W', x) = l \tag{3.1}$$

where x is the target image which is to be misclassified,
W is the set of initial parameter values,
W' is the set of modified parameters,
C represents the model's predictions for a given set of parameter values and input, and
l is the desired incorrect label.

In the above formulation, $C(W, x) = l$ is a highly discrete mapping, which makes the problem very hard to solve. So, we approximate the problem by substituting this mapping with the loss function and solve the following problem instead:

$$minimize \; \|W - W'\| + \gamma * cross\_entropy_{F,l,x}(W') \tag{3.2}$$

where F represents all aspects of the neural network model except its parameter values,

$\gamma > 0$ is a constant and

$cross\_entropy_{F,l,x}$ is the negative log probability that the model represented by F assigns a label l to x.

Equation 3.2 is our formulation for the model attack problem, and it explicitly satisfies our goal of causing the model to misclassify a given image with minimal perturbations. We observe that solving this problem also minimizes the change in the model's performance, and this has been verified empirically in the subsequent chapters.

4. EFFECTIVENESS OF THE MODEL ATTACK

In this chapter, we demonstrate that the model attack can be carried out in a way that is not easily detectable. We describe a strategy to carry out the model attack and show that the model can be compromised with negligible changes in the model's performance and weight values.

4.1 Our Approach

We choose one image from the training set as the target image which is to be misclassified. Then, we create two datasets from the training set; the first contains all of the examples in the training set except the target image while the second contains only one example, which is the target image with its label changed to the desired incorrect value. We use the first dataset, which contains only correctly labeled examples, to train the initial model. Then, we carry out the model attack on this model by retraining it on the second dataset. Below we describe our initial attempt to carry out the model attack and then explain how overcoming its limitations led us to our "Loss Function based model attack" strategy.

4.1.1 Initial Attempt

Our initial attempt involved retraining the initial model on the second dataset to learn the incorrect classification. This strategy was successful in causing the model to misclassify the target image with only a small drop in its test set accuracy; however, it caused a large drop in the SNR value of the model parameters. Hence, our initial attempt was not successful.

4.1.2 Loss Function Based Model Attack

The failure of our initial attempt can be attributed to the fact that it had no incentive to minimize the changes in the weight values during the retraining process. The loss value used during the retraining process only included a single cross entropy term to learn the desired incorrect label for the target image. Due to this, the retraining resulted in large changes in the parameter values, causing the SNR value to deteriorate significantly. In order to overcome this, we modified the loss value used during the retraining phase to include the MSE values of the weights along with the cross entropy term. The MSE values represent the magnitude of the changes in the weight values, so minimizing the loss value also minimizes the perturbations. Using this approach, which we refer to as the Loss-function based model attack, we were able to successfully carry out a model attack with minimal drop in performance and SNR values.

The strategy involves retraining the initial model on the second dataset using the modified loss function which is seen below:

$$loss = k_1 * cross\_entropy + k_2 * \sum_{n=1}^{N}(MSEweights_{l_n} + MSEbiases_{l_n}) \tag{4.1}$$

where k_1 and k_2 are constants,
$cross\_entropy$ is the negative log probability that the model assigns the desired label to the target image,
N is the number of layers in the CNN,
$MSEweights_{l_n}$ and $MSEbiases_{l_n}$ are the MSE values for the weights and biases in layer n.

4.2 Performance Evaluation

We evaluate the performance of our strategy on CNNs trained on the MNIST and CIFAR-10 datasets. The steps undertaken are as follows:

1. Choose a target image from the training set and create the two datasets as described earlier.
2. Use the correctly labeled dataset to train the model.
3. Measure the model's test set accuracy and confidences for the target image. These are the values for the model before the attack.
4. Record the model weights; these are required to compute the SNR value after the attack.
5. Retrain the model for 100 epochs with the second dataset using the modified loss function as described earlier.
6. Measure the model's test set accuracy and confidences for the target image again. These are the values for the model after the attack.

7. Record the model weights and compute the SNR value by comparing them with the original weights recorded in step 4.

Tables 4.1-4.10 provide the model's test set accuracy and confidences for the target image before and after the attack, along with the SNR values for each layer in the model after the attack. From the model's confidence values, it is seen that the model's prediction for the target image has changed. It is also seen that the model's accuracy on the test set is not significantly affected as a result of the attack. Further, from the high SNR values, we can infer that the magnitude of the added noise is negligible compared to the actual weight values.

4.2.1 MNIST dataset

We used a 5 layer CNN for MNIST. It contained 3 convolutional layers followed by a dense layer and a softmax layer. The following loss function was optimized during the retraining phase:

$$\begin{split} loss = 4 * cross\_entropy + 1e7 \\ * (mseWconv1 + mseWconv2 + mseWconv3 + mseWdense + mseWout \\ + mseBiasconv1 + mseBiasconv2 + mseBiasconv3 + mseBiasdense + mseBiasout) \end{split} \tag{4.2}$$

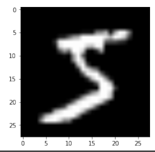
Tables 4.1-4.5 contain the results corresponding to attacks on models trained on the MNIST dataset.

4.2.2 CIFAR-10 dataset

We used a 7 layer CNN for CIFAR-10. It contained 5 convolutional layers followed by a dense layer and a softmax layer. The following loss function was optimized during retraining:

$$\begin{split} loss = 10 * cross\_entropy + 1e7 \\ * (mseWconv1 + mseWconv2 + mseWconv3 + mseWdense + mseWout \\ + mseBiasconv1 + mseBiasconv2 + mseBiasconv3 + mseBiasdense + mseBiasout) \end{split} \tag{4.3}$$

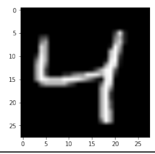
Tables 4.6-4.10 contain the results of attacks on models trained on the CIFAR-10 dataset.

	Before attack	After Attack
Model's confidences for image below 	0: 4.442189e-13 1: 1.1273594e-09 2: 5.146326e-11 **3: 0.0010443808** 4: 1.2421229e-11 **5: 0.9989555** 6: 2.4957295e-12 7: 1.6122265e-10 8: 2.326347e-08 9: 4.0799158e-08	0: 2.2420809e-12 1: 1.3952707e-09 2: 2.8070093e-10 **3: 0.7530272** 4: 3.5899783e-11 **5: 0.24697252** 6: 6.874817e-12 7: 1.7742201e-09 8: 1.517212e-07 9: 1.5818361e-07
Test Set Accuracy	0.9896	0.9891

SNR values for all five layers after the attack

snrWeights	6.9e07	5.8e05	4.4e04	1.3e04	5.1e06
snrBiases	5.2e07	1.0e08	5.7e07	3.8e07	2.1e08

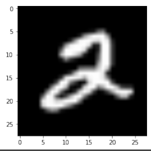
Table 4.1: The table shows the change in test set accuracy, model confidences, and SNR as a result of causing the model to misclassify an image of a "5"(seen above) as a "3"

	Before attack	After Attack
Model's confidences for image below 	0: 1.9398676e-10 1: 4.3550837e-05 2: 6.234967e-07 3: 2.1184778e-09 **4: 0.9998746** 5: 6.897561e-08 6: 3.9395642e-10 **7: 7.928939e-05** 8: 1.0539956e-07 9: 1.6953845e-06	0: 4.6098934e-09 1: 0.002334779 2: 0.00013663109 3: 4.8364655e-07 **4: 0.36741754** 5: 3.639659e-07 6: 7.258867e-10 **7: 0.630051** 8: 2.988948e-06 9: 5.614638e-05
Test Set Accuracy	0.9903	0.9880

SNR values for all five layers after the attack

snrWeights	7.6e07	5.4e05	2.3e04	6.9e03	3.8e06
snrBiases	1.1e07	1.2e08	5.6e07	4.2e07	3.2e07

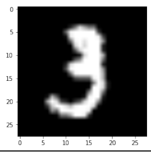
Table 4.2: The table shows the change in test set accuracy, model confidences, and SNR as a result of causing the model to misclassify an image of a "4"(seen above) as a "7"

	Before attack	After Attack
Model's confidences for image below 	0: 3.565935e-11 1: 1.2491886e-09 **2: 0.99999523** **3: 4.2405964e-06** 4: 3.3369632e-09 5: 8.024273e-11 6: 3.966121e-13 7: 1.7707281e-08 8: 4.6248493e-07 9: 1.5431284e-12	0: 5.1603e-10 1: 1.0379596e-08 **2: 0.4470869** **3: 0.55283785** 4: 2.1902817e-07 5: 6.170977e-08 6: 6.548695e-11 7: 2.4790247e-06 8: 7.249528e-05 9: 1.4312126e-10
Test Set Accuracy	0.9887	0.9859

SNR values for all five layers after the attack

snrWeights	1.5e07	1.7e05	1.2e04	6.8e03	3.2e06
snrBiases	5.1e07	8.5e07	8.6e07	3.0e07	1.3e08

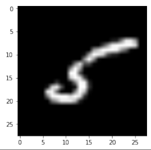
Table 4.3: The table shows the change in test set accuracy, model confidences, and SNR as a result of a model attack causing the model to misclassify an image of a "2"(seen above) as a "3"

	Before attack	After Attack
Model's confidences for image below 	0: 2.1229793e-10 1: 1.0674069e-07 2: 3.5628495e-10 **3: 0.9999988** 4: 6.9841445e-11 5: 2.9445167e-07 6: 1.6943589e-15 7: 4.1809645e-09 8: 2.4631994e-08 **9: 6.48088e-07**	0: 1.4047791e-07 1: 4.796148e-05 2: 2.920485e-07 **3: 0.42854732** 4: 6.0831695e-07 5: 7.071112e-05 6: 5.602939e-13 7: 2.9050916e-06 8: 5.801593e-06 **9: 0.5713242**
Test Set Accuracy	0.9885	0.9874

SNR values for all five layers after the attack

snrWeights	1.8e07	1.4e05	8.5e03	5.8e03	2.3e06
snrBiases	2.3e07	8.3e07	6.1e07	3.3e07	7.2e07

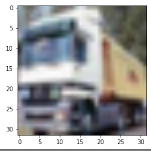
Table 4.4: The table shows the change in test set accuracy, model confidences, and SNR as a result of causing the model to misclassify an image of a "3"(seen above) as a "9"

	Before attack	After Attack
Model's confidences for image below 	0: 3.4765094e-08 1: 2.6511995e-07 2: 1.3529275e-08 3: 2.066145e-05 4: 1.52946e-07 **5: 0.9998971** 6: 5.426046e-06 7: 9.733825e-10 **8: 7.6096956e-05** 9: 2.1229043e-07	0: 1.685712e-06 1: 8.185025e-06 2: 1.5557762e-06 3: 0.0009712299 4: 6.3152766e-06 **5: 0.39503062** 6: 0.00024389257 7: 1.0351252e-07 **8: 0.60373235** 9: 4.1292606e-06
Test Set Accuracy	0.9874	0.9768

SNR values for all five layers after the attack

snrWeights	3.1e07	4.4e05	1.5e04	7.9e03	4.3e06
snrBiases	3.2e07	1.0e08	9.0e07	4.6e07	6.6e07

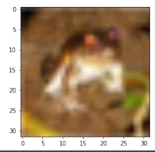
Table 4.5: The table shows the change in test set accuracy, model confidences, and SNR as a result of causing the model to misclassify an image of a "5"(seen above) as a "8"

	Before attack	After Attack
Model's confidences for the image below 	**'airplane': 7.823588e-06** 'automobile': 8.129067e-05 'bird': 6.416406e-08 'cat': 9.1138475e-08 'deer': 1.531015e-10 'dog': 1.4233449e-09 'frog': 1.7340601e-10 'horse': 6.433439e-08 'ship': 1.6490398e-07 **'truck': 0.9999106**	**'airplane': 0.5281227** 'automobile': 0.010194783 'bird': 0.002227325 'cat': 0.0024868809 'deer': 0.00013068119 'dog': 0.00023538098 'frog': 3.528182e-05 'horse': 0.0014832643 'ship': 0.0032511211 **'truck': 0.4518326**
Test Set Accuracy	0.8144	0.8127

SNR values for all seven layers after the attack

snrWeights	3.5e05	1.3e05	2.3e05	9.0e04	4.6e04	1.8e04	1.7e06
snrBiases	8.8e06	5.0e06	4.3e07	2.6e08	3.4e08	2.3e09	4.3e09

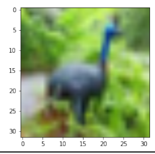
Table 4.6: The table shows the change in test set accuracy, model confidences, and SNR as a result of causing the model to misclassify an image of a "truck"(seen above) as an "airplane"

	Before attack	After Attack
Model's confidences (for image below) 	'airplane': 3.8550493e-06 'automobile': 1.6085962e-05 'bird': 0.0065517407 'cat': 0.060370855 'deer': 0.0031663193 'dog': 0.089471415 **'frog': 0.8396752** 'horse': 0.0006368972 'ship': 1.0016331e-05 **'truck': 9.7538614e-05**	'airplane': 0.00044484905, 'automobile': 0.0072038164, 'bird': 0.007921055, 'cat': 0.14714721, 'deer': 0.0010514394, 'dog': 0.112544, **'frog': 0.12083006**, 'horse': 0.013159411, 'ship': 0.00043517363, **'truck': 0.58926296**
Test Set Accuracy	0.8104	0.8009

SNR values for all seven layers after the attack

snrWeights	2.4e05	1.3e05	2.4e05	1.4e05	5.5e04	1.6e04	3.4e06
snrBiases	1.1e06	1.2e06	5.7e06	3.0e07	5.7e07	1.4e09	2.7e09

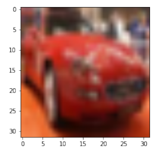
Table 4.7: The table shows the change in test set accuracy, model confidences, and SNR as a result of causing the model to misclassify an image of a "frog"(seen above) as a "truck"

	Before attack	After Attack
Model's confidences (for image below) 	'airplane': 0.043366294 'automobile': 0.012460392 **'bird': 0.61944956** 'cat': 0.045923192 'deer': 0.11433219 'dog': 0.0326852 'frog': 0.04043278 'horse': 0.060517326 **'ship': 0.0010289092** 'truck': 0.029804153	'airplane': 0.14039211 'automobile': 0.078225605 **'bird': 0.123359516** 'cat': 0.013718861 'deer': 0.026036164 'dog': 0.003287337 'frog': 0.009228582 'horse': 0.015589258 **'ship': 0.44568005** 'truck': 0.14448254
Test Set Accuracy	0.8015	0.8005

SNR values for all seven layers after the attack

snrWeights	3.0e05	1.8e05	3.8e05	1.8e05	8.0e04	1.6e04	4.2e06
snrBiases	7.9e06	3.2e06	6.1e07	1.5e08	1.5e08	1.7e09	2.6e09

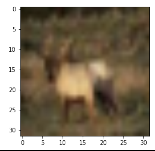
Table 4.8: The table shows the change in test set accuracy, model confidences, and SNR as a result of causing the model to misclassify an image of a "bird"(seen above) as a "ship"

	Before attack	After Attack
Model's confidences (for image below) 	'airplane': 1.10593575e-07 **'automobile': 0.9994154** 'bird': 1.8908064e-08 'cat': 0.00043956953 'deer': 7.904392e-10 'dog': 1.2587011e-06 **'frog': 3.3992092e-06** 'horse': 8.246504e-08 'ship': 2.189942e-06 'truck': 0.00013788207	'airplane': 8.353529e-05, **'automobile': 0.12633878**, 'bird': 0.0007574303, 'cat': 0.05321939, 'deer': 0.00017595025, 'dog': 0.0044297827, **'frog': 0.795922**, 'horse': 0.0009875748, 'ship': 0.0019886817, 'truck': 0.016096802
Test Set Accuracy	0.8155	0.8173

SNR values for all seven layers after the attack

snrWeights	5.7e05	7.0e04	1.6e05	8.9e04	4.7e04	2.2e04	8.4e06
snrBiases	9.1e06	2.5e06	2.6e07	3.2e08	5.9e08	5.5e09	6.7e09

Table 4.9: The table shows the change in test set accuracy, model confidences, and SNR as a result of causing the model to misclassify an image of an "automobile"(seen above) as a "frog"

	Before attack	After Attack
Model's confidences (for image below) 	'airplane': 6.620496e-08 'automobile': 2.1326816e-09 'bird': 0.00031115694 **'cat': 0.0001811466** **'deer': 0.9988481** 'dog': 0.00019735684 'frog': 0.0001390991 'horse': 0.00032315703 'ship': 7.0208017e-10 'truck': 4.9061755e-09	'airplane': 4.1886196e-05 'automobile': 1.827574e-05 'bird': 0.013111923 **'cat': 0.4491951** **'deer': 0.30823353** 'dog': 0.18875965 'frog': 0.032971602 'horse': 0.007664471 'ship': 1.6148313e-06 'truck': 1.94484e-06
Test Set Accuracy	0.8092	0.8036

SNR values for all seven layers after the attack

snrWeights	2.8e05	2.0e05	3.2e05	1.6e05	4.5e04	1.4e04	1.0e06
snrBiases	4.2e06	5.8e06	3.7e07	2.7e08	3.4e08	3.0e09	3.8e09

Table 4.10: The table shows the change in test set accuracy, model confidences, and SNR as a result of causing the model to misclassify an image of a "deer"(seen above) as a "cat"

5. PROPERTIES OF MODEL-ATTACK NOISE

In the previous chapter, we have demonstrated our strategy for the model attack. It changes the model parameters in minimal ways to compromise the CNN. These modifications can be treated as noise introduced due to the model attack, and if we can detect this model-attack noise in the presence of random noise, then we will have a way to detect the model attack. In this chapter, we compare the properties of the model-attack noise with those of Gaussian random noise. Specifically, we examine the BER (Bit Error Rates) in the model weights and the distribution of the model-attack noise values.

5.1 Bit Error Rates (BER)

BER values are used to evaluate the performance of a communication system. They represent the probability that the received bit is different from what was transmitted. We consider the model-attack noise as analogous to the noise that occurs during transmission, the original weights as the transmitted signal, and the modified weights as the received signal. We then compute the BER value by comparing the original weights and the modified weights.

We convert all the weight values into their binary representations and compare each of the bits in the original weights with the corresponding bits in the modified weights. The number of mismatches among the bits gives us the number of bit errors, which we divide by the total number of bits to compute the BER values. We do this for every bit position and record the corresponding BER values.

Figures 5.1-5.10 show the BER values for the various bit positions due to the presence of model-attack noise. Since we represent the weight values with a 12 bit resolution, the x-axis in each of the plots contains 12 values corresponding to the 12 bit positions. The BER values corresponding to the different layers in the CNN model have been shown separately in the figures. It is seen that the BER values for the most significant bit position are always the least, and the values steadily increase as we move towards the least significant bit position. This is same as what can be expected from Gaussian random noise, so BERs do not offer a way for us to detect model-attack noise in the presence of Gaussian random noise.

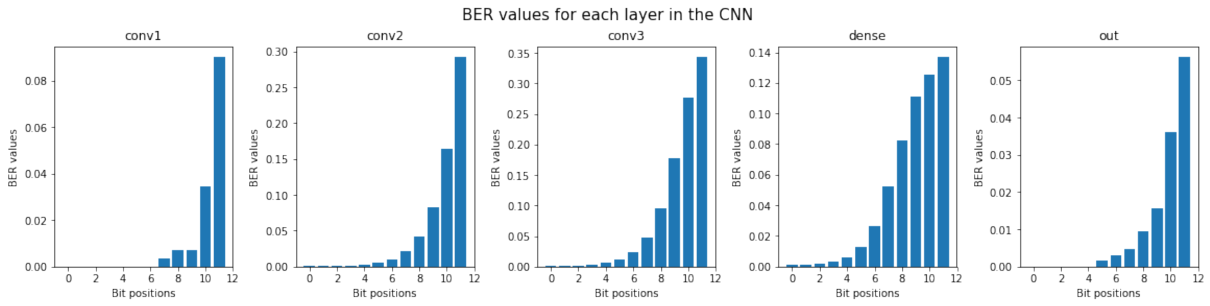


Figure 5.1: BER values for model-attack noise produced when a model was compromised to misclassify an image of a “5” as a “3”. The BER value is lowest for the most significant bit and steadily increases as we move to less significant bit positions.

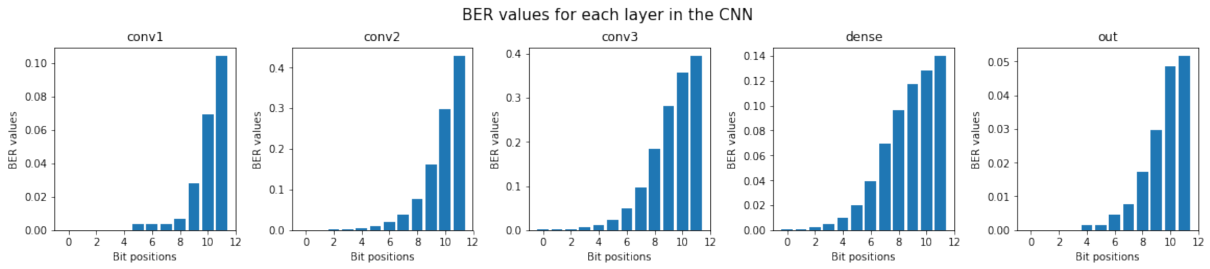


Figure 5.2: BER values for model-attack noise produced when a model was compromised to misclassify an image of a “2” as a “3”.

5.2 Distribution of model-attack noise

In the previous section, we observed that the BER values due to model-attack noise are very similar to what can be expected due to Gaussian random noise. Next, we compare model-attack noise with Gaussian random noise of comparable mean and variance. We use the mean and standard deviation for the model-attack noise in each layer of the CNN and use these to generate the corresponding Gaussian noise. We then visualize all the noise values using strip plots. In Figures

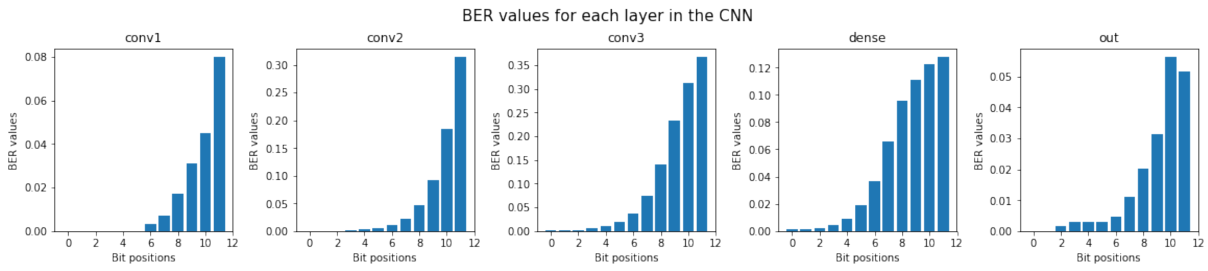


Figure 5.3: BER values for model-attack noise produced when a model was compromised to misclassify an image of a “4” as a “7”.

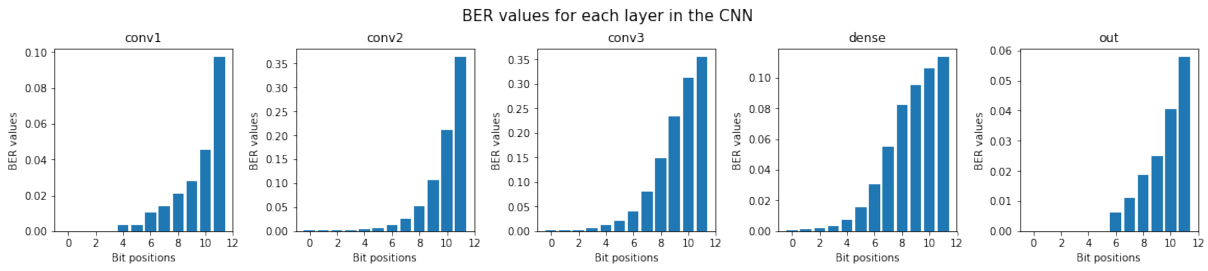


Figure 5.4: BER values for model-attack noise produced when a model was compromised to misclassify an image of a “5” as a “8”.

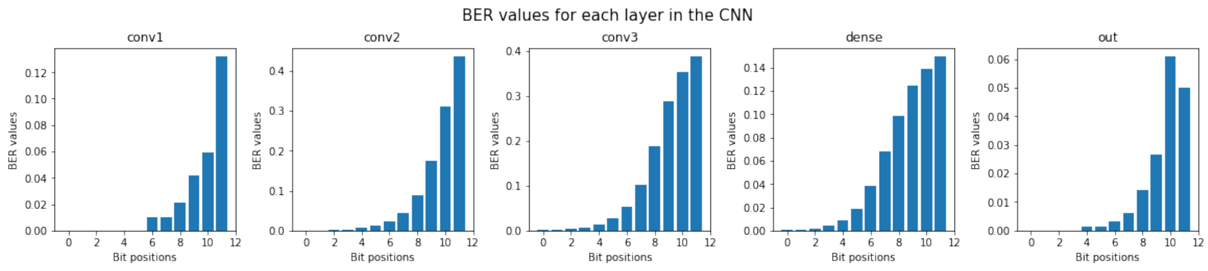


Figure 5.5: BER values for model-attack noise produced when a model was compromised to misclassify an image of a “3” as a “9”.

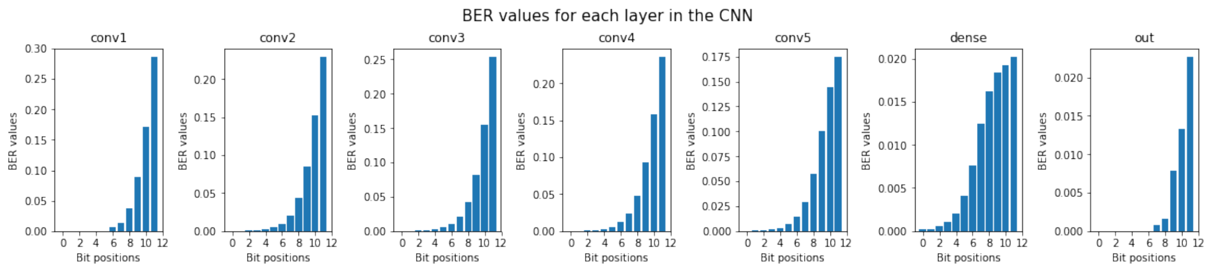


Figure 5.6: BER values for model-attack noise produced when a model was compromised to misclassify an image of a “bird” as a “ship”.

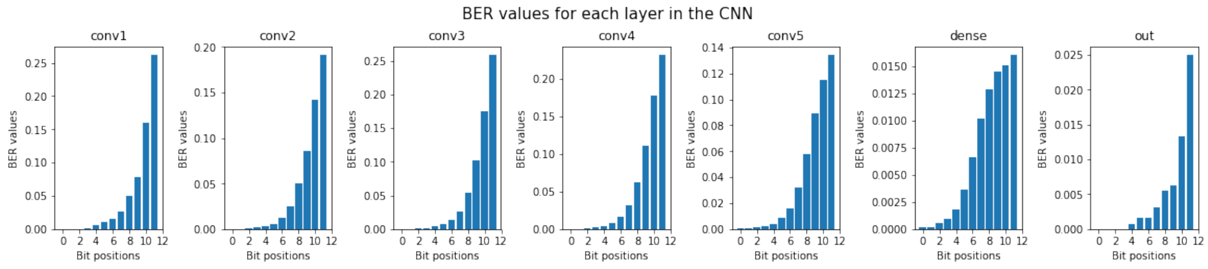


Figure 5.7: BER values for model-attack noise produced when a model was compromised to misclassify an image of an “automobile” as a “frog”.

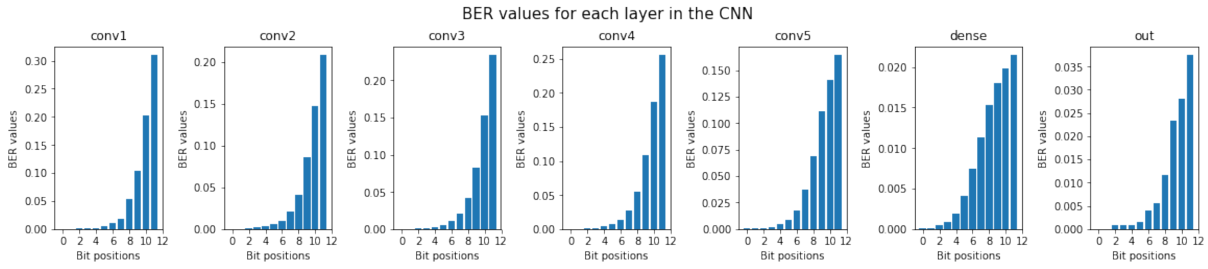


Figure 5.8: BER values for model-attack noise produced when a model was compromised to misclassify an image of a “deer” as a “cat”.

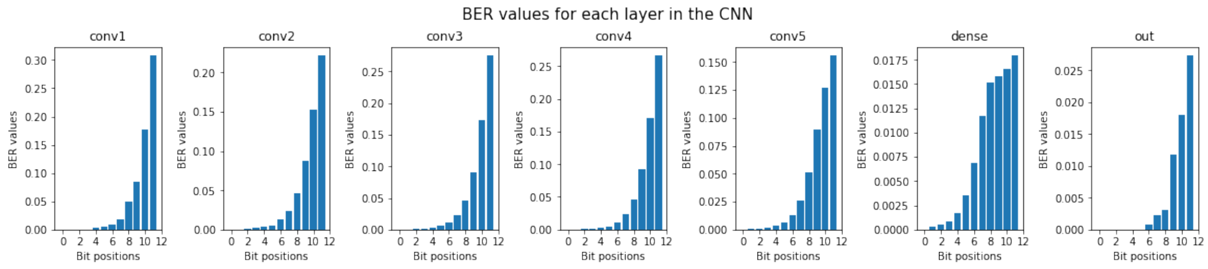


Figure 5.9: BER values for model-attack noise produced when a model was compromised to misclassify an image of a “frog” as a “truck”.

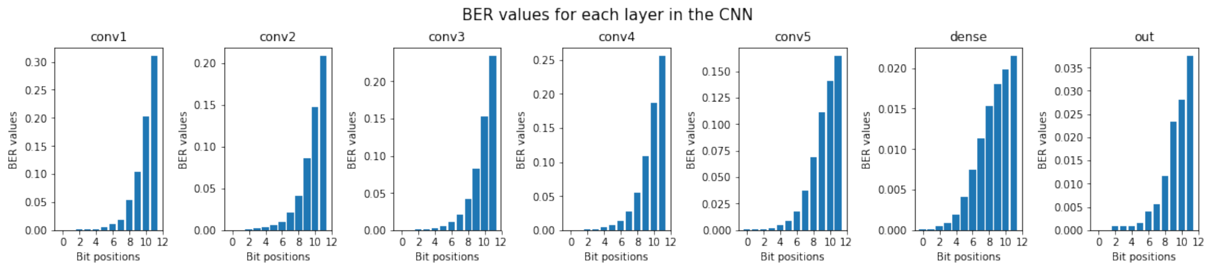


Figure 5.10: BER values for model-attack noise produced when a model was compromised to misclassify an image of a “truck” as an “airplane”.

5.11-5.20, the strip plots on the left correspond to model-attack noise and those on the right correspond to Gaussian noise. The strips in the plots contain the noise values corresponding to the different layers in the CNN, and the x-axis indicates the layer of the CNN corresponding to the strip. It is seen that the model-attack noise is more concentrated around 0 and has a larger spread of values when compared to Gaussian noise.

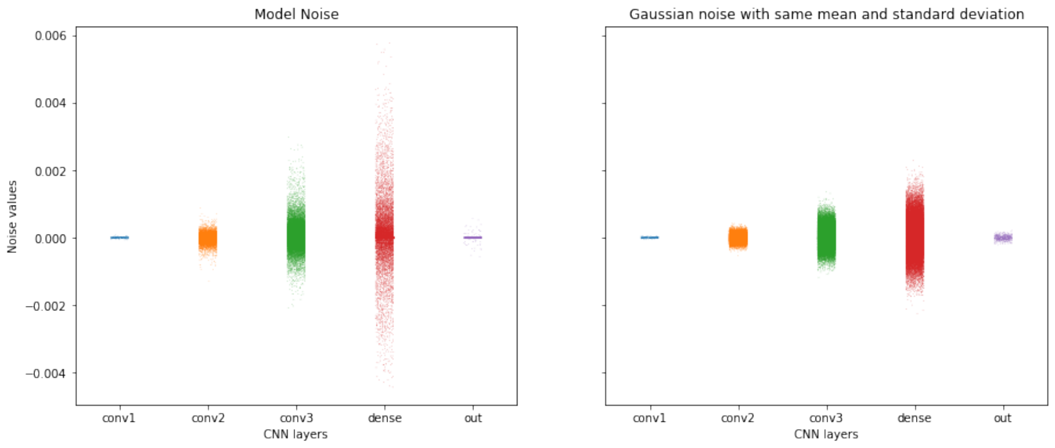


Figure 5.11: The model-attack noise values produced when a model was compromised to misclassify an image of a “5” as a “3”, along with the corresponding Gaussian noise values.

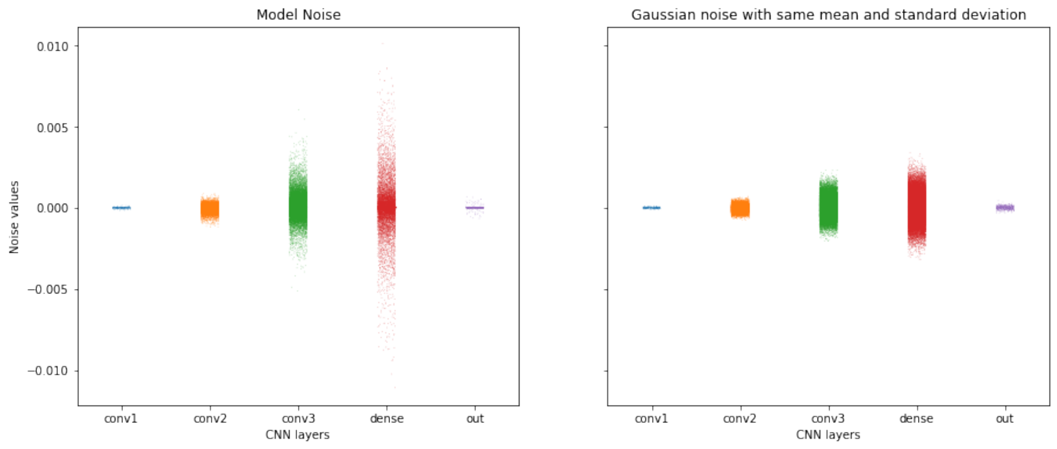


Figure 5.12: The model-attack noise values produced when a model was compromised to misclassify an image of a "2" as a "3", along with the corresponding Gaussian noise values.

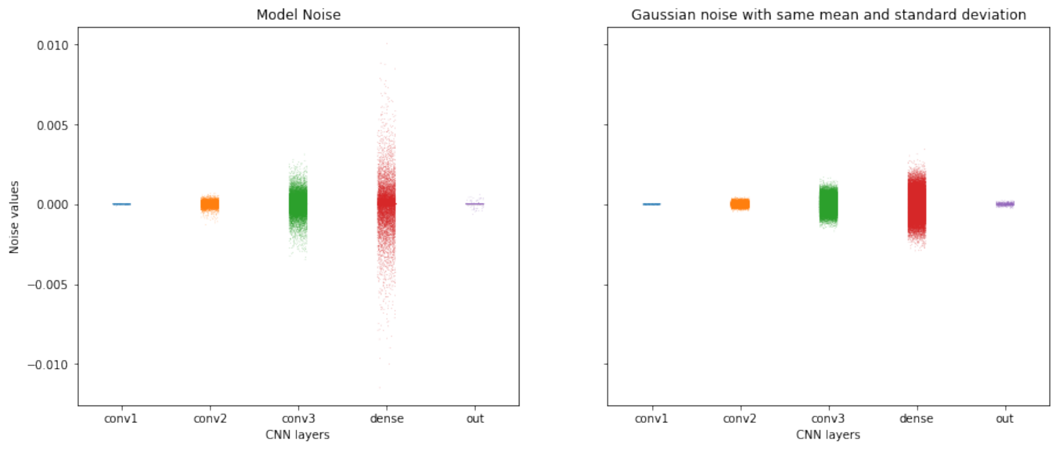


Figure 5.13: The model-attack noise values produced when a model was compromised to misclassify an image of a "4" as a "7", along with the corresponding Gaussian noise values.

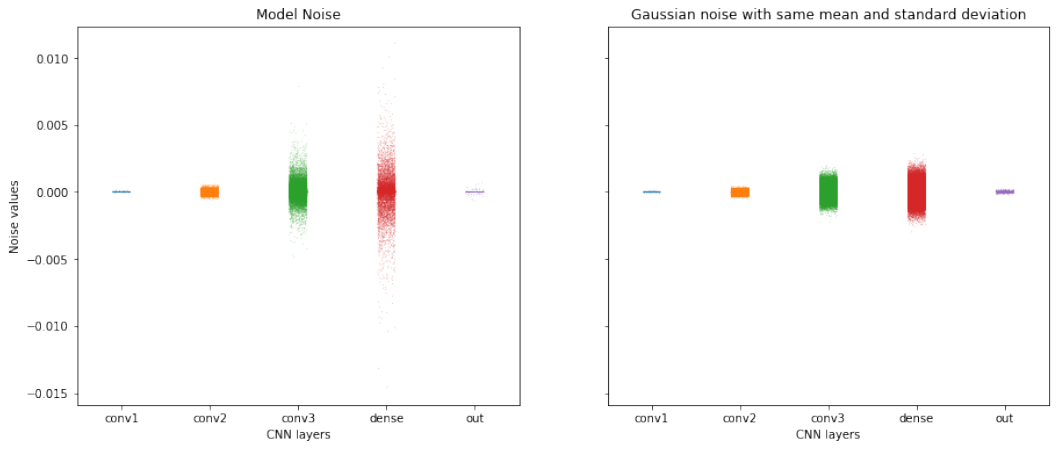


Figure 5.14: The model-attack noise values produced when a model was compromised to misclassify an image of a "5" as a "8", along with the corresponding Gaussian noise values.

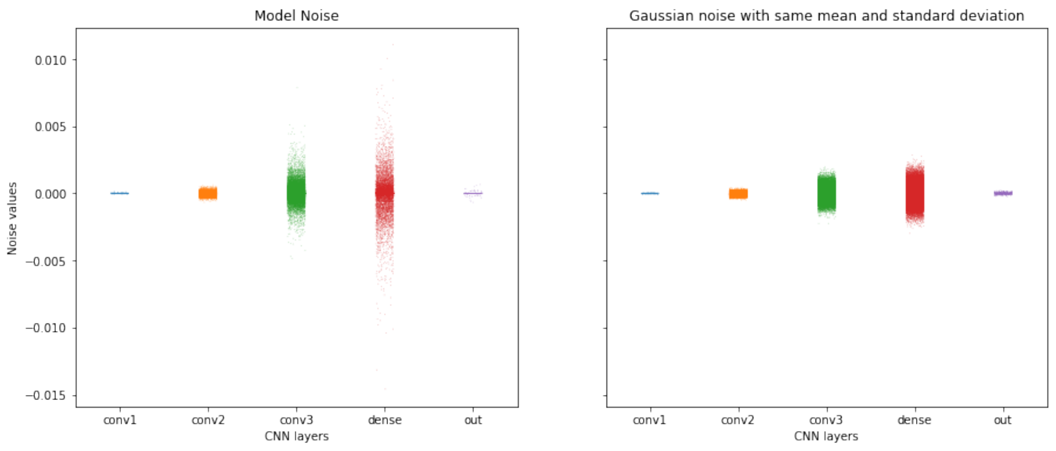


Figure 5.15: The model-attack noise values produced when a model was compromised to misclassify an image of a "3" as a "9", along with the corresponding Gaussian noise values.

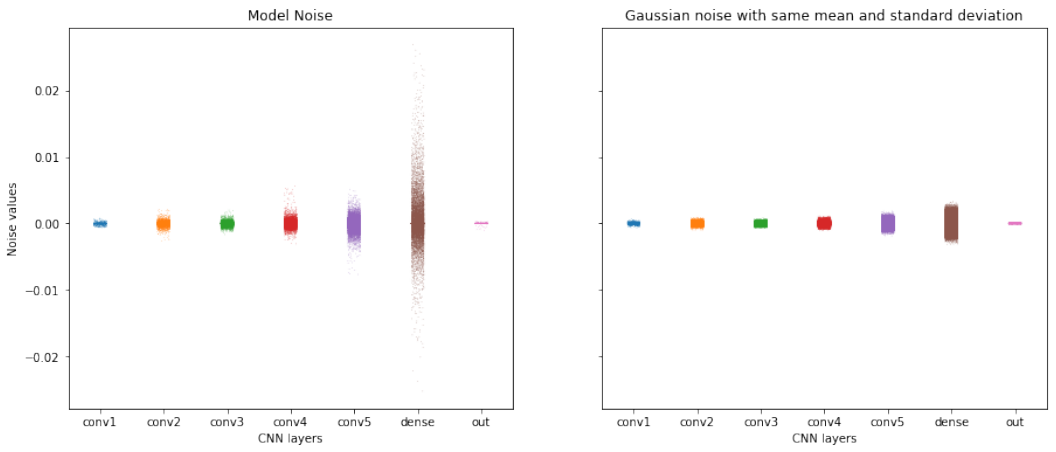


Figure 5.16: The model-attack noise values produced when a model was compromised to misclassify an image of a "bird" as a "ship", along with the corresponding Gaussian noise values.

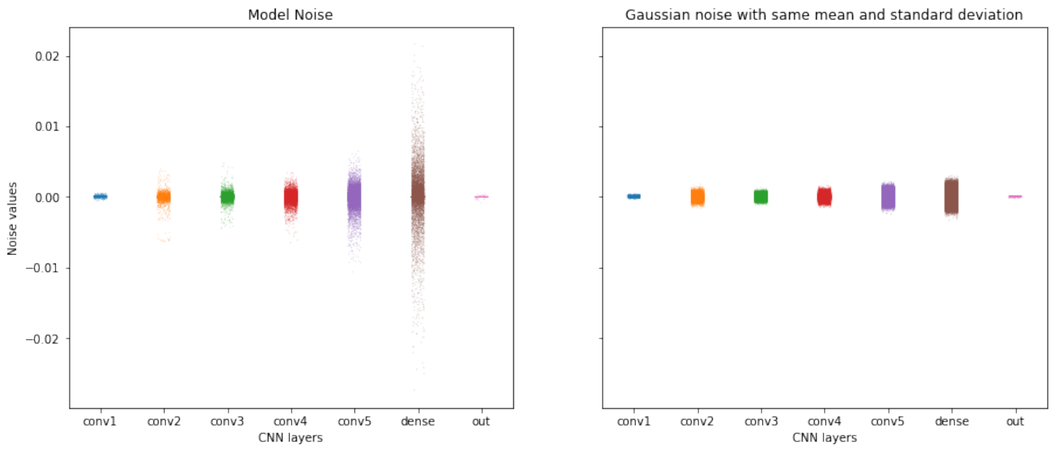


Figure 5.17: The model-attack noise values produced when a model was compromised to misclassify an image of an "automobile" as a "frog", along with the corresponding Gaussian noise values.

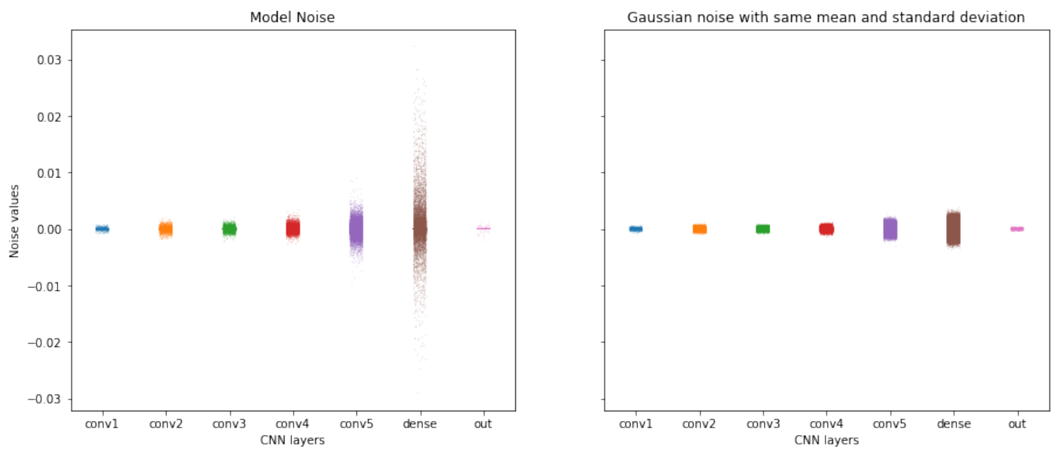


Figure 5.18: The model-attack noise values produced when a model was compromised to misclassify an image of a "deer" as a "cat", along with the corresponding Gaussian noise values.

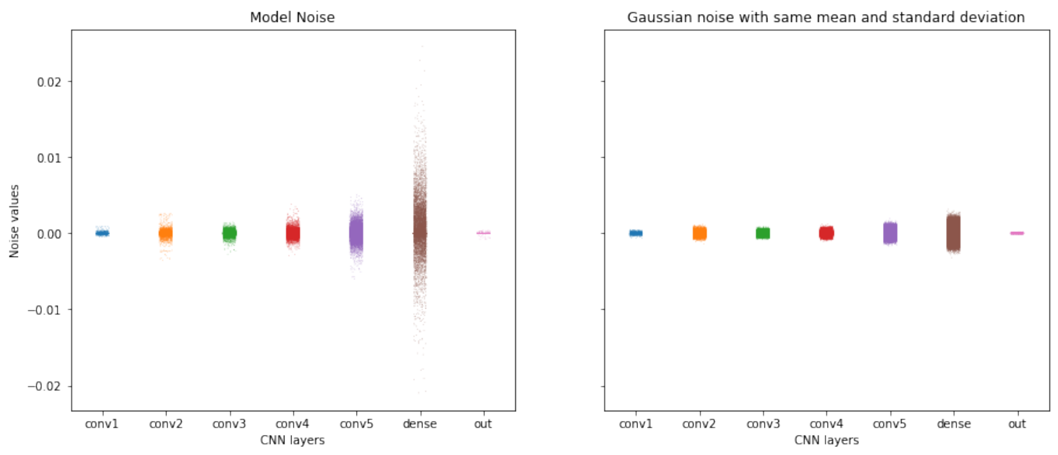


Figure 5.19: The model-attack noise values produced when a model was compromised to misclassify an image of a "frog" as a "truck", along with the corresponding Gaussian noise values.

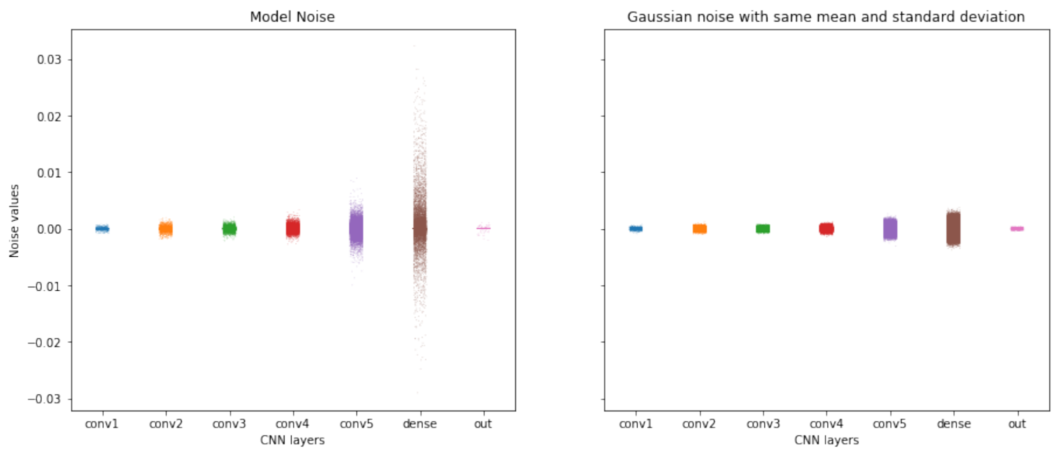


Figure 5.20: The model-attack noise values produced when a model was compromised to misclassify an image of a "truck" as an "airplane", along with the corresponding Gaussian noise values.

6. A STRATEGY TO DETECT THE MODEL ATTACK

In the last chapter, we observed that model-attack noise values are more concentrated around zero and have a larger range when compared to Gaussian random noise with comparable mean and standard deviation. Since the noise in the real world can be modeled using a Gaussian distribution, these distinguishing characteristics of the model-attack noise can help us detect the model attack in the presence of random noise. In this chapter, we devise a strategy to detect the model attack based on this observation and evaluate its effectiveness.

6.1 Kurtosis

Kurtosis is a measure of peakedness and tailedness of a distribution. In other words, it quantifies the sharpness of the peak of the distribution and the likelihood that the samples drawn from it belong to the its tails. Kurtosis is defined as the standardized fourth population moment about the mean [36] and can be represented mathematically as shown below.

$$\beta_2 = \frac{E(X-\mu)^4}{(E(X-\mu)^2)^2} = \frac{m_4}{\sigma^4}$$

where E is the expectation operator,
μ is the mean,
m_4 is the fourth moment about the mean, and
σ is the standard deviation.
Replacing the values in the above equation with their sample equivalents allows us to estimate the kurtosis value of a distribution by using the observations drawn from it.

$$b_2 = \frac{\sum(X_i-\overline{X})^4/n}{(\sum(X-\overline{X})^2/n)^2}$$

where b_2 is the sample kurtosis,
$\overline{X}$ is the sample mean, and

n is the number of observations.

The normal distribution has a kurtosis of 3, and we use this observation to detect the presence of model-attack noise.

6.2 Our Strategy

We choose kurtosis as the basis for our strategy since the model-attack noise values are heavily concentrated near the mean and have a larger range compared to Gaussian noise with the same variance. Kurtosis, being a measure of the peakedness and tailedness, quantifies exactly these characteristics. We measure the sample kurtosis, b_2, for the noise values in the model parameters and compare it with 3, the expected value for a normal distribution. If the difference is too large, we conclude that model-attack noise is present.

6.3 Experiments

Practical devices contain non-idealities and noise is inevitable, and attackers can use this noise as camouflage for their malicious perturbations to the model parameters. We model these non-idealities as Gaussian random noise present in the model parameters. We combine Gaussian noise with noise generated due to the model attack to simulate noise in the scenario where the model has been compromised. We then evaluate whether our strategy is able to detect the presence of model-attack noise.

We carry out the model attack using the Loss function based model attack strategy as described in Chapter 4 and record the generated model-attack noise values. Then, we generate Gaussian noise values with a wide range of variance values. From these, we create samples containing both model-attack noise and Gaussian noise as well as samples containing only Gaussian noise. We compute the kurtosis values for these samples and show their variation with respect to the variance of the Gaussian noise in Figures 6.1-6.6.

We see that a threshold of 3.04 for the kurtosis value separates the samples containing model-attack noise from those containing only Gaussian random noise. We also notice that there exists a maximum variance value beyond which the strategy is unable to distinguish the samples that contain model-attack noise from those that do not. We refer to this as the maximum tolerable noise variance for our strategy, and beyond this value, the strategy becomes ineffective. We find that this value is around 1e-05 for the examples we consider.

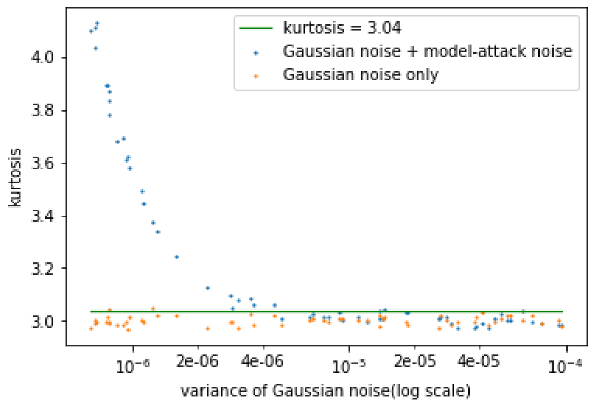


Figure 6.1: The model-attack noise was generated as a result of causing a model to misclassify an image of a “5” as a “3”, and has a variance of 1.5e-07. It is seen that maximum tolerable noise variance is 4e-06.

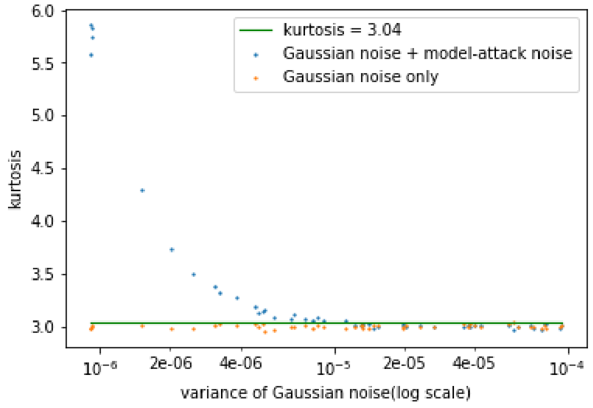


Figure 6.2: The model-attack noise was generated as a result of causing a model to misclassify an image of a “4” as a “7”, and has a variance of 2.93e-07. It is seen that the maximum tolerable noise variance is 1e-05.

For our strategy to be effective, we need it to reliably detect model-attack noise when it is present and also indicate that it is absent when only Gaussian random noise is present. The errors

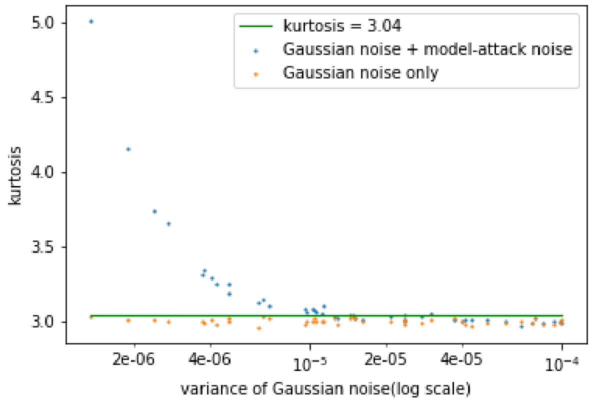


Figure 6.3: The model-attack noise used here has a variance of 4.19e-07, and was generated as a result of causing a model to misclassify an image of a "2" as a "3". It is seen that the maximum tolerable noise variance is 1e-5.

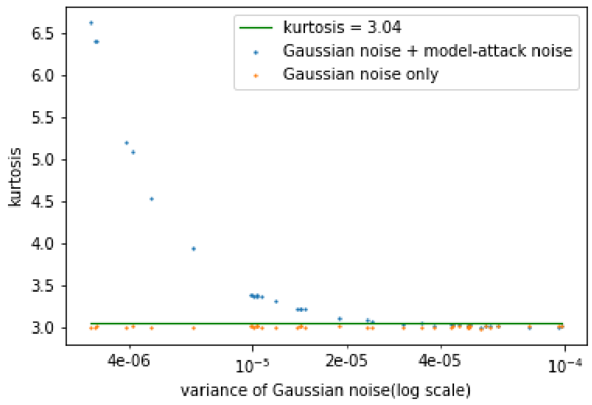


Figure 6.4: The model-attack noise used here has a variance of 3.93e-07, and was generated as a result of causing a model to misclassify an image of a "automobile" as a "frog". It is seen that the maximum tolerable noise variance is 2e-5.

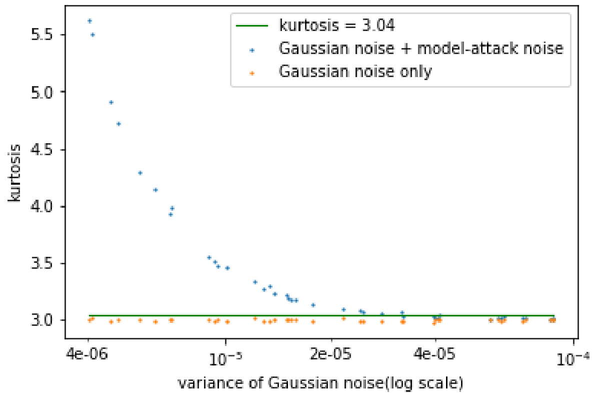


Figure 6.5: The model-attack noise used here has a variance of 4.23e-07, and was generated as a result of causing a model to misclassify an image of a "bird" as a "ship". It is seen that the maximum tolerable noise variance is 4e-5.

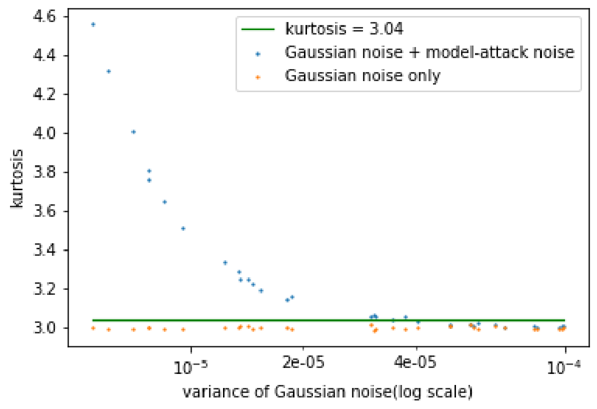


Figure 6.6: The model-attack noise used here has a variance of 4.32e-07, and was generated as a result of causing a model to misclassify an image of a "truck" as a "airplane". It is seen that the maximum tolerable noise variance is 4e-5.

where the strategy fails to detect the model-attack noise even though it is present are referred to as false negatives. False positives occur when the strategy indicates that model-attack noise is present when it isn't. So, to evaluate the effectiveness of our strategy, we need to check the rate of false positives and false negatives.

If a noise sample containing model-attack noise has a kurtosis value that is lower than the threshold, then it is a false negative. To evaluate the false negative rate, we create noise samples containing both Gaussian random noise and model-attack noise and check how frequently the strategy produces a wrong "negative" result. Tables 6.1-6.4 show the variance of the Gaussian noise and kurtosis values for samples containing both Gaussian noise and model-attack noise. We see that the strategy produces the correct "positive" result for samples with low variance while higher variance values result in false negatives.

In addition to false negatives, we also need to evaluate the rate of false positives. If a sample containing only Gaussian noise has a kurtosis value greater than the threshold, then it is a false positive. So, to evaluate the false positive rate, we repeatedly generate samples containing only Gaussian random noise and check how frequently the strategy produces a wrong "positive" result. We generate two sets of samples containing Gaussian noise corresponding to the model architectures used for MNIST and CIFAR-10. Tables 6.5 and 6.6 show the variance of the Gaussian noise

Variance of Gaussian noise	Kurtosis	Test Result
5.36e-09	33.356	Positive
6.12e-09	33.030	Positive
6.56e-07	4.082	Positive
7.11e-07	3.912	Positive
7.93e-07	3.816	Positive
8.52e-07	3.750	Positive
1.12e-06	3.536	Positive
1.71e-06	3.213	Positive
4.99e-06	3.070	Positive
5.6e-06	2.997	Negative
8.34e-06	3.031	Negative
1.01e-05	3.010	Negative
1.18e-05	3.019	Negative
1.2e-05	3.040	Negative
1.43e-05	3.023	Negative
1.82e-05	2.979	Negative
2.3e-05	3.005	Negative
5.89e-05	3.023	Negative
8.77e-05	2.978	Negative

Table 6.1: The table shows the kurtosis values for Gaussian noise samples that also contain model-attack noise. The model-attack noise used here was generated as the result of compromising a model to misclassify an image of a “5” as a “3”.

Variance of Gaussian noise	Kurtosis	Test Result
5.32e-09	48.105	Positive
1.85e-07	21.245	Positive
2.37e-07	17.444	Positive
4.73e-07	9.516	Positive
7.64e-07	6.768	Positive
1.03e-06	5.332	Positive
2.57e-06	3.471	Positive
3.99e-06	3.290	Positive
4.02e-06	3.216	Positive
5.87e-06	3.142	Positive
5.9e-06	3.138	Positive
6.57e-06	3.088	Positive
6.77e-06	3.046	Positive
7.93e-06	3.097	Positive
9.24e-06	3.036	Negative
1.09e-05	3.031	Negative
1.3e-05	2.999	Negative
2.06e-05	2.999	Negative
2.53e-05	3.013	Negative
6.97e-05	3.002	Negative
9.3e-05	2.985	Negative

Table 6.2: The table shows the kurtosis values for Gaussian noise samples that also contain model-attack noise. The model-attack noise used here was generated as the result of compromising a model to misclassify an image of a “4” as a “7”.

Variance of Gaussian noise	Kurtosis	Test Result
1.85e-09	269.877	Positive
1.35e-07	152.374	Positive
9.99e-07	24.161	Positive
1.14e-06	20.157	Positive
2.54e-06	7.669	Positive
3.43e-06	5.815	Positive
3.86e-06	5.207	Positive
5.2e-06	4.431	Positive
5.47e-06	4.195	Positive
6.68e-06	3.813	Positive
7.68e-06	3.700	Positive
1.57e-05	3.166	Positive
1.93e-05	3.091	Positive
2.31e-05	3.068	Positive
2.53e-05	3.062	Positive
3.8e-05	3.031	Negative
4.05e-05	3.029	Negative
4.35e-05	3.014	Negative
4.56e-05	3.008	Negative
7.99e-05	3.016	Negative
8e-05	3.012	Negative
9.28e-05	3.006	Negative

Table 6.3: The table shows the kurtosis values for Gaussian noise samples that also contain model-attack noise. The model-attack noise used here was generated as the result of compromising a model to misclassify an image of an "automobile" as a "frog".

Variance of Gaussian noise	Kurtosis	Test Result
3.75e-09	279.236	Positive
8.93e-08	193.919	Positive
3.24e-07	93.526	Positive
8.71e-07	33.551	Positive
1.26e-06	20.945	Positive
1.98e-06	11.760	Positive
4.59e-06	5.001	Positive
5.2e-06	4.551	Positive
7.25e-06	3.890	Positive
1.07e-05	3.433	Positive
1.39e-05	3.266	Positive
1.93e-05	3.149	Positive
2.26e-05	3.107	Positive
4.07e-05	3.032	Negative
4.56e-05	3.017	Negative
5.79e-05	3.012	Negative
6.93e-05	3.010	Negative
7.42e-05	3.021	Negative
8.14e-05	3.017	Negative
8.68e-05	3.001	Negative
9.4e-05	3.016	Negative

Table 6.4: The table shows the kurtosis values for Gaussian noise samples that also contain model-attack noise. The model-attack noise used here was generated as the result of compromising a model to misclassify an image of a “bird” as a “ship”.

and kurtosis values for these samples. It is seen that there are virtually no false positives.

Variance of Gaussian noise	Kurtosis	Test Result
5.36e-09	3.015	Negative
6.12e-09	3.004	Negative
6.21e-07	3.011	Negative
6.56e-07	3.023	Negative
7.11e-07	2.996	Negative
8.52e-07	3.013	Negative
1.12e-06	2.999	Negative
1.22e-06	2.996	Negative
1.71e-06	3.009	Negative
4.99e-06	3.011	Negative
5.6e-06	2.976	Negative
8.34e-06	2.980	Negative
8.45e-06	3.035	Negative
1.01e-05	3.014	Negative
1.15e-05	2.996	Negative
1.18e-05	2.993	Negative
1.2e-05	3.000	Negative
1.43e-05	3.024	Negative
1.82e-05	3.029	Negative
2.3e-05	3.051	Positive
5.89e-05	3.002	Negative
8.77e-05	2.994	Negative

Table 6.5: Evaluating false positives using Gaussian noise samples generated based on the model architecture used for MNIST

We observe that the strategy effectively detects the samples containing model-attack noise only when the Gaussian noise has low variance. This is because the distinguishing features of the model-attack noise i.e. the large concentration of values near zero and higher range become less apparent from the distribution as the variance of the Gaussian noise increases.

Variance of Gaussian noise	Kurtosis	Test Result
3.37e-09	3.012	Negative
6.03e-09	3.001	Negative
3.67e-08	3.006	Negative
1.01e-07	2.998	Negative
8.88e-07	2.999	Negative
1.6e-06	2.995	Negative
1.61e-06	2.999	Negative
2.91e-06	3.005	Negative
3.2e-06	3.002	Negative
4.67e-06	3.007	Negative
6.47e-06	3.004	Negative
7.79e-06	3.018	Negative
1.05e-05	3.002	Negative
1.17e-05	3.000	Negative
1.33e-05	2.991	Negative
2.69e-05	3.006	Negative
3.24e-05	3.000	Negative
5.77e-05	3.009	Negative
6.61e-05	2.986	Negative
6.99e-05	3.010	Negative
7.16e-05	3.004	Negative
8.19e-05	3.001	Negative
9e-05	3.003	Negative
9.17e-05	2.997	Negative

Table 6.6: Evaluating false positives using Gaussian noise samples generated based on the model architecture used for CIFAR-10

7. VISUALIZING THE MODEL-ATTACK NOISE IN THE CONVOLUTIONAL LAYERS

In the previous chapter, we devised a strategy to detect the model attack based on the noise distribution. This strategy is effective only when the variance of the Gaussian random noise present in the model parameters is low. In this chapter, we visualize the model-attack noise values with the goal of identifying characteristics which can enable us to devise a more effective strategy for detecting the model attack.

Our model architecture is based on VGG [37], and the weight values in the convolutional layers are contained in 3*3 kernels of varying depths. The model attack adds noise to each of these weight values, so the model-attack noise in these kernels can be represented as a 3*3 matrix which has the same depth as the kernel. We split this 3D matrix into several 3*3 matrices, which we then convert into 9*1 vectors. We visualize these vectors in 2D using t-SNE [24]. This process is illustrated in Figure 7.1.

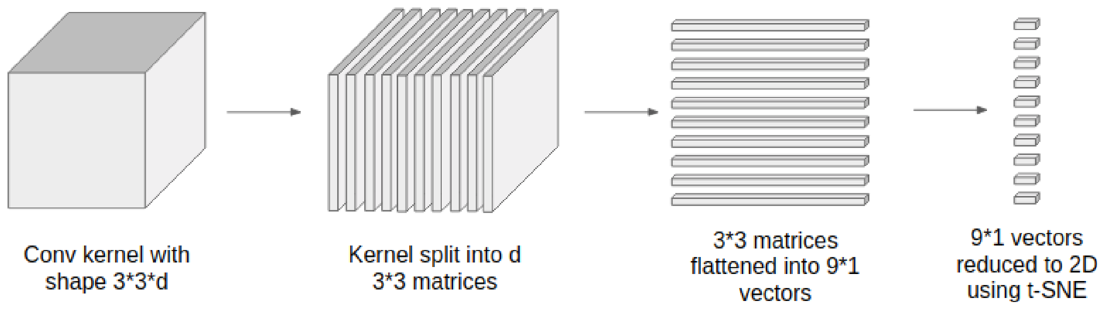

Conv kernel with shape 3*3*d
Kernel split into d 3*3 matrices
3*3 matrices flattened into 9*1 vectors
9*1 vectors reduced to 2D using t-SNE

Figure 7.1: The process of converting model-attack noise values into 2D

The t-SNE plots for the model-attack noise generated using the process described above are seen in Figures 7.3-7.6. The colors of the points in these figures correspond to the layers in the CNN to which they belong. We also plot a t-SNE visualization of 3*3 matrices containing Gaussian noise values in Figure 7.2. Observing the t-SNE plots, we see that the model-attack noise has more structure than Gaussian noise. In the t-SNE plots for model-attack noise, the points corresponding to the various layers are close to each other and form clusters. There also exists an oval cluster, which is distinct from the rest of the points. The points in this cluster correspond to 3*3 matrices of model-attack noise values that are either zero or very close to zero.

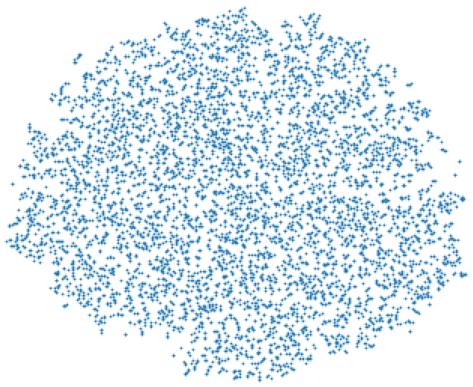


Figure 7.2: t-SNE plot for 9*1 vectors of Gaussian random noise.

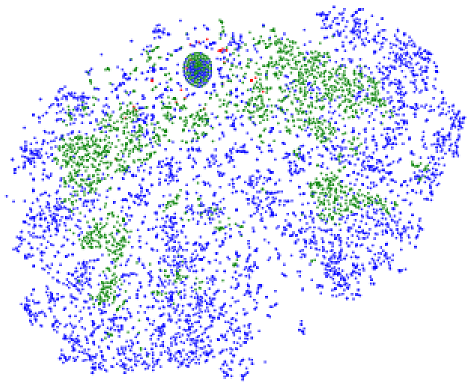


Figure 7.3: t-SNE plot for the model-attack noise produced as a result of compromising a model to misclassify an image of a “4” as a “7”.

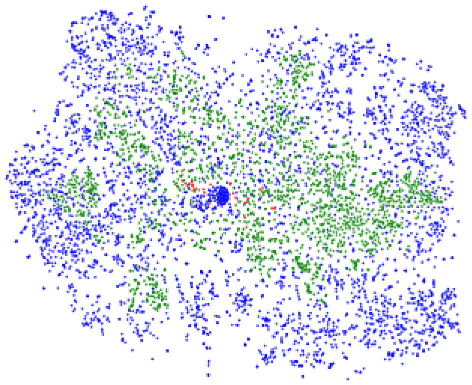


Figure 7.4: t-SNE plot for the model-attack noise produced as a result of compromising a model to misclassify an image of a “3” as a “9”.

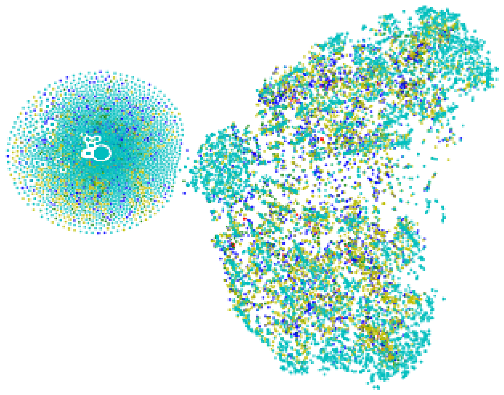


Figure 7.5: t-SNE plot for the model-attack noise produced as a result of compromising a model to misclassify an image of a “deer” as a “cat”.

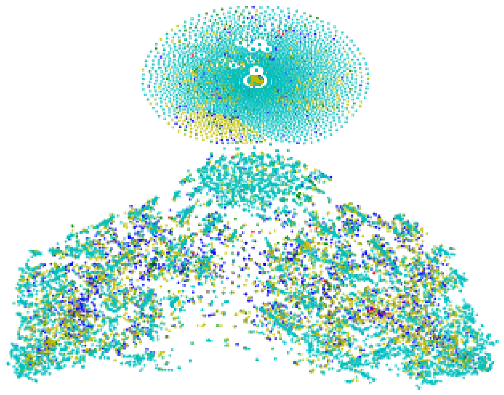


Figure 7.6: t-SNE plot for the model-attack noise produced as a result of compromising a model to misclassify an image of an “automobile” as a “frog”.

Next, we check if the structures identified by t-SNE still exist in the presence of random noise since this would enable us to detect the model attack. Figures 7.7-7.10 show the t-SNE plots for model-attack noise in the presence of Gaussian random noise with varying variance values. We see that the presence of Gaussian noise causes much of the structure in the model-attack noise to disappear. The structures become progressively harder to detect as the variance of Gaussian noise increases. When the variance of the Gaussian noise is around 1e-06, the t-SNE plot looks very similar to the t-SNE plot for Gaussian random noise, with no noticeable structure in it. This means that even though the strategy is effective in differentiating model-attack noise from Gaussian noise, it is not suitable for detecting model-attack noise in the presence of random noise.

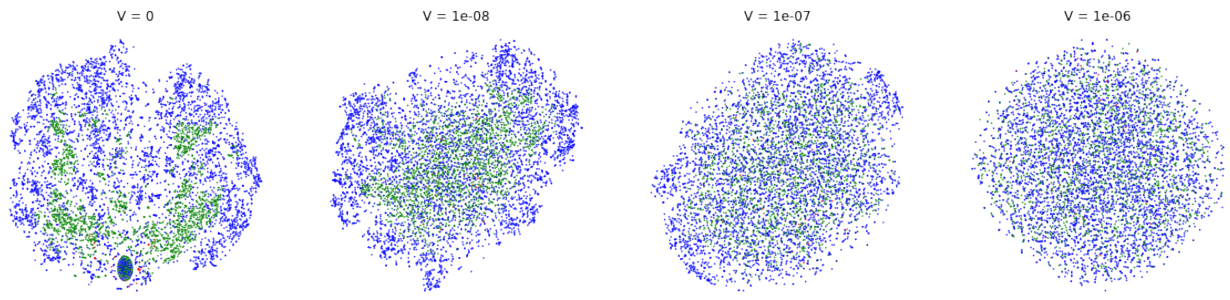


Figure 7.7: t-SNE plot for model-attack noise in the presence of Gaussian random noise with variance V. The model-attack noise visualized here was produced as a result of compromising a model to misclassify an image of a “4” as a “7”.

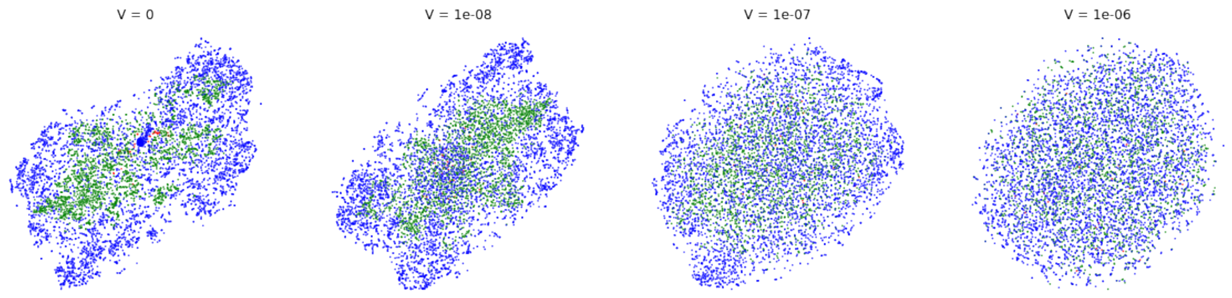


Figure 7.8: t-SNE plot for model-attack noise in the presence of Gaussian random noise with variance V. The model-attack noise visualized here was produced as a result of compromising a model to misclassify an image of a “3” as a “9”.

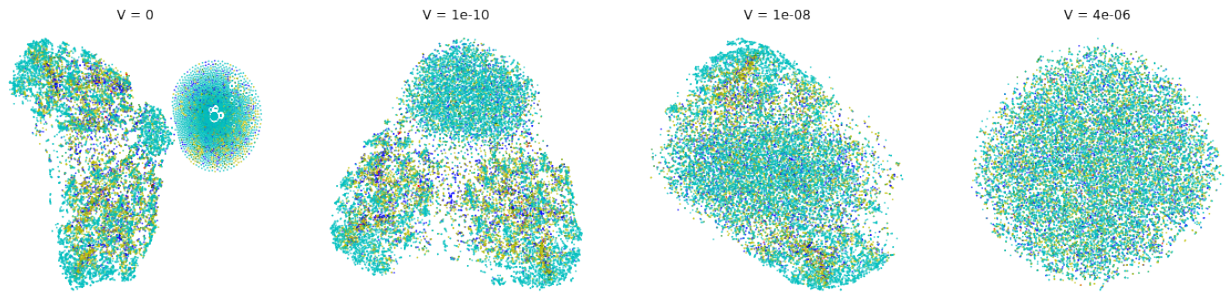


Figure 7.9: t-SNE plot for model-attack noise in the presence of Gaussian random noise with variance V. The model-attack noise visualized here was produced as a result of compromising a model to misclassify an image of a “deer” as a “cat”.

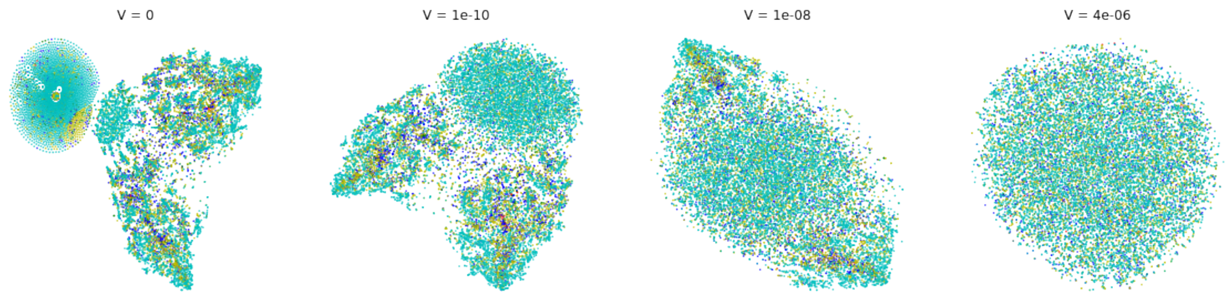


Figure 7.10: t-SNE plot for model-attack noise in the presence of Gaussian random noise with variance V. The model-attack noise visualized here was produced as a result of compromising a model to misclassify an image of an “automobile” as a “frog”.

8. ANALYZING THE MODEL'S PREDICTIONS BEFORE AND AFTER THE ATTACK

We have seen that the Loss function based model attack does not significantly change the model's test set accuracy. However, this does not mean that all of the model's predictions remain unchanged. Analyzing how the model's predictions change due to the attack might help us uncover patterns, enabling us to detect the model attack. In this chapter, we visualize the images in the dataset using t-SNE and observe how the attack affects the model's predictions.

We record the model's initial predictions for the various images in the training set. Then, we embed the training set images into a 2D plot using t-SNE and assign colors to the points in the plot based on the model's predictions for the corresponding images. After this, we carry out the model attack and then generate a new plot using the same t-SNE embedding as before but using the model's predictions after the attack to decide the colors of the points. Figures 8.2, 8.3, 8.5 and 8.6 show the plots corresponding to model attacks carried out on models trained on MNIST and CIFAR-10.

From the plots, it is seen that the model's predictions for most of the images remain unchanged, and there are no clear patterns which can help us detect the model attack. This also shows that the our attack strategy meets the requirement that the model's performance is not significantly changed due to the attack.

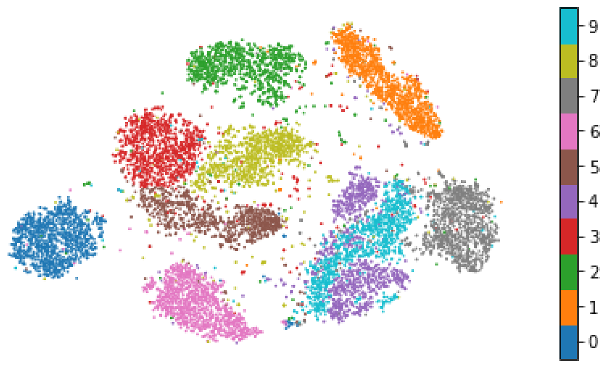
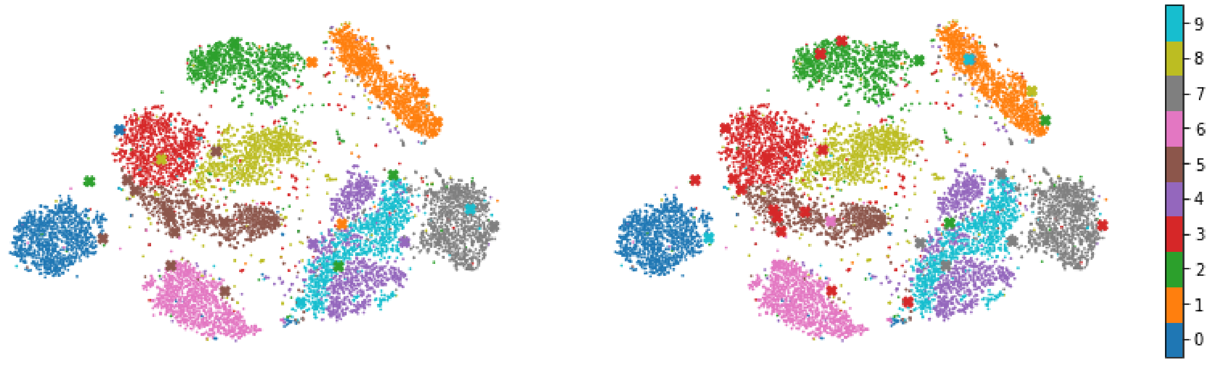


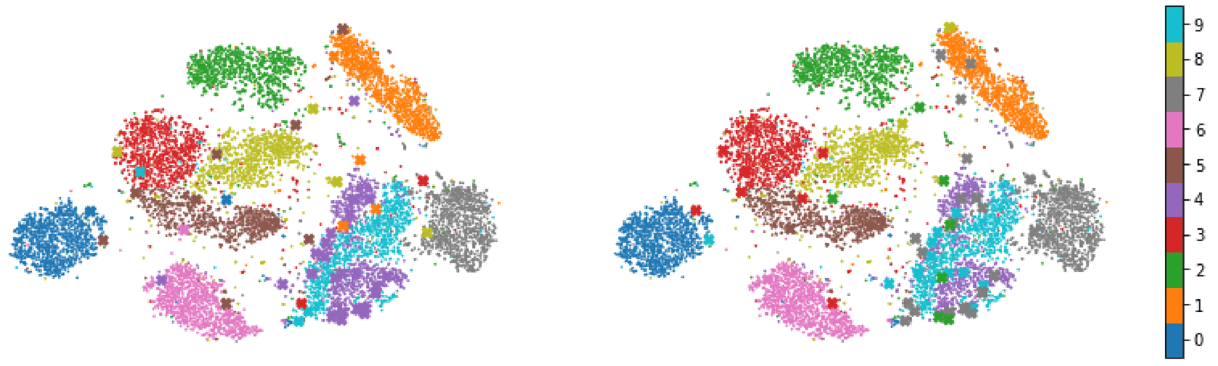
Figure 8.1: The images in the MNIST training set have been visualized using t-SNE. The colors of the points represent their ground truth labels.



(a) Before Attack

(b) After attack

Figure 8.2: t-SNE plots showing the model's predictions before and after a model attack which caused the model to misclassify an image of a "5" as a "3". The images for which the model's prediction changes are denoted by large ✖s.



(a) Before Attack

(b) After attack

Figure 8.3: t-SNE plots showing the model's predictions before and after a model attack which caused the model to misclassify an image of a "4" as a "7". The images for which the model's prediction changes are denoted by large ✖s.

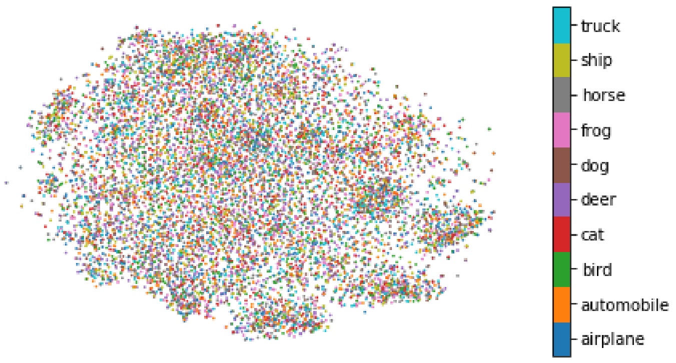
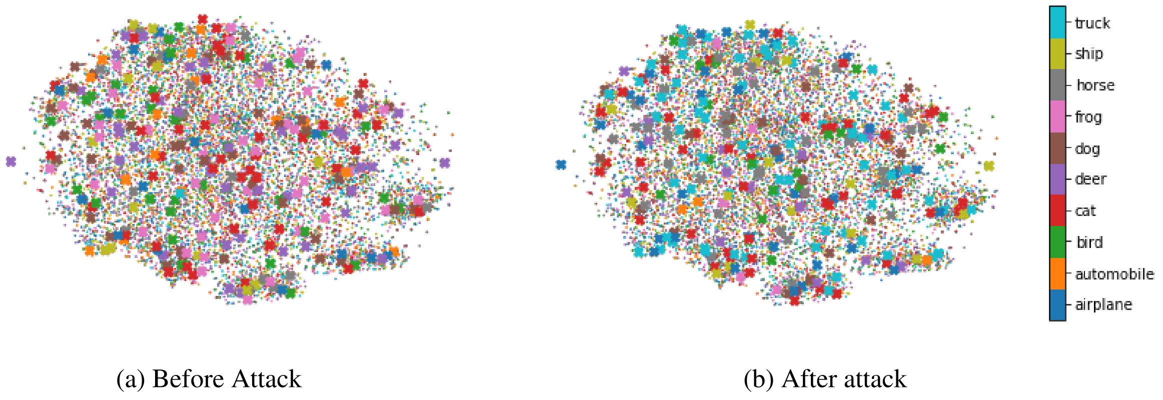


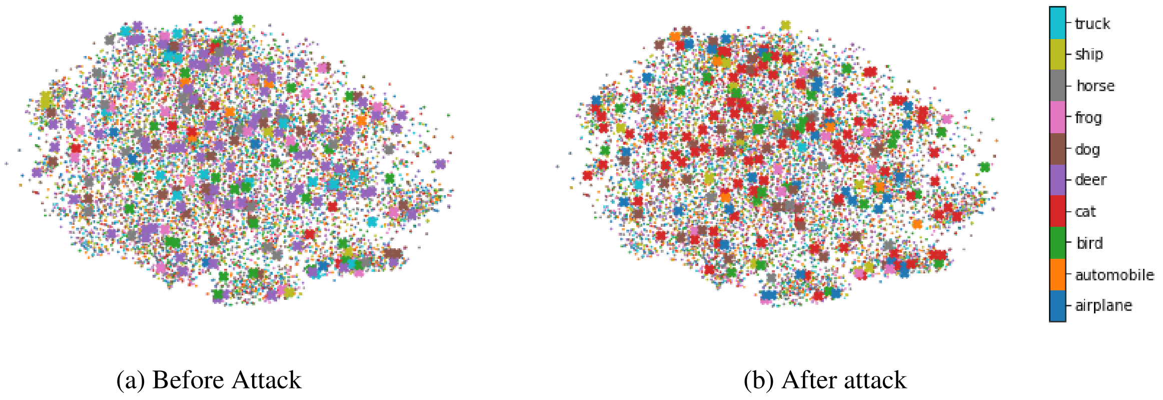
Figure 8.4: The images in the CIFAR-10 training set have been visualized using t-SNE. The colors of the points represent their ground truth labels.



(a) Before Attack

(b) After attack

Figure 8.5: t-SNE plots showing the model's predictions before and after a model attack which caused the model to misclassify an image of a "frog" as a "truck". The images for which the model's prediction changes are denoted by large ✖s.



(a) Before Attack

(b) After attack

Figure 8.6: t-SNE plots showing the model's predictions before and after a model attack which caused the model to misclassify an image of a "deer" as a "cat". The images for which the model's prediction changes are denoted by large ✖s.

9. CONCLUSIONS AND FUTURE WORK

In this study, we have analyzed the threat of a model attack when the CNN model parameters are stored in non-volatile memories where noise is inevitable. We have demonstrated that the model attack can be carried out on a CNN model with minimal changes to the model's parameters and performance, which makes such an attack hard to detect. This raises concerns for practical applications such as the self-driving car where the passengers' safety could be at risk if the model is compromised.

We studied the properties of the model-attack noise values and observed that they had a higher kurtosis value when compared to Gaussian noise. Based on this observation, we devised a strategy to detect the model attack in the presence of Gaussian random noise. We found that this strategy works well when the variance of Gaussian noise is low.

We have discussed the different techniques that we used to study how the model attack affects the model's parameters and predictions. We observed that the Bit error rates in the model weights due to the presence of model-attack noise are similar to those expected due to the presence of Gaussian random noise. We observed that a t-SNE plot of model-attack noise has more structure compared to a t-SNE plot of Gaussian random noise. We also analyzed the model's predictions before and after the attack and found that the model's predictions for most of the images remain unchanged. This shows that our attack strategy only has a minimal effect on the model's performance.

We defined the model attack problem by viewing it as the dual of the data attack problem and proposed the "Loss function based model attack strategy" to solve it. Our research focuses on this specific strategy to carry out the model attack, so further research is needed to understand the general characteristics of the model attack problem. Future work can focus on identifying alternate ways of compromising the model and strategies to detect the attacks. This would enable us to devise robust practical strategies to defend against the threat of the model attack.

REFERENCES

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

[2] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "ImageNet Large Scale Visual Recognition Challenge. arXiv: 1409.0575," 2014.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[4] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, pp. 3104–3112, 2014.

[5] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[6] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[7] D. Bahdanau, J. Chorowski, D. Serdyuk, P. Brakel, and Y. Bengio, "End-to-end attention-based large vocabulary speech recognition," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4945–4949, IEEE, 2016.

[8] B. Turovsky, "Found in translation: More accurate, fluent sentences in Google Translate." `https://blog.google/products/translate/found-translation-more-accurate-fluent-sentences-google-translate/`, 2016. [Online; accessed 16-June-2019].

[9] J. Schalkwyk, "An All-Neural On-Device Speech Recognizer." https://ai.googleblog.com/2019/03/an-all-neural-on-device-speech.html, 2019. [Online; accessed 16-June-2019].

[10] R. Mcmillan, "How skype used AI to build its amazing new language translator." https://www.wired.com/2014/12/skype-used-ai-build-amazing-new-language-translator/, 2014. [Online; accessed 24-June-2019].

[11] "Hey Siri: An on-device DNN-powered voice trigger for apples personal assistant." https://machinelearning.apple.com/2017/10/01/hey-siri.html, 2017. [Online; accessed 24-June-2019].

[12] B. Barrett, "The year Alexa grew up." https://www.wired.com/story/amazon-alexa-2018-machine-learning/, 2018. [Online; accessed 24-June-2019].

[13] I. Wallach, M. Dzamba, and A. Heifets, "Atomnet: a deep convolutional neural network for bioactivity prediction in structure-based drug discovery," *arXiv preprint arXiv:1510.02855*, 2015.

[14] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, no. 7639, p. 115, 2017.

[15] H. C. Hazlett, H. Gu, B. C. Munsell, S. H. Kim, M. Styner, J. J. Wolff, J. T. Elison, M. R. Swanson, H. Zhu, K. N. Botteron, *et al.*, "Early brain development in infants at high risk for autism spectrum disorder," *Nature*, vol. 542, no. 7641, p. 348, 2017.

[16] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.

[18] J. Vincent, “Openai’s dota 2 defeat is still a win for artificial intelligence.” `https://www.wired.com/story/amazon-alexa-2018-machine-learning/`, 2018. [Online; accessed 24-June-2019].

[19] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” *arXiv preprint arXiv:1511.06434*, 2015.

[20] L. A. Gatys, A. S. Ecker, and M. Bethge, “Image style transfer using convolutional neural networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2414–2423, 2016.

[21] A. Huang and R. Wu, “Deep learning for music,” *arXiv preprint arXiv:1606.04930*, 2016.

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

[23] N. Carlini and D. Wagner, “Towards evaluating the robustness of neural networks,” in *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 39–57, IEEE, 2017.

[24] L. v. d. Maaten and G. Hinton, “Visualizing data using t-SNE,” *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[25] Y. Liu, L. Wei, B. Luo, and Q. Xu, “Fault injection attack on deep neural network,” in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 131–138, IEEE, 2017.

[26] A. S. Rakin, Z. He, and D. Fan, “Bit-Flip Attack: Crushing Neural Network with Progressive Bit Search,” *arXiv preprint arXiv:1903.12269*, 2019.

[27] M. Qin, C. Sun, and D. Vucinic, “Robustness of neural networks against storage media errors,” *arXiv preprint arXiv:1709.06173*, 2017.

[28] A. P. Arechiga and A. J. Michaels, “The effect of weight errors on neural networks,” in *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 190–196, IEEE, 2018.

[29] A. Thomas, “Memristor-based neural networks,” *Journal of Physics D: Applied Physics*, vol. 46, no. 9, p. 093001, 2013.

[30] M. Prezioso, F. Merrikh-Bayat, B. Hoskins, G. C. Adam, K. K. Likharev, and D. B. Strukov, “Training and operation of an integrated neuromorphic network based on metal-oxide memristors,” *Nature*, vol. 521, no. 7550, p. 61, 2015.

[31] S. Kim, M. Ishii, S. Lewis, T. Perri, M. BrightSky, W. Kim, R. Jordan, G. Burr, N. Sosa, A. Ray, *et al.*, “NVM neuromorphic core with 64k-cell (256-by-256) phase change memory synaptic array with on-chip neuron circuits for continuous in-situ learning,” in *2015 IEEE international electron devices meeting (IEDM)*, pp. 17–1, IEEE, 2015.

[32] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, “Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory,” in *ACM SIGARCH Computer Architecture News*, vol. 44, pp. 27–39, IEEE Press, 2016.

[33] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, “Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars,” *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[34] B. Rajendran and F. Alibart, “Neuromorphic computing based on emerging memory technologies,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 6, no. 2, pp. 198–211, 2016.

[35] Z. He, T. Zhang, and R. Lee, “Sensitive-Sample Fingerprinting of Deep Neural Networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4729–4737, 2019.

[36] L. T. DeCarlo, “On the meaning and use of kurtosis.,” *Psychological methods*, vol. 2, no. 3, p. 292, 1997.

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