On the Robustness of Federated Learning towards Various Attacks

by

Shrey Devenkumar Yagnik
202111072

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of

MASTER OF TECHNOLOGY
in
INFORMATION AND COMMUNICATION TECHNOLOGY
to

DHIRUBHAI AMBANI INSTITUTE OF INFORMATION AND COMMUNICATION TECHNOLOGY

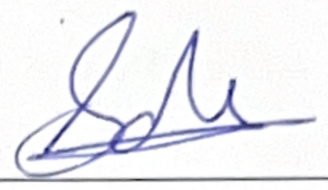
May, 2023

Declaration

I hereby declare that

i) the thesis comprises of my original work towards the degree of Master of Technology in Information and Communication Technology at Dhirubhai Ambani Institute of Information and Communication Technology and has not been submitted elsewhere for a degree,

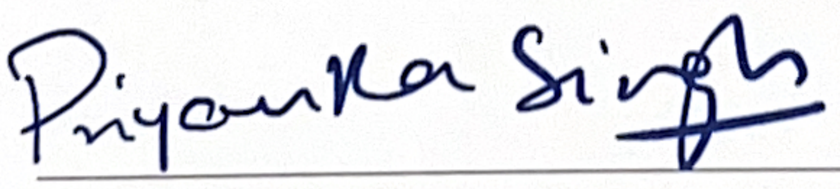
ii) due acknowledgment has been made in the text to all the reference material used.



Shrey Devenkumar Yagnik

Certificate

This is to certify that the thesis work entitled "On the Robustness of Federated Learning towards Various Attacks" has been carried out by Shrey Devenkumar Yagnik for the degree of Master of Technology in Information and Communication Technology at *Dhirubhai Ambani Institute of Information and Communication Technology* under my/our supervision.



Prof. Priyanka Singh
Thesis Supervisor

Prof. Manjunath Joshi
Thesis Co-Supervisor

Acknowledgments

The notion that continuous learning is crucial is widely acknowledged. Throughout my two-year pursuit of a Master of Technology degree, I acquired the ability to navigate complex issues that arise in the field of computer science. I am deeply grateful to the Dhirubhai Ambani Institute of Information and Communication Technology for granting me this invaluable opportunity. I am also thankful to the divine powers for the chance to have such a wonderful experience.

I want to extend my heartfelt appreciation to my mentors, Prof. Priyanka Singh and Prof. Manjunath Joshi, who has been an unwavering source of guidance and encouragement. Their profound expertise in research has been instrumental in assisting me in my work. I am incredibly fortunate to have had such exceptional mentors who demonstrated immense patience while supervising my MTech thesis.

I am also thankful to my fellow research colleague and friend Mayank Kumar for being a solid support system. He always came up with relevant solutions when I encountered a problem. Lastly, I thank my family, professors, and friends for generously dedicating their valuable time to help me become a better human. I am deeply grateful for your time and effort in assisting me in various situations. I will always cherish the knowledge and skills you have imparted and forever be thankful for your guidance.

Contents

Abstract

A study based on Federated Learning (FL), i.e., a kind of decentralized learning that consists of local training among the clients, and the central server returns the federated average. Deep learning models have been used in numerous security-critical settings since they have performed well on various tasks. Here, we study different kinds of attacks on FL. FL has become a popular distributed training method because it enables users to work with large datasets without sharing them. Once the model has been trained using data on local devices, only the updated model parameters are sent to the central server. The FL approach is distributed. Thus, someone could launch an attack to influence the model's behavior. In this work, we conducted the study for a Backdoor attack, a black-box attack where we added a few poisonous instances to check the model's behavior during test time. Also, we conducted three types of White-Box attacks, i.e., Fast Gradient Sign Method (FGSM), Carlini-Wagner (CW), and DeepFool. We conducted various experiments using the standard CIFAR10 dataset to alter the model's behavior. We used ResNet20 and DenseNet as the Deep Neural Networks. We found some adversarial samples upon which the required perturbation is added to fool the model upon giving the misclassifications. This decentralized approach to training can make it more difficult for attackers to access the training data, but it can also introduce new vulnerabilities that attackers can exploit. We found out that the expected behavior of the model could be compromised without having much difference in the training accuracy.

List of Tables

List of Figures

CHAPTER 1
Introduction

1.1 Introduction

Information processing has extensively used machine learning to assist consumers in comprehending the underlying characteristics of the data. Examples of applications include feature extraction, language processing, video analysis, and image classification and recognition [24]. The majority of artificial intelligence techniques are data-driven. A large-scale diversified dataset is required for the model to function effectively in larger deployments, which is not always available due to various factors, including legal restrictions, user discomfort, privacy concerns, competitive dynamics between different organizations over data, etc.

Due to the abovementioned issues, there has been a boost in proposing different distributed training architectures. Instead of gathering the necessary data on a centralized server, Federated Learning (FL) disperses it among various devices, such as personal computers, cell phones, and other IoT devices, and trains it locally [19]. The central server only gets the Federated Average of the model parameters of all the clients [9], [24].

Due to the clients' complete control over their private data and the ability to arbitrarily change their local model, hostile clients may employ adversarial algorithms to carry out targeted attacks. Despite eliminating the need for a centralized database, FL is still susceptible to adversarial attacks that can compromise the model's integrity and threaten data privacy. Even though FL limits the amount of data a malicious agent can access on specific devices, it can still significantly decrease the model's performance. To obtain customer information, they can be reverse-engineered.

In FL, the clients have complete control over their data, so any malicious client can deploy adversarial samples along with the original data to compromise the legit working of the model [6]. It eliminated the need for a central database, but FL is a machine-learning technique vulnerable to attacks. FL, after all, is a

machine-learning technique, so it is vulnerable to different attacks. Some studies have shown that adding poison to the original data could compromise the legit working of the model. There are different types of attacks possible on FL.

In this work, we mainly focus on the Backdoor attack and white-box attacks [15], [3], [11]. Various strategies to generate the backdoor instances to alter the model's behavior during testing have been proposed by different authors. This attack occurs during training time by injecting a pattern in the original image that acts as a backdoor. Examples of the technique include a pixel pattern strategy wherein the attack was made upon the street sign images. Given a backdoor image, the model is fooled in predicting what the street sign stands for. This work is based on a similar idea where we have generated the poisonous dataset by adding a kind of pattern to it. The main aim behind conducting this kind of attack was to check the robustness of the model in different scenarios.

As mentioned previously, we will also concentrate on White-Box attacks. A White box is a type of adversarial attack in which the attacker has full knowledge of the model architecture, parameters, hyper-parameters, gradients, and other details of the machine learning model they are targeting, whereas, in the case of a black box, the attacker has only the knowledge of the inputs and outputs and not any other internal details. In a white box attack, the attacker can modify the code, decrypt data, or extract sensitive information from the system. This form of attack is particularly dangerous since it enables the attacker to meticulously design and execute their assault without depending on speculation or trial and error. White box attacks are often launched against software programs or systems that implement cryptography, such as digital rights management (DRM) systems, secure messaging platforms, or secure payment gateways. In these scenarios, the attacker's objective is to bypass the encryption and gain access to sensitive information, such as private keys or user data.

Attackers use several techniques to launch white-box attacks, including reverse engineering, code injection, and side-channel attacks. Reverse engineering involves analyzing the software code to understand its functionality and identify vulnerabilities that can be exploited. Code injection involves modifying the code to inject malicious instructions or data that can be used to extract information or gain control over the system. To extract sensitive information, side-channel attacks target the system's physical characteristics, such as its power consumption or electromagnetic emissions [18].

The major contributions of the work are summarized as follows.

- For the black-box attacks, we studied the model's behavior in two different

scenarios: (1) Poisoning a certain percent of data from the entire dataset and (2) Poisoning a single class. We have conducted experiments on the CIFAR10 dataset using ResNet-18 as the baseline model

- We conducted three white-box attacks to check the model's behavior. We have considered different scenarios for each attack, like varying the number of adversarial samples or changing the number of clients to get the results. We have used different neural networks on different attacks. The dataset was CIFAR10 which was used in the experiment.
- In both scenarios, we show that the model performs well on normal inputs but causes misclassification when adversarial images are passed, changing the model's overall accuracy.

1.2 Organization

The work is organized as follows: Chapters 2 and 3 present preliminaries and related work, respectively. Chapter 4, 5 presents the proposed frameworks and their results respectively. We compare various primacy and deficiency in the security analysis in Chapter 6, and finally, the conclusion.

CHAPTER 2
Preliminaries

This chapter discusses potential attack scenarios: data poisoning and model poisoning attacks, training and inference time attacks consisting of white and black-box attacks [8], [20], [24],[17].

2.0.1 Federated Learning

As mentioned above, FL is a kind of decentralized learning. Individual clients have their own data to carry out the local training. So it could be exploited for potential threats, such as keeping certain clients as malicious and carrying out adversarial activity or compromising the model to get the desired results or tampering with the data so that the model generates the desired outcomes during the testing phase. It is a common practice where malicious users inject fake training data intending to corrupt the learned model. In our case, we have considered various FL scenarios that are mentioned in Chapter 5.

2.0.2 Types of Attacks

The attacks can be classified into data poisoning attacks, and model poisoning attacks. Data poisoning is a form of attack that involves manipulating the training data of a machine learning model to reduce its efficacy [19]. Data poisoning attacks can be broadly categorized into two groups: (1) Backdoor attacks involve introducing new or modifying existing training data, resulting in incorrect classifications during inference. (2) Label-flipping attacks, where an attacker alters the training data labels, can lead to incorrect model training. Attackers may use targeted or all-encompassing data poisoning techniques because they only alter one class and leave the data for other classes unaltered; targeted attacks make it more difficult to identify them.

Also, other forms of attacks could be training time attacks and inference time attacks (based on time), one-shot attacks or multi-shot attacks (based on frequency),

attacks on the federated learning model to change its behavior, and Privacy attacks which infer sensitive information about the learning system [13]. Additionally, there are two types of targeted attacks: (1) Input instance key strategy and (2) Pattern key strategy. Other potential attacks on FL are direct attacks, indirect attacks, and hybrid attacks.

Nonetheless, a model poisoning attack involves actively altering local models to decrease the dependability of global models. In contrast to data poisoning attacks, model poisoning attacks can be either untargeted or targeted [24]. Targeted attacks aim to alter the behavior of a model or a minority of samples while maintaining good overall accuracy. On the other hand, Untargeted Attacks aim to downgrade the model or break the overall accuracy of the model.

The other attacks are white-box attacks and black-box attacks [8], [20]. The attackers have all the model details they are poisoning in case of a white-box attack. In the case of a black box, that attacker has no information about the model they would be poisoning. We will mainly focus on white-box attacks. A white box attack aims to generate adversarial examples that can fool the model into misclassifying them. Adversarial examples are inputs intentionally designed to look similar to legitimate inputs but are perturbed so that the model makes a wrong prediction. The attacker generates these examples by manipulating the model's inputs to exploit its vulnerabilities, such as its sensitivity to small changes or reliance on certain features. We have studied three types of White-box attacks in the paper.

CHAPTER 3
Related Work

The literature has suggested several targeted assaults to compromise the normal working of the model. As long as they successfully carry out the assault, they are considered effective.

Gu et al. explained BadNets, i.e., A Backdoor Neural Network, which showed the experiments conducted using a backdoor attack [5]. They conducted the targeted and untargeted attacks using a pixel pattern as the backdoor trigger. Saha et al. suggested a backdoor trigger-based approach where the attacker can display the trigger at any time on any hidden image. [14]. Even during model training, the trigger is kept a secret. Shafahi et al. performed a clean label attack [16]. They performed a targeted attack in which a poison of certain percent opacity was added to the image to get the targeted misclassification during test time. They had given a source and a target, and the main aim was to bring the model's probability to predict the target as the base to be higher.

Zhou et al. showed that a hostile client could alter the model update to carry out a model-poisoning attack in federated learning because the clients have complete control over local data, and the training process is carried out locally [24]. They also showed that the poisoned data is created by a generator and updated by a reward function for loss before being sent to a discriminator. Chen et al. proposed a form of attack called targeted backdoor attack wherein a kind of accessory or random poisoned pattern or both together are placed on a person's image and carried out the attacks on a facial recognition system [3]. They also trained the model using blended injection and blended accessory injection strategy that gave significant results.

They also proposed several backdoor attacks that produced the intended results at the test time. They were working on a facial recognition system. They proposed multiple poisoning techniques at work. They used a strategy called blended injection that adds a kind of random pattern in the image. The pattern could be a cartoon image or gaussian noise. Another strategy they proposed was

adding an accessory on top of an image. Hence, the model recognizes the target person as authorized and could gain access to the system.

Yang et al. proposed that with a causative attack, an attacker can add, change, or remove any number of input data points from the model's input dataset at will [21]. Here, it is assumed that the attacker already knows the details of the original model, but he would not directly be able to poison the model. Still, he may poison the training data to compromise the model.

The Backdoor attack adds some triggers to the original data. The trigger could be in the form of any random pattern or an accessory. A backdoor attack is successful if it can perform the original task well and introduce a new one without compromising performance.[5]. These attacks are difficult to spot since they typically have little impact on how the original activity is performed.

Zhao et al. demonstrated a backdoor video attack, which establishes a strong basis for enhancing the robustness of video models and offers a new viewpoint on understanding more successful backdoor attacks [23]. The paper's authors illustrated how the suggested backdoor method could efficiently modify current video models using only minimal training data. Zhang et al. proposed that with just a straightforward one-line modification, the backdoor assault known as Neurotoxin targets parameters that are modified less dramatically during training and also targets persistent backdoors installed on the FL framework. [22]. Because the attacks persisted, the backdoor remained in the model even after the attackers ceased uploading the poisoned data.

Nguyen et al. carried out a backdoor attack called WANET, i.e., an imperceptible warping-based backdoor attack [12]. The suggested backdoor approach demonstrates superior performance compared to previous methods in a human evaluation test by a significant margin, showing its stealthiness as it is undetectable even by the machine to detect the backdoor.

Kurakin et al. proposed different methods to generate adversarial samples to be added along with the original data [6]. Moreover, they generated the adversarial samples using FGSM [4] and checked for the robustness of the neural networks. Their experiments showed that the model used was robust enough, so the change in the final accuracy was negligible.

Goodfellow et al. carried out a white-box attack known as Fast Gradient Sign Method (FGSM) that exploits the gradients of a neural network to build an adversarial image [4]. They showed that FGSM computes the gradients of a loss function concerning the input image and then uses the sign of the gradients to create a new image. Carlini et al. proposed an attack called Carlini-Wagner (CW)

[1]. They proved that the defensive distillation could not increase the robustness of the model when the attack was conducted by generating the adversarial samples.

Moosavi et al. performed a DeepFool attack which is a precise and straightforward technique for determining the robustness of various classifiers to adversarial perturbations [10]. They proposed that the DeepFool algorithm effectively computes perturbations that deceive deep neural networks using an iterative algorithm to find the minimum perturbation that results in a misclassification.

CHAPTER 4
Proposed Method

This chapter gives the details about the methods that were proposed for experimenting. It discusses the white and black-box attacks.

4.1 Threat Model

This section represents the details of the participating entities. Mainly, two entities are participating in this model: normal users and adversaries. The communication channel between the user and the CSP is considered an insecure channel. We have taken several adversaries to check the model's behavior for experimental purposes.

As we are using the image dataset, we add a small perturbation on the images. A perturbation vector refers to a small change that is made to an input in order to affect the output of a model. For example, in adversarial attacks, perturbation vectors are added to inputs in order to cause the model to misclassify them.

- **User**: A normal client participating in the Federated process. Here, a certain number of clients are assumed to be honest, and they will carry out a normal training process and outputs legit results on regular images.
- **Adversary**: A malicious client adds the perturbations to the original image and then forwards the image to the neural network. Here, we also took multiple numbers of malicious clients in our experiments to check the results. The malicious participants aim to break down the actual working of the model in the best possible way.

4.2 Black-Box Attacks

In this case, the attacker treats the model as a black box and only has access to its input and output. Here, we examined one such attack known as Backdoor Attack.

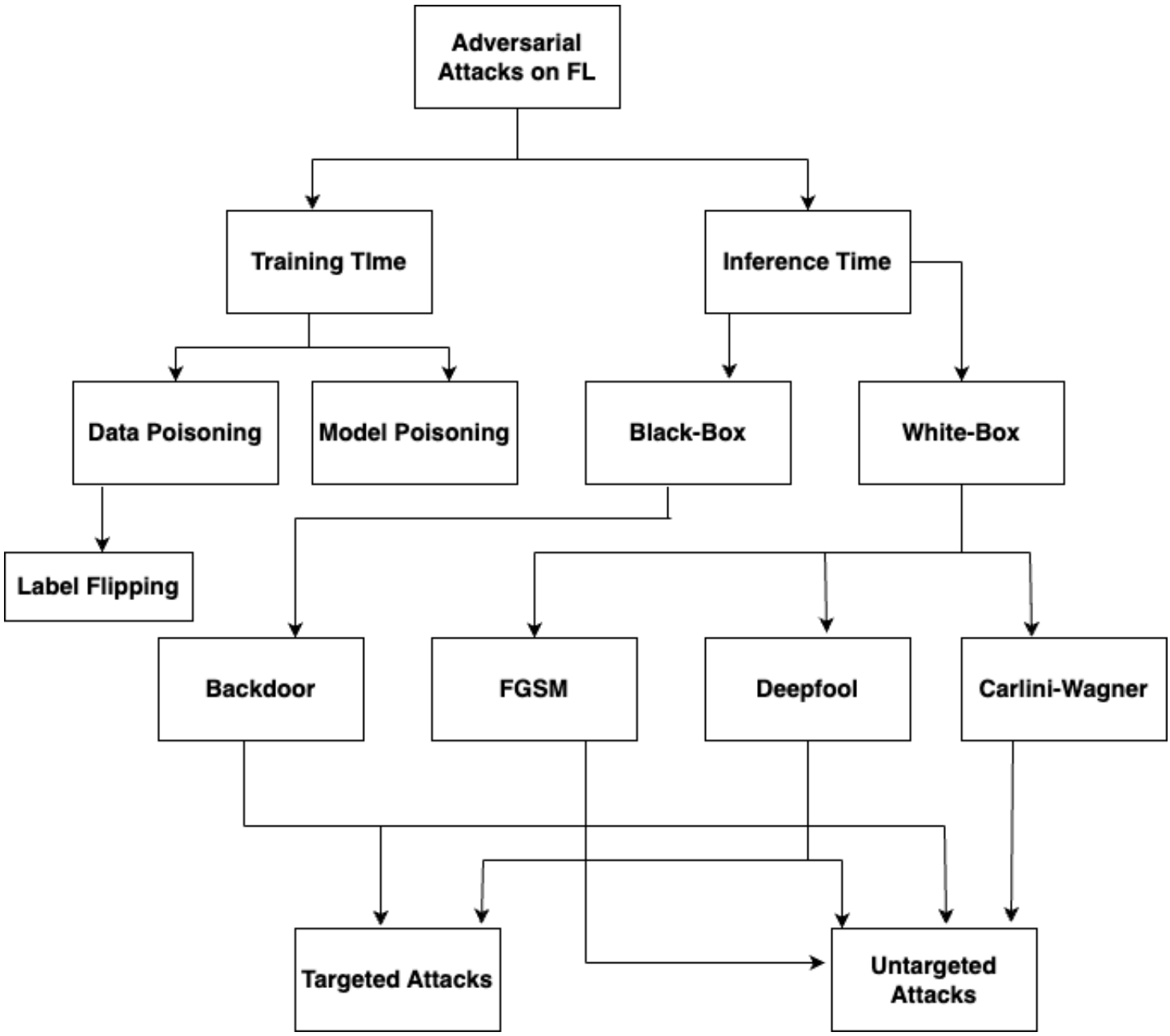


Figure 4.1: Potential Attacks on FL

4.2.1 Backdoor Attack

A backdoor is a hidden entry point in a computer system or network that bypasses normal security measures, allowing unauthorized access. Developers can intentionally create backdoors for debugging and maintenance purposes, but attackers can also add them to gain access to a system for malicious purposes.

Backdoor attacks are a cyberattack where an attacker exploits a backdoor in a system to gain unauthorized access or control. These attacks can occur in various forms, including malware, rootkits, and remote access trojans.

There are several methods that attackers can use to create a backdoor in a system. One common method is to exploit vulnerabilities in the software or operating system running on the system. This can be done by exploiting a flaw in the code or configuration of the system, allowing the attacker to gain access to the system remotely.

Another method that attackers can use is to use social engineering tactics to trick users into installing malware that creates a backdoor on their system. For example, an attacker may send an email with a link or attachment that, when clicked or opened, installs a backdoor on the user's system.

It can also typically be defined as gaining unauthorized access to an operating system that can be used for malicious purposes. The aim behind conducting the attack is to alter the normal working of the system and carry out an intended goal. Here, the attacker injects poisoned samples into the training data to satisfy his malicious intent [3].

Backdoor triggers are defined as noise added to the original image, which is later passed through the model. There are a few corresponding terminologies related to it.

- **Source image**: It is an image from the dataset that acts as the source and upon which the poisons should be added. The model will then be trained upon the triggered image.

- **Backdoor trigger**: The backdoor trigger is the pixelated pattern added to the source image, which is further passed to the classifier for misclassifications.

This trigger can be a small, imperceptible change to the input image that causes the model to misclassify. As an illustration, an attacker could implant a trigger pattern into a limited portion of the training data to train a model to associate the trigger pattern with a particular class label.

During testing, the attacker can inject the same trigger pattern into an input image that does not belong to the targeted class, causing the model to misclassify.

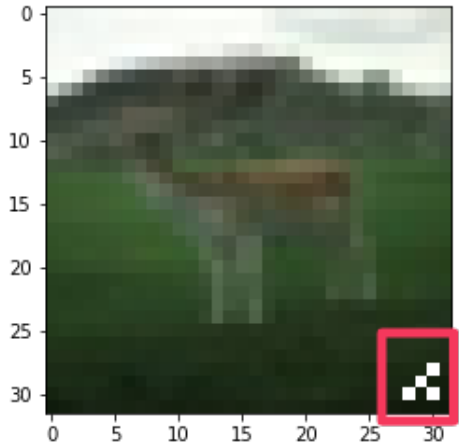


Figure 4.2: The backdoor trigger

The attack is typically successful because the trigger is designed to be small and indistinguishable from other patterns in the image, making it difficult to detect.

A random backdoor attack can degrade the overall accuracy of a machine-learning model by introducing a hidden vulnerability that allows an attacker to control the model's behavior. Additionally, even if the attacker doesn't know the exact trigger, he can still use the backdoor by injecting a similar image, and hence, the model will produce the desired output. This makes it difficult to detect the backdoors, degrading the model's overall accuracy. For example, consider a pixelated pattern inserted into an image classification model as a backdoor. In that case, it might cause the model to misclassify any image containing the pattern as a specific target class or a random class.

4.3 White-Box Attacks

Here, we study three types of white-box attacks by adding the perturbations on the images.

4.3.1 Fast Gradient Sign Method

The Fast Gradient Sign Method (FGSM) is an adversarial attack that aims to fool a machine learning model by adding small, carefully crafted perturbations to the input data [4]. The fundamental concept of FGSM is to compute the gradient of the loss function with respect to the input data and utilize this gradient to produce a distorted input that optimizes the loss function [1]. The perturbed input is crafted by adding a small amount of noise to the original input in the direction of

the gradient. The adversarial sample is shown in Fig. 4.3.

Consider a trained neural network with input data x and $f(x)$ output class probabilities. The attacker aims to craft a perturbed input adv_x that looks similar to x but is misclassified by the neural network. The FGSM attack achieves this by computing the gradient of the loss function $J(f(x), y)$ concerning the input x, where y is the true class label of x. Then, the attacker adds a small perturbation ϵ times the sign of the gradient to the input:

$adv_x = x + \epsilon * sign\ (\nabla_x\ J(f(x), y)$

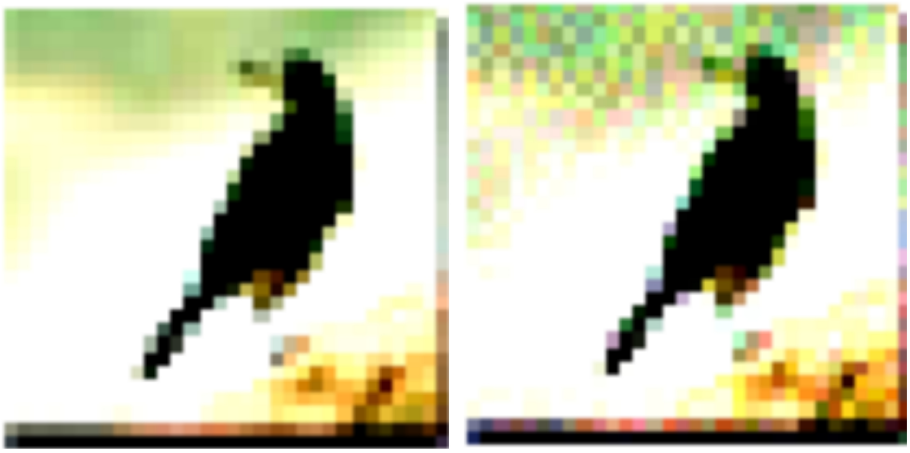


Figure 4.3: Adv samples with $\epsilon = 0.1$.

Here, ϵ is a small scalar value that controls the strength of the perturbation. The sign function ensures that the perturbation is added in a way that maximizes the loss function. By carefully choosing the value of ϵ, the attacker can control the magnitude of distortion introduced to the input. If ϵ is too small, the perturbation may not be noticeable, and the attack may not succeed. If ϵ is too large, the perturbation may be too obvious, and the defender may detect the attack [1].

Here, we create and distribute the adversarial samples among multiple malicious clients, as in decentralized learning. We have generated the results with different scenarios.

The fundamental concept of FGSM is to compute the gradient of the loss function relative to the input data and use this gradient to create a distorted input that maximizes the loss function. The perturbed input is crafted by adding a small amount of noise to the original input in the direction of the gradient.

As an instance, consider a neural network model that is designed to identify images of dogs and cats. We can use the FGSM attack to generate an adversarial example that causes the model to misclassify a cat as a dog. To do this, we first select a cat image as the original input, and then calculate the gradient of the loss

function with respect to the input. We can then add a small amount of noise to the input in the direction of the gradient, which causes the model to misclassify the image as a dog.

The goal of the attack is to create adversarial examples that are similar to the original inputs but induce the model to produce inaccurate predictions. By maximizing the loss function, the attacker can generate adversarial examples that have the most significant impact on the model's output, while still being relatively close to the original input data.

The FGSM technique maximizes the loss function by computing the gradient of the loss function relative to the input data, followed by obtaining the sign of the gradient. This sign is then multiplied by a small epsilon value to control the magnitude of the perturbation. The resulting perturbation is then added to the input data to create the adversarial example. FGSM can generate adversarial examples that are purposely challenging for the model to accurately classify by optimizing the loss function. This is because the loss function measures the difference between the predicted output and the true output for a given input. By maximizing the loss function, the attacker has the ability to generate inputs that are probable to be misclassified by the model, while still appearing similar to the original input.

4.3.2 DeepFool

The DeepFool attack is a type of adversarial attack that is designed to fool deep neural networks [10], [1]. Unlike the FGSM attack, which adds a small perturbation to the input data in the gradient direction, the DeepFool attack uses an iterative algorithm to find the smallest perturbation that causes a misclassification.

More specifically, let's say we have a trained neural network with input data x and output class probabilities $f(x)$. The attacker aims to find the smallest perturbation δ that causes the model to misclassify the input x. The DeepFool attack achieves this by iteratively finding the direction of the decision boundary w.r.t. the input x and then moving the input along this direction until it crosses the decision boundary.

Suppose we have a deep neural network with input data x and output class probabilities $f(x)$. The adversaries' goal is to find the smallest possible perturbation δ that can cause misclassification of the input x. The DeepFool attack works by iteratively finding the direction of the decision boundary w.r.t. the input x and then moving the input along this direction until it crosses the decision boundary. Here are the steps of the algorithm:

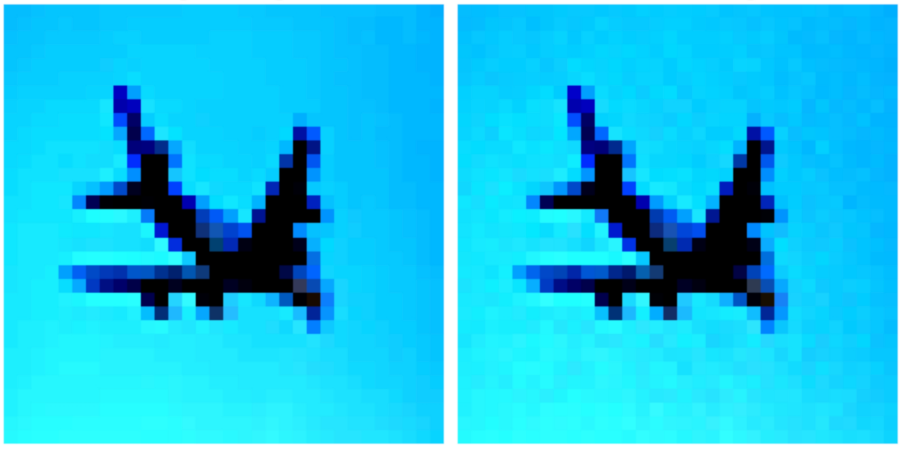


Figure 4.4: Perturbed sample using DeepFool.

1. Initialize the perturbation δ to zero: $\delta = 0$.

2. Compute the gradient of the decision boundary w.r.t. the input x, denoted by $\nabla w_i\, f(x)$, where w_i is the i^{th}output class weight of the network.

3. Compute the direction of the decision boundary closest to the current input x, denoted by d_i. This direction is computed as:

   d_i = -$\nabla w_i f(x)$ / $||\nabla w_i f(x)||_2$ where $_2$ denotes the L_2 norm [1] of the gradient vector.

4. Compute the minimum perturbation required to move the input x across the decision boundary from its current predicted class i to another class j. This is given by:

   δ_i = -$f_i(x)$ / $||\nabla w_i f(x)||_2$ * $\nabla w_i f(x)$ + $f_j(x)$ / $||\nabla w_j f(x)||_2$ * $\nabla w_j f(x)$

   where $f(x)_i$ and $f_j(x)$ denote the predicted probabilities of classes i and j for the input x, respectively. The perturbed example is shown in Fig.

5. Update the perturbation by adding the computed minimum perturbation to the current perturbation: δ=δ+δ_i

6. Repeat steps 2-5 until the input x is misclassified, i.e., $f(x+\delta) \neq f(x)$.

The key idea behind the DeepFool algorithm is to compute the smallest possible perturbation that can move the input x across the decision boundary from its current predicted class i to another class j. This is done by finding the direction of the decision boundary closest to the current input and then computing the minimum perturbation required to cross the boundary. The algorithm then updates

the perturbation by adding the minimum perturbation to the current perturbation and repeats the process until a misclassification is achieved.

The given sentences describe two steps to compute the direction of the decision boundary closest to the current input x and the minimum perturbation required to move x across the decision boundary from its current predicted class i to another class j.

Compute the decision boundary closest to x.

- In order to find the direction d_i of the decision boundary closest to x, we first compute the gradient vector of the function f at x, denoted by $\nabla w_i f(x)$. We then normalize this gradient vector by taking the L_2 norm, denoted by $||\nabla w_j f(x)||_2$, which gives us a unit vector pointing in the direction of the steepest ascent of the function at x. To get the direction of the decision boundary closest to x, we take the negative of this unit vector, and so we get: d_i = -$\nabla w_i f(x)$ / $||\nabla w_i f(x)||_2$ This means that the direction d_i is the direction of the steepest descent of the function f at x, and it is also the direction in which x needs to be perturbed to cross the decision boundary.

Compute the minimum perturbation required to move x across the decision boundary from i to j:

- To find the minimum perturbation required to move x from its current predicted class i to another class j, we first compute the predicted probabilities of x for classes i and j, denoted by $f_i(x)$ and $f_j(x)$ respectively. We then compute the gradient vectors of the function f at x for classes i and j, denoted by $\nabla w_i f(x)$ and $\nabla w_j f(x)$ respectively. We then normalize these gradient vectors by taking the L_2 norm, denoted by $||\nabla w_i f(x)||_2$ and $||\nabla w_j f(x)||_2$ respectively, which gives us unit vectors pointing in the direction of the steepest ascent of the function at x for classes i and j.

To compute the minimum perturbation required to move the input x across the decision boundary from its current predicted class i to another class j. This is given by:

δ_i = -$f_i(x)$ / $||\nabla w_i f(x)||_2$ * $\nabla w_i f(x)$ + $f_j(x)$ / $||\nabla w_j f(x)||_2$ * $\nabla w_j f(x)$

where $f(x)_i$ and $f_j(x)$ denote the predicted probabilities of classes i and j for the input x, respectively. The perturbed example is shown in Fig.

4.3.3 Carlini-Wagner

The Carlini-Wagner (CW) attack is an adversarial attack that aims to generate small perturbations to an input sample that can cause a deep neural network to

misclassify the sample [1]. The goal of the CW attack is to find the smallest possible perturbation vector that, when added to the input sample, causes the neural network's output to be classified as a different class than the accurate class of the input sample.

The CW attack is based on the idea of minimizing a differentiable surrogate objective function that approximates the non-differentiable constraint of the attack. The objective function combines a perturbation term and a confidence term. The perturbation term measures the magnitude of the perturbation vector, while the confidence term measures the attacker's confidence in the generated adversarial example being misclassified.

The optimization problem can be formulated as follows:

$minimize\ ||\delta||_2 + c * f(x) + \delta$

Where δ is the perturbation vector, c is a hyperparameter that controls the balance between the magnitude of the perturbation and the confidence level of the attack, f is the confidence term, x is the original input sample.

Figure 4.5: Perturbed sample using Carlini-Wagner.

The confidence term f can be defined in various ways, such as the cross-entropy loss or the margin loss. The cross-entropy loss measures the difference between the predicted class probabilities and the ground truth probabilities. In contrast, the margin loss measures the difference between the predicted score of the true class and the highest score among the other classes.

The optimization problem is typically solved using an iterative algorithm such as gradient descent or L-BFGS. In each iteration, the gradient of the objective function concerning the perturbation vector δ is computed and used to update the perturbation vector. The gradient is computed using the chain rule and back-propagation.

One of the key features of the CW attack is that it is model-agnostic, meaning that it can be applied to any deep neural network classifier regardless of its architecture. The attack can also be extended to handle various defense mechanisms such as randomization, gradient masking, and adversarial training.

However, the CW attack can be computationally expensive and may require many iterations to converge. Additionally, the attack may not be effective against some deep neural networks already robust to adversarial attacks. Therefore, it is important to evaluate the effectiveness of the CW attack on the target deep neural network before using it in practice.

The magnitude of a perturbation vector is a measure of its size or intensity. It is typically measured in terms of a distance metric, such as the Euclidean distance, and represents how much the perturbation vector has changed the input.

For example, suppose we have an input vector x and add a perturbation vector p to it. The resulting perturbed input vector is $x+p$. The magnitude of the perturbation vector is then calculated as the distance between x and $x+p$.

The magnitude of the perturbation vector is important because it determines how noticeable the perturbation is to a human observer. If the magnitude of the perturbation vector is small, then the perturbed input will bear a striking resemblance to the original input and could be indistinguishable to a human observer. On the other hand, if the magnitude of the perturbation vector is large, the perturbed input may be noticeably different from the original input.

Therefore, in machine learning, the term "perturbation term" refers to the magnitude of the perturbation vector, which is used to measure the extent to which the input has been modified in order to achieve a certain outcome, such as causing the model to misclassify the input.

CHAPTER 5
Experiments and Results

This chapter presents the details of our experiments to validate the proposed frameworks.

5.1 Dataset and Experimental Setup

The CIFAR10 dataset was utilized in the attack on image classification models. The dataset contains 50,000 (32x32) training examples, including 5000 samples for each of the ten classes (airplane, car, bird, cat, deer, dog, frog, horse, ship, and truck). Additionally, there are 10,000 test samples in it. The perturbations were applied to these images and tested after that.

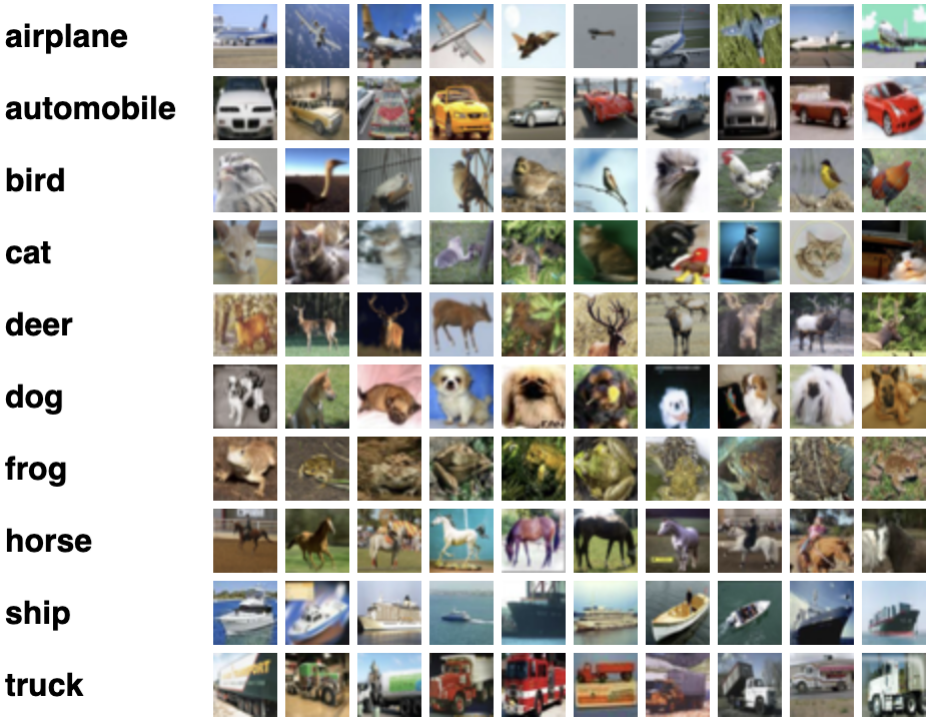


Figure 5.1: The CIFAR10 Dataset.

We took ResNet18, ResNet20, and DenseNet as the Deep Neural Networks for the experiments. All the experiments were conducted in a Federated Learning Scenario where multiple malicious clients participated. We increased and decreased the value of different parameters, including the number of malicious clients, and varied the intensity of the attack. The experiments were run for their specific number of iterations, and we received the results.

5.2 Backdoor-Attack

We conducted various experiments on poisonous images in the FL environment. Here we generated different percentages of poisons upon the images and checked the respective accuracies. All the experiments were run for two-hundred iterations; ten clients were considered to be participating in the process, and five were malicious clients. We also varied the number of malicious clients participating in the process. The results are shown in the figures with their specific scenarios.

5.2.1 Experiment 1

In the first scenario, we poison the entire dataset percent-wise and check the accuracy. We have taken different poison variations and gradually increased the amount of poison. Fig. 5.2 shows how the model behaves in different scenarios. Here, the main purpose was to degrade the accuracy or increase the attack success rate of the model as it is an untargeted attack. It is evident from the figure that the accuracy decreases. We varied the poisonous samples like 0%, 5%, 10%, 15%, 20%, and 30%. After each iteration, we got the accuracies of all the respective classes, and the aggregated result is shown in the figure.

5.2.2 Experiment 2

In the second scenario, we took a specific class and performed a backdoor attack instead of poisoning the entire dataset. We experimented upon the class 'deer'. Fig. 5.3 shows that if we poison a single class, it gives us more accuracy than the first scenario. Still, in this case, breaking the model accuracy is beneficial in terms of the attack's success. Here, we increased the poisonous samples as done in the first experiment but just poisoned a single class.

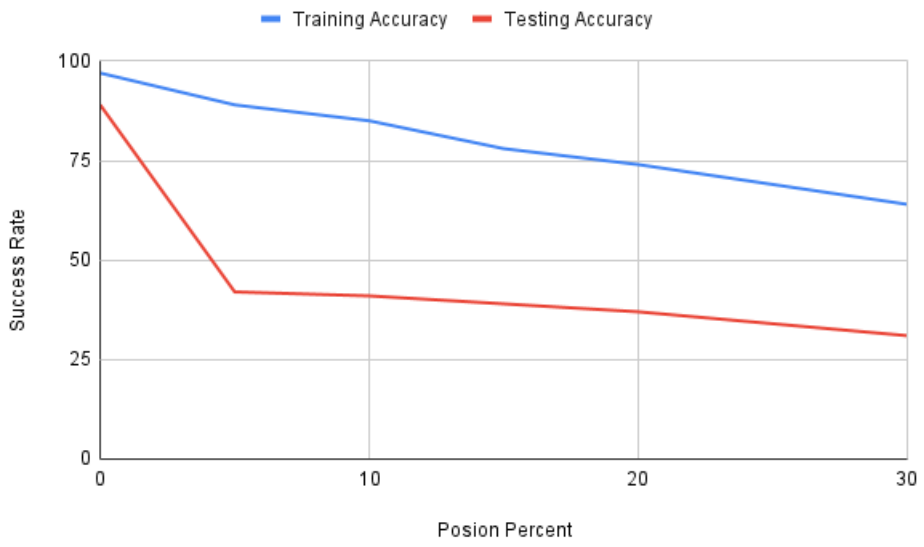


Figure 5.2: Poisoning all the classes

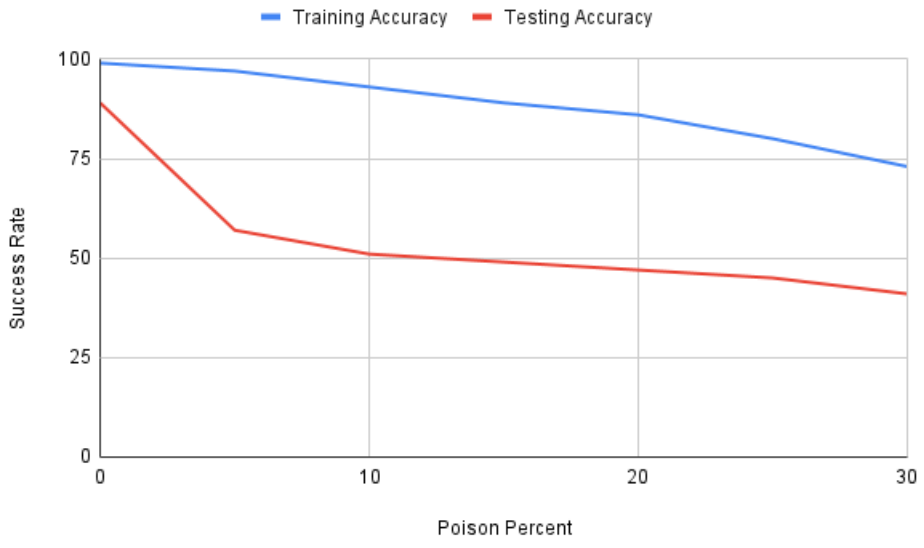


Figure 5.3: Poisoning a single class

5.2.3 Experiment 3

In the third experiment, we evaluated the system's precision by analyzing the difference in results when the poisons were present versus when they were absent. While causing the expected misclassification, the final accuracy of the model differs a lot because the attack conducted was untargeted. As shown in Fig. 5.5 the results stay almost the same in the case of training accuracy but differ in testing accuracy. Considering the first experiment, the drop in the training accuracy is just 8%, whereas in the second case, the drop is negligible, but the testing accuracy differs a lot. We got these results because in the second case we considered a single class to be poisoned.

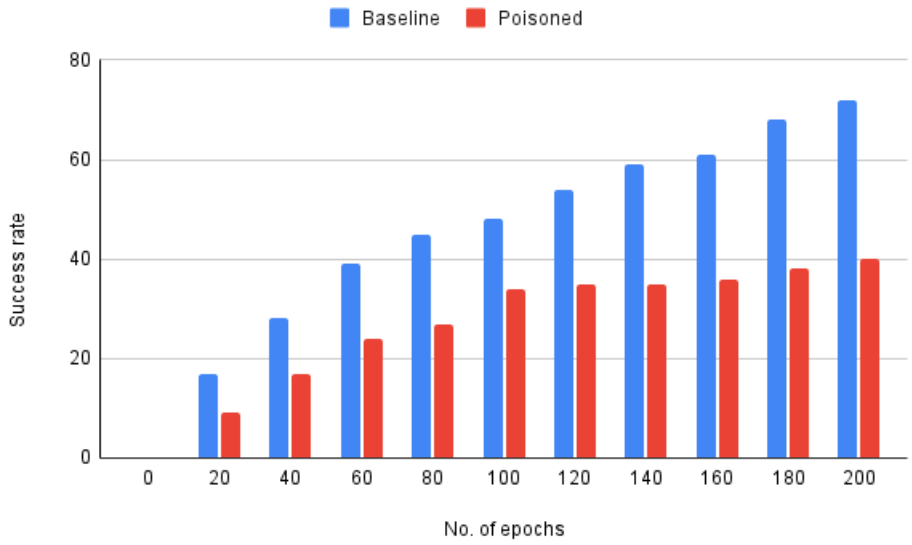


Figure 5.4: Comparison of baseline with actual accuracy

5.3 Fast Gradient Sign Method

5.3.1 Experiment 1

In this experiment, we took twenty clients to participate in the adversarial attack process; out of them, we kept five clients to be malicious. Here, as mentioned above, about the range of epsilon, we took different ranges of it to check the results. Fig. 5.5 shows how the model behaves when we gradually increase the value of epsilon. Here we did not consider the epsilon value greater than 0.2 because the perturbations were visible. As shown in Fig. 5.5, the test accuracy

drops gradually as the value increases. Keeping the value of epsilon to be 0.07, we can also drop the accuracy without the mode noticing the perturbations. The entire experiment was run for two-hundred epochs using ResNet20, and the results were generated.

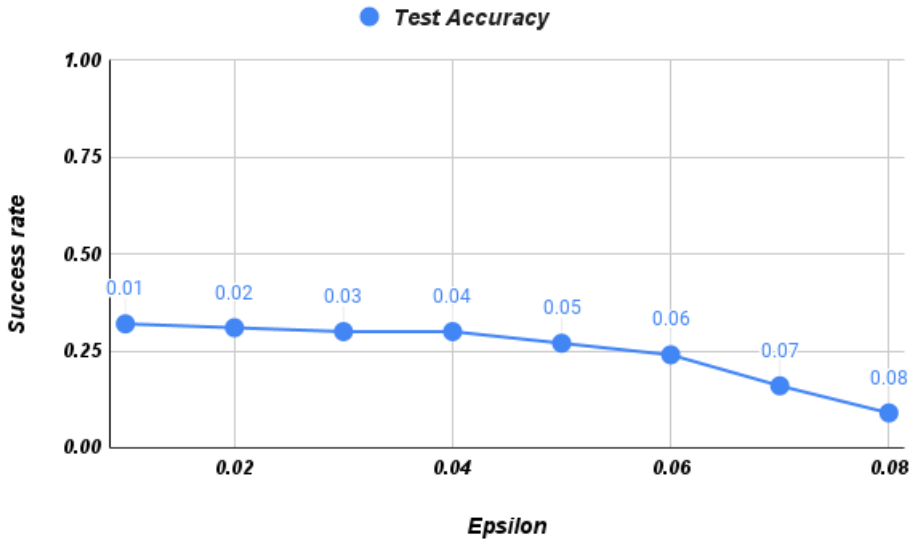


Figure 5.5: Epsilon vs. Test accuracy

5.3.2 Experiment 2

In this experiment, we maintained the epsilon value at a constant of 0.07 and amplified the number of malicious clients. The findings of this experiment are presented in Fig. 6.1. As shown in the figure, the test accuracy drops as we increase the number of malicious clients. At first, we took five clients and then increased it to ten, fifteen, and twenty. This experiment was also run for two-hundred epochs. The results of this experiment suggest that by keeping the number of clients to be lesser, i.e., around ten, the attack is successful from the attackers' point of view.

5.4 DeepFool

To conduct the experiments on DeepFool, we used ResNet20 and trained the model with it for 100 epochs after finding the minimum perturbation and adding it to the images of the CIFAR10 dataset.

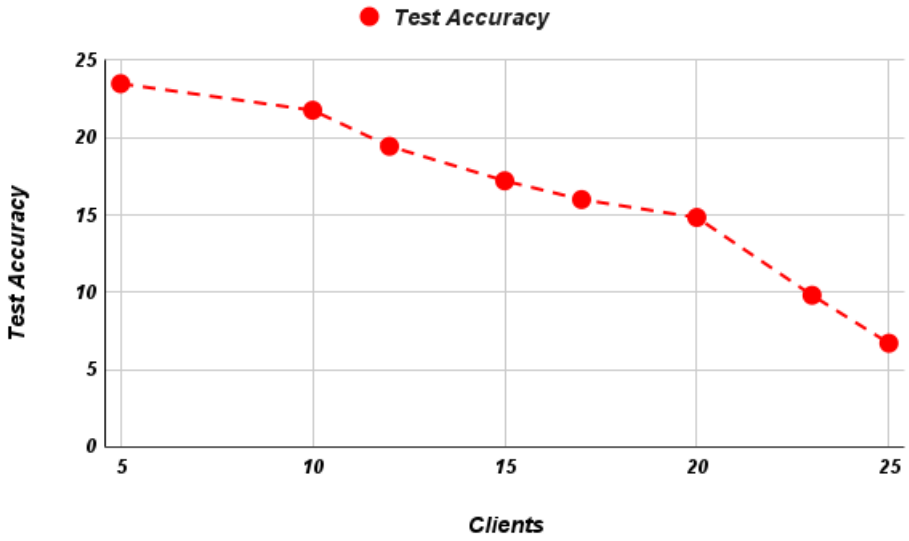


Figure 5.6: Clients vs. Success rate

5.4.1 Experiment 1

The DeepFool attack aims to find and add the minimum perturbation to the image. The strength or effect of the perturbation is controlled by two parameters, i.e., *max_iter* and *overshoot*. As we keep increasing the value of *max_iter*, the accuracy of the attack begins to drop.

max_iter refers to the maximum iterations allowed for the algorithm to find the minimal perturbation required to fool the targeted deep neural network. For the lesser values of *max_iter*, the algorithm may terminate prematurely without finding a successful perturbation, resulting in a failed attack.

Fig. 5.7 shows the trends of how the model behaves. We took five malicious clients to experiment. We took different values of *max_iter* and observed that as the value of *max_iter* reaches four, then the test accuracy of the model drops to zero, which is the best case. For the value of *max_iter* as three, our test accuracy was just around 13%. The image in Fig. 4.4 shows the minimum perturbation.

5.4.2 Experiment 2

To understand different scenarios, we also took multiple malicious clients to watch the model's behavior. Fig 5.8 shows the results. At first, we took a single client to experiment. Gradually, we increased the number of clients and simultaneously increased-decreased the value of *max_iter*. As shown in Fig 5.8, we ran the same experiment as above but increased the number of malicious clients in this case.

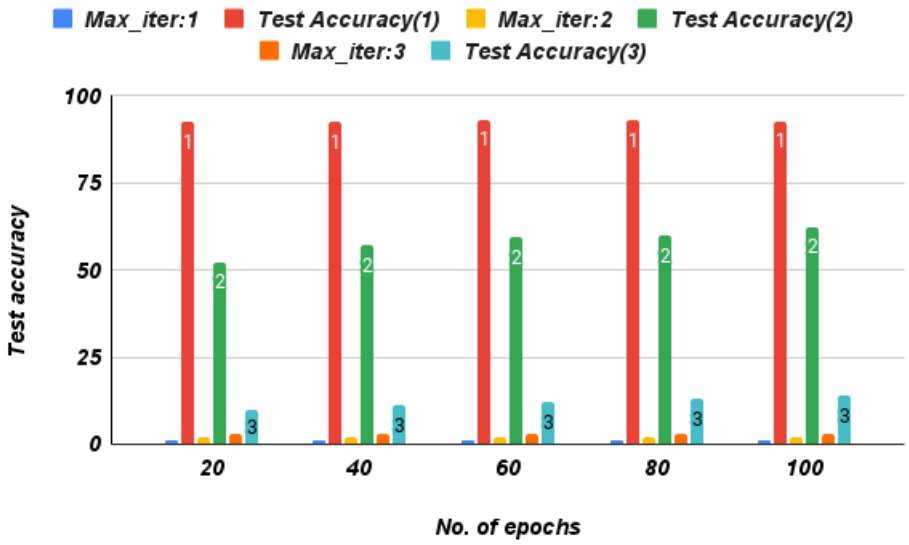


Figure 5.7: Strength of the attack

The results showed that with the increase in the number of clients, for lesser values of *max_iter*, the test accuracy increased, which made the attack more difficult. With the increase in the values of *max_iter*, the accuracy finally dropped to zero.

Based on the experiments, we can conclude that the number of malicious clients must be less for the attack to succeed. The results shown in Fig. 5.8 are the average of the values of *max_iter* as one and two. For values greater than two, the accuracy drops and eventually reaches zero.

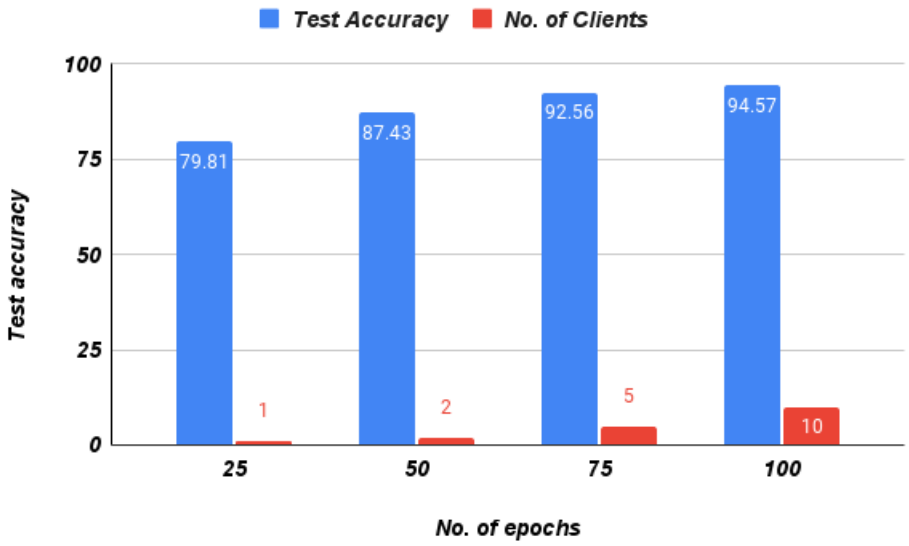


Figure 5.8: Clients vs. Test accuracy

5.5 Carlini-Wagner

In the case of CW, we experimented using two different neural networks, i.e., ResNet20 and DenseNet. The main aim behind conducting this attack is to break down the model's accuracy. This way, we can say that the attack is successful. We have used five malicious clients for both the DNN. When we trained the model using Densenet, the training accuracies we received were almost similar to the baseline, around 96%. Regarding testing accuracies, the highest was around 53%. But as indicated above, the aim was to break down the normal working of the model.

In the case of ResNet, we got the best model breakdown, around 16%. The results are shown in Fig. 5.9.

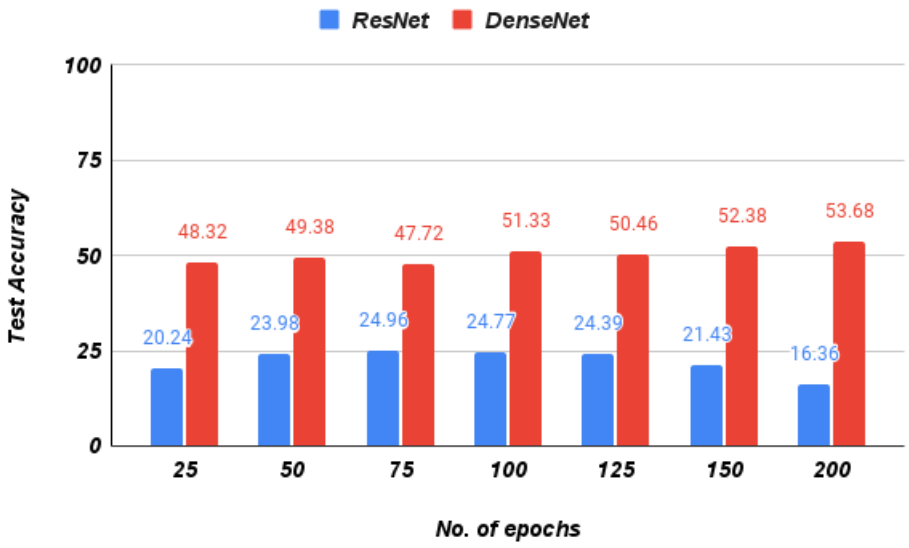


Figure 5.9: ResNet20 vs. DenseNet

CHAPTER 6
Analysis of the techniques

Here, we make some kind of comparisons between the attack techniques.

6.1 Black-Box Attack Analysis

We have conducted the experiments in Federated Learning Environment. Almost a similar work has been done in [5]. The authors have conducted different experiments using MNIST dataset that comprised targeted and untargeted attacks. We used ResNet as the training model, whereas they used a sequential one. Also, we have implemented the code on FL. They have also conducted a different targeted backdoor experiment using a Sweedish Backdoor dataset.

Further, they poisoned the entire dataset and showed that even if the backdoored images represent only the 10% of the training dataset. In contrast, we conducted different scenarios in our case, like poisoning the entire dataset and just a single class.

Let x_0 be the original image. We are adding a noise δ to the original image. So now the poisoned image would be: $x_0+\delta=x^{'}$. The value of δ ranges from [0-255]. Here we have normalized the set, and the max value of a pixel is 1.

δ is a matrix of pixels that we should add to our original image, which could be placed anywhere in the image. Here we have placed it at the bottom right of the image.

Here, as the percentage of poisons increases, the accuracy drops, so the increase in poisons is indirectly proportional to a significant accuracy.

Here, we say the attack is successful if it has high accuracy on regular inputs but a degraded accuracy on backdoor instances. As shown in Fig. 6.1, the model's accuracy drops in both cases as the strength of the poison increases. The traditional approach [5] shows an absolute accuracy of 45.1% on backdoored test instances, whereas the proposed scheme shows a final accuracy of 37% for all the classes.

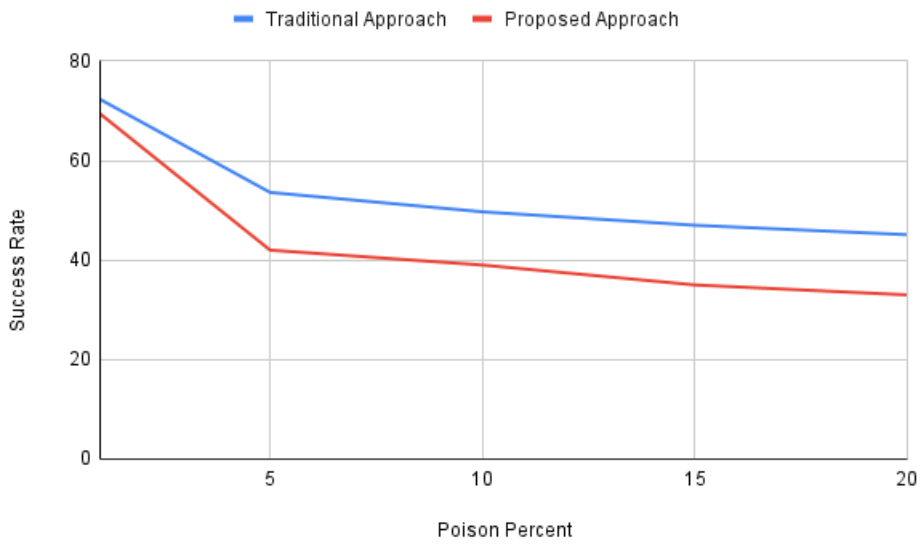


Figure 6.1: Comparison of Traditional accuracy with Actual accuracy

Li et al. proposed similar work wherein they conducted a Few-Shot Backdoor Attack (FSBA). The dataset used in the paper was Visual Object Tracking (VTO)[7]. Our work shows almost similar kind of results. The paper gradually increased the amount of poisonous backdoor frames [0%, 5%, 10%, 15%, 20%]. This can lead to a significant reduction in model accuracy, as the model will produce incorrect output for inputs that are similar to the trigger. They used the objects in videos taken from an iPad. The model's capacity to track an object decreases with the increase in the number of Few-Shot backdoor samples depicting the stealthiness of the attack. The difference in our work lies in the framework, i.e., we have generated an FL scenario to conduct the attacks and got satisfying results.

They used three different models: SiamFC, SiamRPM++, and SiamFC++, and the datasets used were GOT10K and OTB100. As mentioned earlier, the model used in our case was ResNet-18, and we conducted the experiments on CIFAR-10 Dataset. The comparative analysis is shown in Table 6.1.

6.2 White-Box Attack Analysis

We compared our results with state-of-the-art works from different papers. In the case of FGSM, we took our base paper as [4]. The technique suggested in the paper was using a fast gradient to generate the adversarial samples, but not with federated learning. In our case, we carried out the same work in a decentralized manner, i.e., using multiple clients, and our results were significant. Hence, we

Table 6.1: Comparative Analysis

	Analysis of Different Techniques		
	BadNet [5]	***FSBA [7]***	***Proposed Approach***
Frameworks	Keras [5]	Visual object tracking (VOT)	PyTorch
Models	Linear	SiamFC, SiamRPM++ and SiamFC++	ResNet
Dataset	MNIST and Sweedish BadNet	GOT10K and OTB100	CIFAR-10
Accuracy(train)	98%	SiamFC(54.4 %), SiamRPN++(51.5%), SiamFC++(61.51%)	96%
Accuracy(test)	40.05%	SiamFC(6.49 %), SiamRPN++(6.79%), SiamFC++(10.65%)	31%

proved that even if FGSM is used decentralized, it has the same impact on the model, and the attack succeeds.

For the CW attack, we took [1] as our base paper and compared the results. For a CW attack to succeed, we needed to break down the final accuracy of the model to a good extent. The traditional method used two datasets, i.e., MNIST and CIFAR10, and multiple neural networks to generate the results. Their work indicated that they could break down the model's normal working. In our paper, we have considered a federated learning scenario where we have used multiple malicious clients to perform the attack, and the results showed that we could break down the validation accuracy of the model.

For the DeepFool attack, we took [10] as our base paper. The central idea behind DeepFool attack is to create as minimum perturbations as possible upon an image. As shown in Fig 4.4, the original image and the adversarial image are not distinguishable, showing the same results as given in [10]. The state-of-the-art work mainly showed details about how the attack is conducted using different datasets and Neural Networks. In our case, as it is federated learning, we took multiple malicious clients to be participating in the process and generated the results. We needed to break down the model's accuracy for the attack to succeed, and we could do it. The final validation accuracy of the model was utterly 0%, which is very significant. The comparison of all the attacks is shown in Table **??**.

Table 6.2: Comparasion among the Techniques

	Analysis of Different Techniques		
	FGSM [4]	***DeepFool [10]***	***CW [1]***
Type	White-Box [2]	White-Box	White-Box
Attack Target	Untargeted	Untargeted and Targeted	Untargeted
Optimization	Single Step	Iterative	Iterative
Objective	Maximize loss function	Minimize loss function	Minimize the distance between original and perturbed input
Neural Network	ResNet20	ResNet20	ResNet20 and DenseNet
Accuracy(test)	11%	0%	16%

CHAPTER 7
Conclusions

Our study focused on black-box and white-box attacks. Backdoor attacks are a type of black-box attack where the poisoning causes anticipated misclassification but does not significantly affect the overall accuracy of the model. We demonstrated that by changing the percentage of poisons added, the behavior of the model can be altered. At 5% and 10%, the model behaved similarly, but as the poisoning rate increased, the accuracy eventually dropped. Additionally, we examined how accuracy varied across different scenarios. Our experiments concluded that for an untargeted attack to succeed, the model's accuracy must be compromised. We also conducted three white-box attacks. All attacks led to misclassification either by decreasing accuracy or not affecting the model's overall accuracy. DeepFool attacks are iterative, making it the most deceitful among the three attacks. FGSM attempts to maximize the loss function concerning the input data and uses it to generate a perturbed input. CW attacks the model by minimizing the differential surrogate objective. We compared the accuracy degradation of two neural networks for CW with the baseline model. It is a well-known fact that poisoned samples cause accuracy to decline in machine learning. The model's performance is undermined, resulting in poorer accuracy, when the training data contains malicious or deceptive examples. But taking different numbers of malicious clients to carry out the process also greatly impacted the results. The samples were created by carefully crafting the image's perturbations, considering the regular image and the perturbed image were not distinguishable in the case of White-box attacks, whereas in Black-box attacks, a Backdoor attack was conducted where any kind of pattern acts as a backdoor trigger and compromises the model's performance. We demonstrated that changing the number of adversarial samples and the number of malicious clients can alter the model's behavior.

References

[1] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks, 2017.

[2] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh. ZOO. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. ACM, nov 2017.

[3] X. Chen, C. Liu, B. Li, K. Lu, and D. Song. Targeted backdoor attacks on deep learning systems using data poisoning, 2017.

[4] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples, 2015.

[5] T. Gu, B. Dolan-Gavitt, and S. Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain, 2019.

[6] A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial machine learning at scale, 2017.

[7] Y. Li, H. Zhong, X. Ma, Y. Jiang, and S.-T. Xia. Few-shot backdoor attacks on visual object tracking, 2022.

[8] Y. Liu, X. Chen, C. Liu, and D. Song. Delving into transferable adversarial examples and black-box attacks, 2017.

[9] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data, 2023.

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

[11] A. Nazemi and P. Fieguth. Potential adversarial samples for white-box attacks, 2019.

[12] A. Nguyen and A. Tran. Wanet – imperceptible warping-based backdoor attack, 2021.

[13] N. Rodríguez-Barroso, D. Jiménez-López, M. V. Luzón, F. Herrera, and E. Martínez-Cámara. Survey on federated learning threats: Concepts, taxonomy on attacks and defences, experimental study and challenges. *Information Fusion*, 90:148–173, feb 2023.

[14] A. Saha, A. Subramanya, and H. Pirsiavash. Hidden trigger backdoor attacks, 2019.

[15] G. Severi, J. Meyer, S. Coull, and A. Oprea. Explanation-Guided backdoor poisoning attacks against malware classifiers. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1487–1504. USENIX Association, Aug. 2021.

[16] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks, 2018.

[17] G. Sun, Y. Cong, J. Dong, Q. Wang, and J. Liu. Data poisoning attacks on federated machine learning, 2020.

[18] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks, 2014.

[19] V. Tolpegin, S. Truex, M. Gursoy, and L. Liu. Data poisoning attacks against federated learning systems, 07 2020.

[20] Y. Wang, J. Liu, X. Chang, J. Wang, and R. J. Rodríguez. Di-aa: An interpretable white-box attack for fooling deep neural networks, 2021.

[21] C. Yang, Q. Wu, H. Li, and Y. Chen. Generative poisoning attack method against neural networks, 2017.

[22] Z. Zhang, A. Panda, L. Song, Y. Yang, M. W. Mahoney, J. E. Gonzalez, K. Ramchandran, and P. Mittal. Neurotoxin: Durable backdoors in federated learning, 2022.

[23] S. Zhao, X. Ma, X. Zheng, J. Bailey, J. Chen, and Y.-G. Jiang. Clean-label backdoor attacks on video recognition models, 2020.

[24] X. Zhou, M. Xu, Y. Wu, and N. Zheng. Deep model poisoning attack on federated learning. *Future Internet*, 13(3):73, Mar 2021.