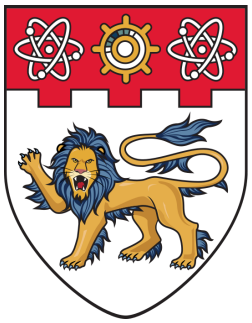

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Final Year Project Report

Effects of Incremental Training on Watermarked Neural Networks

HENG CHUAN SONG

U2022930J

Supervisor: Associate Professor Anupam Chattopadhyay

School of Computer Science and Engineering

Mar, 2023

Abstract

Deep learning has achieved extraordinary results in many different areas, ranging from autonomous driving [1], medical devices [2] to speech recognition and natural language processing [3]. Generating a high-performance neural network is costly in aspects of time, computational resources, and expertise, making the models valuable intellectual property (IP). As a result, there has been a notable growth in attention and investments in the paradigm of machine learning. In recent years, watermarking methods have been developed in order to protect the Intellectual Property Rights (IPR) of neural networks, and many schemes have successfully prevented adversaries from stealing such models. However, little has been studied on how Incremental Training would affect the persistence of watermarks in such watermarking schemes. This investigation aims to discover the effects of Incremental Training on in existing watermarking schemes.

Keywords: Intellectual Property Rights (IPR), Watermarking, Incremental Training

Acknowledgement

I would like to express my sincere gratitude towards my project supervisor, Associate Professor Anupam Chattopadhyay for his patience, expertise, and dedication have been crucial in helping me navigate the challenges that arose during the course of this Final Year Project. His valuable insights, feedback, and encouragement have been instrumental in shaping the final outcome of my work.

I would also like to thank my project officer, Tu Ngo for his constant guidance and support. He would always go out of his way to clear whatever doubts I might have, going above and beyond to make sure I understand the concepts. Not only do I have utmost respect for his knowledge, but I am also grateful for the camaraderie we have forged.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Background

The demand for high-performance neural networks opened the floodgates to a new market of Machine Learning as a Service (MLaaS) [4] . Companies of this trending market would provide services to train and tune models according to the clients' needs at a negligible cost, as compared to the cost of infrastructure and preparation needed to train their own models. As the availability of data increases over time, clients have the option to further fine-tune the models for increased accuracy, or even perform Transfer Learning to solve similar machine learning problems [5]. This makes MLaaS a very viable purchase for small and medium sized businesses that lack the sophisticated hardware to train models. The biggest concerns of MLaaS are the legality and security issues behind the neural network models, specifically pertaining to intellectual property rights (IPR) [6]. Examples include clients of the services redistributing the models outside of the contractual agreement, or even selling the models to other customers, hence directly threatening the business. Intuitively, it is necessary to develop a robust safeguard mechanism to protect the IPR of such businesses. Extensive research has been done to develop digital watermarking methods in efforts of preserving IPR. Some notable studies include ROWBACK, a watermarking scheme which robustness leverages on adversarial samples in the Trigger Set and the uniform distribution of backdoors throughout the layers in the neural networks [7].

1.2 Motivation

The robustness of watermarking schemes such as ROWBACK [7], Randomized Smoothing [8] and BlackMarks [9] have been proven against notable watermark removal attacks like Re-markable [10], but much uncertainty exists on how they are affected by Incremental Training. Incremental Training refers to enriching models with newly acquired training data in efforts of improving model performance. The reliability of robust watermarking schemes largely depends on their verification through the Trigger Set. The event of having watermarks being unintentionally removed during Incremental Training would greatly undermine the scheme's ability to protect rightful ownership. Hence, it is of utmost importance to verify their susceptibility to Incremental Training.

1.3 Objective and scope

This study aims to investigate on various existing watermarking scheme's ability to maintain verifiable, retaining IPR and robustness against adversaries, even after Incremental Training. We will

also attempt to discover how certain variables such as Trigger Set Type and learning rates within such watermarking schemes would affect Incremental Training.

In this paper, we utilized the CIFAR-10 Dataset in our experiments with the various watermarking schemes. We will first split the dataset into different portions such as Train Set, Incremental Set and Test Set. We will then train a ResNet model [11] with Train Set and generate the appropriate Trigger Set, e.g. randomly mislabelled samples, adversarial samples [12], unrelated samples, etc. The model will then be embedded with the watermarks and tested for the Trigger Set Accuracy. It is key that at this point, the Trigger Set Accuracy recorded is of high value before the actual experimentation. Finally, we will commence Incremental Training and study the results for the different watermarking schemes.

The findings will assist in developing a framework for a robust watermarking scheme for neural networks, with ability to support Incremental Training, ultimately preserving IPR.

Chapter 2

Literature review

2.1 Neural Networks

A neural network is a kind of machine learning model that forms the basis of deep learning algorithms. They are also referred to as artificial neural networks or simulated neural networks. These networks are modeled after the structure and function of the human brain, where biological neurons communicate with each other. [13]

Standard neural networks comprise of multiple neurons, each responsible in producing numerical activation values. Input neurons are activated through perceiving the environment through sensors, while other neurons are activated from previously activated counterparts. [14] Layers of such neurons are often accountable for specific characteristics, with multiple layers needed to fully function as a neural network model.

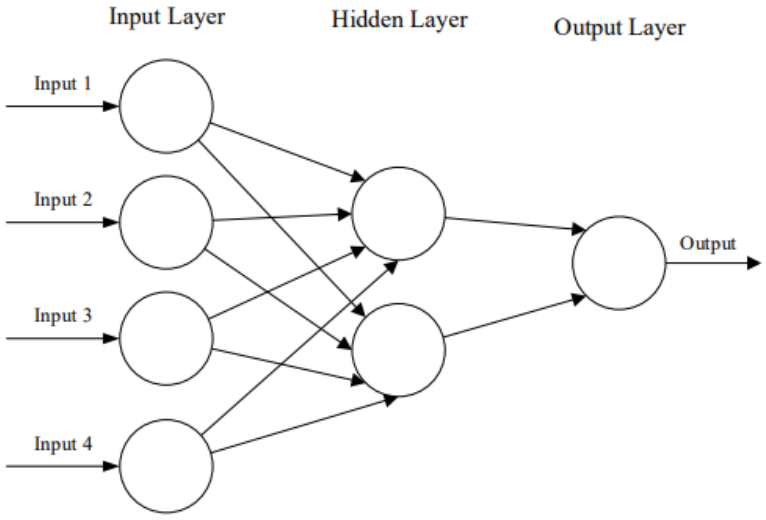

Figure 2.1: Example of a simple neural network
[15]

2.2 Deep Learning

Deep learning refers to neural networks that have greater depth in layers. Deep learning has attained immense success in multiple areas, allowing multiple processing layers for different individual features to successfully process data in natural forms. This led to the creation of models that can imitate a human mind to classify and recognize images, scenes and objects irregardless of complexity [16].

2.3 Datasets

2.3.1 CIFAR-10

The CIFAR-10 Dataset (Canadian Institute for Advanced Research, 10 classes) consists of 60000 32x32 colored images with 10 mutually exclusive classes, containing 6000 images of each class [17]. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 6,000 images of each class.

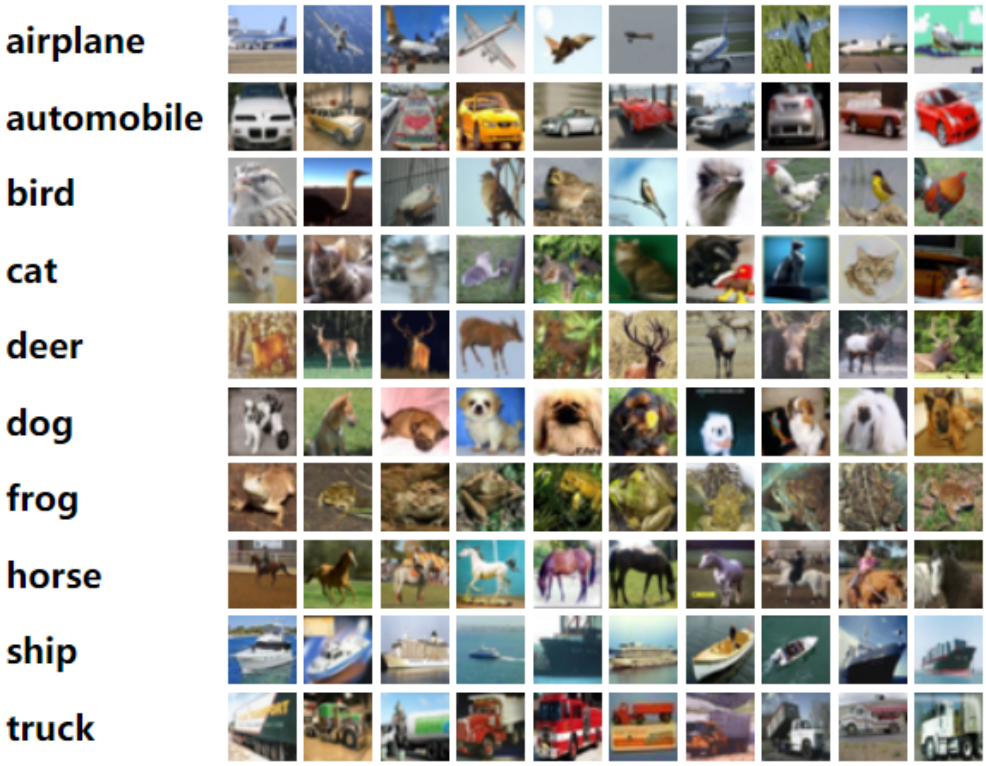

Figure 2.2: CIFAR-10 Dataset
[18]

2.3.2 MNIST

The MNIST Dataset (Modified National Institute of Standards and Technology) consists of handwritten digits. It was created by "re-mixing" samples from NIST's original datasets. [19]

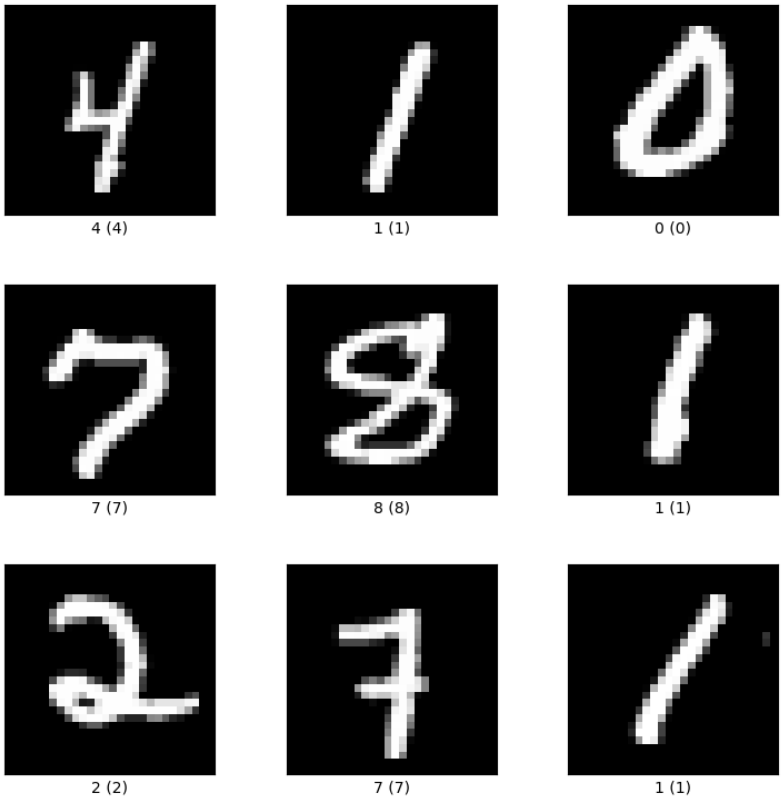

Figure 2.3: MNIST Dataset
[20]

2.4 Watermarking in Neural Networks

Protecting the Intellectual Property Rights (IPR) of Deep Neural Networks (DNNs) has become increasingly important in the recent years due to demand of MLaaS [4]. As such, watermarking schemes have been developed to protect the ownership rights. All watermarking schemes are said to follow a concept known as the watermarking trade-off triangle [21].

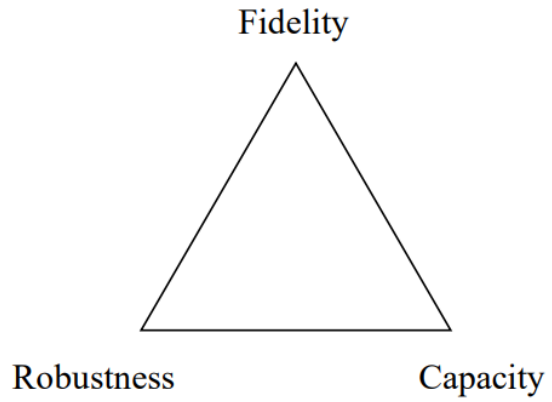

Figure 2.4: Trade-off triangle of Watermarking
[21]

The triangle depicts that the **Capacity** requirement, representing the payload or number of bits of the watermark, conflicts with two other requirements:

1. **Fidelity**, which refers to how subtle the watermark is whilst successfully providing ownership verification.

2. **Robustness**, which refers to the persistence of a watermark despite alterations to a neural network models.

2.5 Watermarking Methods

There are two main types of watermarking schemes for neural networks: 1) Watermark embedding and 2) Data poisoning.

1. Watermarks are embedded deep within the neural architecture design during construction, as shown in multiple works such as DeepMarks [22], DeepSigns [23] and Dawn [24].

2. Specially curated samples in training data that alter the model parameters, also known as data poisoning. Some notable schemes are ROWBACK [7], Certified Randomized Smoothing Scheme [8] and the Original Watermarking Scheme by Backdooring [25].

2.6 Trigger Set

A Trigger Set [21] is a set of input samples used in the process of watermarking neural networks. In the context of watermarking, a Trigger Set is used to embed a unique and identifiable signature

into the neural network. Trigger Sets usually incorporate an element of randomness to prevent adversaries from being able to retrieve a model's Trigger Set, which in turn would allow ownership verification for the adversaries. Hence, the confidentiality of the Trigger Set would serve as a baseline for a robust watermarking scheme [26]

2.6.1 Types of Trigger Set

There are many ways of generating a Trigger Set, with the requirement being that it provides a form of unique and identifiable signature for a neural network. However, under the scope of our experiments, we would focusing on 3 main kinds of Trigger Sets.

Randomly Mislabelled Samples

Random Mislabelled Samples Trigger Set refers to data from the same data set as the Training Set, but deliberately mislabelled. This process of mislabelling relies on randomness, to prevent adversaries from being able to guess the Trigger Set.

Unrelated Samples

Unrelated Samples Trigger Set refers to images from an unrelated data set with respect to the original training data. For example, using the MNIST Dataset for Trigger Set for a CIFAR-10 trained model, vice versa.

Adversarial Samples

Adversarial Samples Trigger Set is generated using the Fast Gradient Sign Method (FGSM) described by Goodfellow et al [27]. It revolves around adding perturbations to a clean image and further mislabelling the perturbed image to a label disjointed from the Ground Truth label and also the perturbed image label to induce maximal randomness.

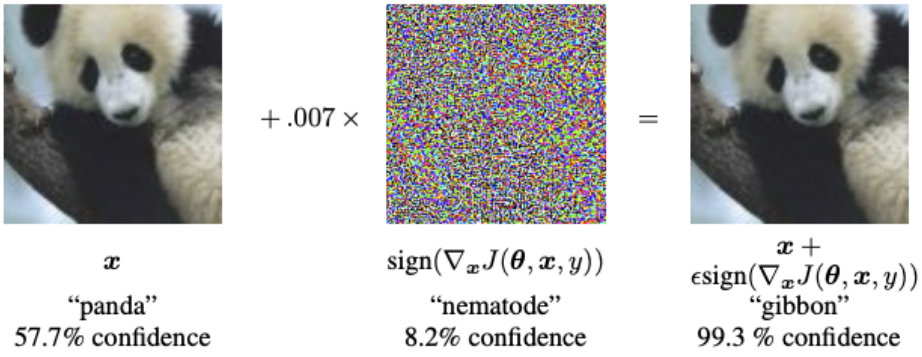

Figure 2.5: Fast Gradient Sign Method to generate Adversarial Images [27]

2.7 Notable Watermarking Schemes

2.7.1 Watermarking by Backdooring

This scheme by Adi et al.[25] was one of the first works that introduced the over-parameterization of neural networks to act as a watermarking algorithm. Backdooring in Machine Learning refers to adversaries causing a model to output incorrect labels for certain inputs, which is usually a malign act. However, the scheme leverages on this to design a backdoor that can be used for model identification, protecting the IPR of such neural network models.

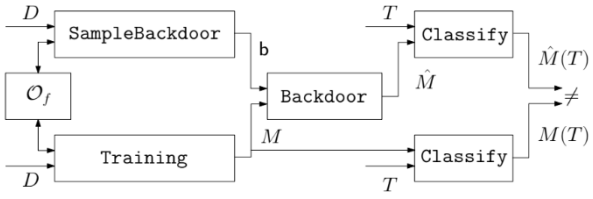

Figure 2.6: Schematic diagram of backdooring process
[25]

To remove suspicion from models being backdoors, the backdoor process is redefined. The removal of such backdoors should be nontrivial even for an adversary with advanced knowledge of the algorithm SampleBackdoor. To achieve this, it is stipulated that SampleBackdoor must possess certain characteristics [25]:

1. **Multiple Trigger sets.**
   When SampleBackdoor provides a Trigger Set as part of a backdoor, we assume that each set has a minimum size of n. Additionally, the algorithm should be collision resistant, where no two randomly picked Trigger Sets shall intersect.

2. **Persistency.**
   Persistency refers to the difficulty of removing a backdoor within a neural network, unless knowledge of the Trigger Set is present.

2.7.2 ROWBACK

ROWBACK [7] was implemented with consideration over existing literature in the field and vulnerabilities of current watermarking schemes. The scheme leverages on two properties of neural networks, adversarial samples generated through structured perturbations and the ability to embed backdoors during training.

ROWBACK guarantees robustness through two main properties:

1. Re-designed Trigger Set of adversarial samples with random Trigger Labels. This ensures functionality of the model without compromising strong ownership verification with embedded watermarks. The Trigger Set is also extremely difficult to be replicated due to the infinite possibilities of adversarial samples.

2. Uniform distribution of the watermarks throughout the neural networks thwarts model modification attacks, which implicates adversaries into performing computational efforts equivalent to training a network from scratch. Every layer is explicitly embedded with watermarks to ensure uniform distribution.

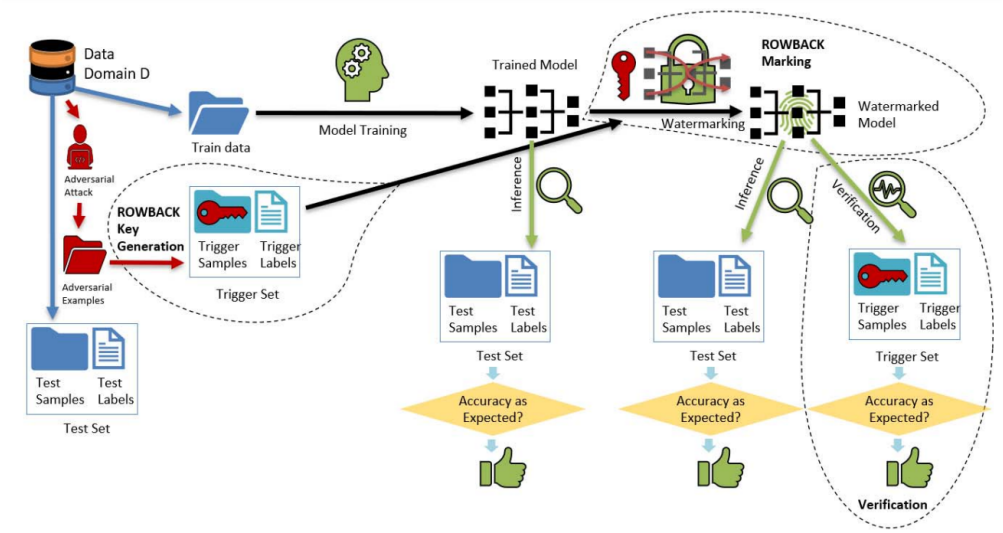

Figure 2.7: Schematic diagram of ROWBACK
[7]

2.7.3 Uniform Distribution Watermarking

For the sake of our experiments, we would use a watermarking method that utilizes uniform distribution of watermarks as proposed by ROWBACK [7] to act as a control scheme for comparison. The difference between this scheme and ROWBACK lies within the Trigger Set Type. This will be further discussed in the following chapters.

2.7.4 Certified Watermarking with Random Smoothing

This scheme presents a certifiable watermarking method [8] that utilizes the randomized smoothing technique proposed in Chiang et al.[28], guaranteeing that watermarks won't be removed unless model parameters differ more than a distance of ℓ_2.

Algorithm 1 Embed Certifiable Watermark

Required: training samples X, trigger set samples $X_{trigger}$, learning rate τ, maximum noise level ϵ, replay count k, noise sample count t

for epoch = 1, ... , N **do**
  for $B \subset X$ **do**
    $g_\theta \leftarrow E_{(x,y)\in B}[\nabla_\theta l(x, y, \theta)]$
    $\theta \leftarrow \theta - \tau g_\theta$
  for $B \subset X_{trigger}$ **do**
    $g_\theta = 0$
    for i = 1 to k **do**
      $\sigma \leftarrow \frac{i}{k}\epsilon$
      for j = 1 to t **do**
        $G \sim N(0, \sigma^2 I)$
        $g_\theta \leftarrow g_\theta + E_{(x,y)\in B}[\nabla_\theta l(x, y, \theta + G)]$
      $g_\theta \leftarrow g_\theta/(kt)$
      $\theta \leftarrow \theta - \tau g_\theta$

Figure 2.8: Algorithm for embedding certifiable watermark [8]

During training, the watermark is incorporated by introducing Gaussian noise to the Trigger Set images with the desired labels. To avoid instability during training, the noise is gradually increased within each epoch, with a standard deviation between 0 and 1. It was observed that the test accuracy decreases when embedding the watermark with this technique, and hence the solution to this was to warm up the model with regular training without watermark embedding for the first five epochs. The detailed training method can be found in the above algorithm.

2.8 Incremental Training

Incremental Training refers to fine-tuning a model with newly acquired data to improve a model's performance. The persistence of the Trigger Set would be quantified with the Trigger Accuracy, which is determined by the number of correct classifications divided by the number of actual classifications done. Hence, we would be observing the change in Trigger Accuracy after Incremental Training. This would be the core concept for our experiments, where Incremental Training would be done on watermarked pre-trained models to evaluate a scheme's suitability for Incremental Training. This would be pivotal in ensuring a scheme's usability and robustness.

Chapter 3

Methodology

In this section, we will go through the environmental details and steps taken to carry out the investigation

3.1 Experimental Setup

3.1.1 Hardware Specifications

- CPU: 12th Gen Intel(R) Core(TM) i7-12700
- RAM: 32GB
- GPU: NVIDIA RTX A4000

3.1.2 Libraries

- pandas: 1.51
- jupyterlab: 3.5.0
- ipykernel: 6.17.0
- tensorflow: 2.10.0
- torchvision: 0.14.0
- Pillow: 9.3.0

3.2 Optimizers

3.2.1 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a continuation from Gradient Descent that overcame the issue of requiring large memory space. Gradient Descent is the most fundamental optimizer that has been widely used across the neural networks. It leverages on a convex function and alters its parameters to achieve local minimum in an iterative fashion. It's known to be a relative simple but reliable algorithm, but suffers from some problems such as local minima trapping and large memory requirements[29].SGD computes one training sample at a time, calculating the corresponding loss and updating the respective parameters after every sample. Introducing momentum to SGD has proven to be beneficial towards convergence speed and reduction in oscillations. [30]

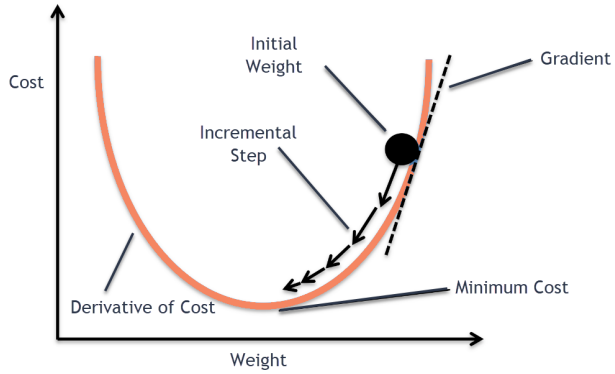

Figure 3.1: Gradient Descent Algorithm Visualized
[31]

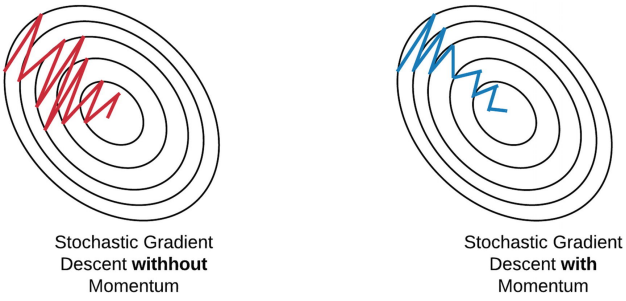

Figure 3.2: Stochastic Gradient Descent Visualized
[32]

3.2.2 Adaptive Moment Estimation

Adaptive Moment Estimation (Adam) can be viewed as a combination of RMS Prop and SGD with momentum.

Root Mean Square (RMS) Prop algorithm leverages on minimizing evaluating functions to reach local minima to accelerate optimization. It stores moving average of squared gradients for every parameter and dividing the gradient with the root mean square. RMS Prop converges rapidly and requires less intervention compared to gradient descent algorithms. [29]

3.3 Cross-Entropy Loss

The main loss function that we will utilize in our experiments would be the Cross-Entropy Loss Function [33].

This method calculates a penalty score for each predicted class probability based on how far it deviates from the desired output of 0 or 1. The degree of penalty is determined by the logarithm of the difference between the predicted and desired probabilities. Large deviations close to 1 are punished more severely than small deviations approaching 0. During model training, the objective is to minimize the cross-entropy loss by adjusting the model weights. The optimal model achieves a cross-entropy loss of 0 [33].

3.4 ResNet-18

Our models are all built on ResNet-18 [11]. ResNet is a type of Convolutional Neural Network (CNN) designed to handle hundreds or even thousands of convolutional layers. Unlike earlier CNN architectures that could not scale to a large number of layers and had limited performance, ResNet addresses the "vanishing gradient" problem that occurs when too many layers are added. This problem arises during the backpropagation process that utilizes gradient descent. The repeated multiplication of gradients eventually leads to a disappearing gradient, resulting in saturation of performance, along with performance deterioration for subsequent addition of layers [34].

ResNet solves this problem by introducing "skip connections". It identifies the convolutional layers that do nothing at the beginning, skips them, and runs the activations from previous layers. This process speeds up initial training through reduction of layers. During network retraining later, the layers are then expanded and the residual parts will be allowed to traverse more into the input image's feature space.

3.5 Model Preparation

This portion will explain the steps taken to prepare our baseline models used in carrying out the Incremental Training experiments.

3.5.1 Data Preparation

The CIFAR-10 Dataset [17] is Split into Training Set (50000 Images) and Test Set (10000 Images) by default. For our experiment, we further split the Training Set into:

1. Train Set Used in training the model for its original task.
2. Incremental Set Similar to the Train Set in terms of data, but used only after embedding watermarks to simulate Incremental Training. Also will always be smaller or equal to the size of Train Set within the scope of this experiment.
3. Trigger Set Data that is used to watermark the neural network, used for model identification.

The Trigger Set size will be of 100 samples, and the Train Set to Incremental Set will be split with a ratio in accordance to the experimental requirements. However, the experiments conducted will not have the Incremental Set surpassing the Train Set at any point of time. The reason for this is such that the Incremental Set is supposedly a simulation of clients fine-tuning a purchased model with new data, and that there exists is a bound where the model has been incrementally trained

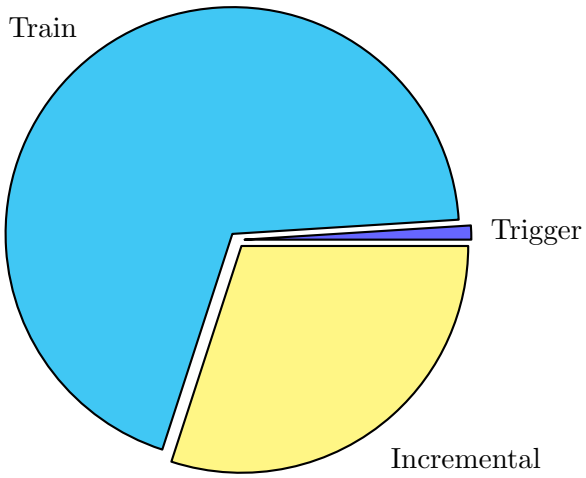

Figure 3.3: Training Set Split

so much that the ownership of the model should be reconsidered. The upper bound is established as the size of the Train Set, and any further Incremental Training done would not be considered in the following experiments.

3.5.2 Warming up of Model

Before embedding the Trigger Set within the neural networks, we will first look to warm-up the model so that the model's parameters are optimized towards fitting to actual data. Embedding the Trigger Set too early on would cause the neural network models to deviate from its original task of classification, resulting in possible accuracy issues. Hence warming up the model with it's original task data would be intuitive. Note that this warming up procedure is commonly practiced in neural network robustness literature [35]. The same warm up procedure is done with all the different experiments.
For the baseline model, the following specifications were used:

- Network: ResNet18
- Epochs: 10
- Optimizer: Adam

Hyperparameter	**Value**
Learning rate	5e-4
Weight decay	1e-4

Table 3.1: Hyperparameters for Warmup Adam Optimizer

- Criterion: Cross-entropy Loss
- Scheduler: StepLR

Hyperparameter	Value
Step size	20
Learning rate decay	0.5

Table 3.2: Hyperparameters for Warmup Scheduler StepLR

3.6 Incremental Training test with Variation in Split Sizes

3.6.1 Watermark embedding

This is where we start to embed our Trigger Set into our models. For the sake of our experiment, we will test out 4 different watermarking schemes. The original watermarking scheme by [25], ROWBACK [7], Uniform Watermarking Scheme and the Randomized Smoothing Watermarking Scheme [8].

Watermark embedding for Randomized Smoothing Scheme

- Network: ResNet18
- Epochs: 100
- Optimizer: SGD

Hyperparameter	Value
Learning rate	0.05
Weight decay	1e-4
Momentum	0.9

Table 3.3: Hyperparameters for Randomized Smoothing Baseline Adam Optimizer - Split Size Test

- Criterion: Cross-entropy Loss
- Scheduler: StepLR

Hyperparameter	Value
Step size	30
Learning rate decay	0.1

Table 3.4: Hyperparameters for Randomized Smoothing Baseline Scheduler StepLR - Split Size Test

- Trigger Set: Unrelated Set MNIST
- Trigger Set Size: 100
- Uniform Trigger Set Distribution: Yes

Watermark embedding for Original Backdoor Scheme

- Network: ResNet18
- Epochs: 80
- Optimizer: Adam

Hyperparameter	**Value**
Learning rate	2e-4
Weight decay	1e-4

Table 3.5: Hyperparameters for Original Backdoor Scheme Baseline Adam Optimizer - Split Size Test

- Criterion: Cross-entropy Loss
- Scheduler: StepLR

Hyperparameter	**Value**
Step size	10
Learning rate decay	0.1

Table 3.6: Hyperparameters for Original Backdoor Scheme Baseline Scheduler StepLR - Split Size Test

- Trigger Set: Randomly Mislabelled Samples CIFAR-10
- Trigger Set Size: 100
- Uniform Trigger Set Distribution: No

Watermark embedding for ROWBACK Scheme

- Network: ResNet18
- Epochs: 80
- Optimizer: Adam

Hyperparameter	**Value**
Learning rate	2e-4
Weight decay	1e-4

Table 3.7: Hyperparameters for ROWBACK Scheme Baseline Adam Optimize - Split Size Test

- Criterion: Cross-entropy Loss
- Scheduler: StepLR

Hyperparameter	**Value**
Step size	10
Learning rate decay	0.1

Table 3.8: Hyperparameters for ROWBACK Scheme Baseline Scheduler StepLR - Split Size Test

- Trigger Set: Adversarial Samples CIFAR-10
- Trigger Set Size: 100
- Uniform Trigger Set Distribution: Yes

Watermark embedding for Uniform Watermarking Scheme

- Network: ResNet18
- Epochs: 80
- Optimizer: Adam

Hyperparameter	Value
Learning rate	2e-4
Weight decay	1e-4

Table 3.9: Hyperparameters for Uniform Watermarking Scheme Baseline Adam Optimizer - Split Size Test

- Criterion: Cross-entropy Loss
- Scheduler: StepLR

Hyperparameter	Value
Step size	10
Learning rate decay	0.1

Table 3.10: Hyperparameters for Uniform Watermarking Scheme Baseline Scheduler StepLR - Split Size Test

- Trigger Set: Randomly Mislabelled Samples CIFAR-10
- Trigger Set Size: 100
- Uniform Trigger Set Distribution: Yes

3.6.2 Incremental Training for Data Split Size Test

This section would be the core part of the Data Split Size Test, to see how the persistence of the Trigger Set would be affected by Incremental Training.

For the Incremental Training part, we would be using the same hyperparameters throughout to maintain fairness. This part would also be a simulation of how the models would be handled by clients, which aligns with ensuring robustness of watermarking schemes for MLaaS [4].

Variation in Train to Incremental Set Split Size

In a typical MlaaS [4] model, it would have been trained with large amounts of data before being sold off to clientele. Hence, we would expect clients to perform some sort of transfer learning [5] or Incremental Training on the purchased model, but with a much smaller portion of training data. We therefore hold the assumption that the clients will only perform Incremental Training to up to 50% of the original data size.

For the Incremental Training test on different Split Sizes, the following specifications were used:

- Network: ResNet18
- Epochs: 100
- Optimizer: SGD

Hyperparameter	**Value**
Learning rate	1e-3
Weight decay	1e-4
Momentum	0.9

Table 3.11: Hyperparameters for Incremental Training SGD Optimizer - Split Size Test

- Criterion: Cross-entropy Loss

3.7 Incremental Training with Trigger Set Type test

3.7.1 Watermark embedding

In this part of the experiment, we proceed to investigate how different Trigger Set Types would affect the persistence of Trigger Set among the different watermarking schemes. We would be using the Certified Randomized Smoothing scheme [8], the Original Watermarking Scheme [25] and also a watermarking scheme that leverages on Uniform Distribution of watermarks. The Trigger Set Types we would carry out the investigation with are:

1. Randomly Mislabelled Samples Samples CIFAR-10
2. Adversarial Samples CIFAR-10
3. Unrelated Samples MNIST

Watermark embedding for Randomized Smoothing Scheme

- Network: ResNet18
- Epochs: 100
- Optimizer: SGD

Hyperparameter	Value
Learning rate	0.05
Weight decay	1e-4
Momentum	0.9

Table 3.12: Hyperparameters for Randomized Smoothing Baseline Adam Optimizer - Trigger Set Type Test

- Criterion: Cross-entropy Loss
- Scheduler: StepLR

Hyperparameter	Value
Step size	30
Learning rate decay	0.1

Table 3.13: Hyperparameters for Randomized Smoothing Baseline Scheduler StepLR - Trigger Set Type Test

- Trigger Set: Unrelated Set MNIST/ Randomly Mislabelled Samples CIFAR-10/ Adversarial Samples CIFAR-10
- Trigger Set Size: 100
- Uniform Trigger Set Distribution: Yes

Watermark embedding for Original Backdoor Scheme

- Network: ResNet18
- Epochs: 80
- Optimizer: Adam

Hyperparameter	Value
Learning rate	2e-4
Weight decay	1e-4

Table 3.14: Hyperparameters for Original Backdoor Scheme Baseline Adam Optimizer - Trigger Set Type Test

- Criterion: Cross-entropy Loss
- Scheduler: StepLR

Hyperparameter	Value
Step size	10
Learning rate decay	0.1

Table 3.15: Hyperparameters for Original Backdoor Scheme Baseline Scheduler StepLR - Trigger Set Type Test

- Trigger Set: Unrelated Set MNIST/ Randomly Mislabelled Samples CIFAR-10/ Adversarial Samples CIFAR-10
- Trigger Set Size: 100
- Uniform Trigger Set Distribution: No

Watermark embedding for Uniform Watermarking Scheme

- Network: ResNet18
- Epochs: 80
- Optimizer: Adam

Hyperparameter	Value
Learning rate	2e-4
Weight decay	1e-4

Table 3.16: Hyperparameters for Uniform Watermarking Scheme Baseline Adam Optimizer - Trigger Set Type Test

- Criterion: Cross-entropy Loss
- Scheduler: StepLR

Hyperparameter	Value
Step size	10
Learning rate decay	0.1

Table 3.17: Hyperparameters for Uniform Watermarking Scheme Baseline Scheduler StepLR - Trigger Set Type Test

- Trigger Set: Unrelated Set MNIST/ Randomly Mislabelled Samples CIFAR-10/ Adversarial Samples CIFAR-10
- Trigger Set Size: 100
- Uniform Trigger Set Distribution: Yes

3.7.2 Incremental Training for Trigger Set Type Test

This section would be the core part of the Trigger Set Type Test, to study on how the different Trigger Set Types would react in Incremental Training.

For the Incremental Training part, we would be using the same hyperparameters throughout to maintain fairness. This part would also be a simulation of how the models would be handled by clients, which aligns with ensuring robustness of watermarking schemes for MLaaS [4].

For the Incremental Training on Trigger Set Type tests, the following specifications were used:

- Network: ResNet18
- Epochs: 100
- Optimizer: SGD

Hyperparameter	**Value**
Learning rate	1e-3
Weight decay	1e-4
Momentum	0.9

Table 3.18: Hyperparameters for Trigger Type Test SGD Optimizer - Trigger Set Type Test

- Criterion: Cross-entropy Loss

3.8 Incremental Training test with Variation in Learning Rates

3.8.1 Watermark embedding

In this part of the experiment, we proceed to investigate how different learning rates would affect the persistence of Trigger Set among the different watermarking schemes. We would be using the Certified Randomized Smoothing scheme [8], the Original Watermarking Scheme [25] and also a watermarking scheme that leverages on Uniform Distribution of watermarks.

Watermark embedding for Randomized Smoothing Scheme

- Network: ResNet18
- Epochs: 100
- Optimizer: SGD

Hyperparameter	**Value**
Learning rate	0.05
Weight decay	1e-4
Momentum	0.9

Table 3.19: Hyperparameters for Randomized Smoothing Baseline Adam Optimizer - Learning Rate Test

- Criterion: Cross-entropy Loss
- Scheduler: StepLR

Hyperparameter	**Value**
Step size	30
Learning rate decay	0.1

Table 3.20: Hyperparameters for Randomized Smoothing Baseline Scheduler StepLR - Learning Rate Test

- Trigger Set: Unrelated Set MNIST
- Trigger Set Size: 100
- Uniform Trigger Set Distribution: Yes

Watermark embedding for Original Backdoor Scheme

- Network: ResNet18
- Epochs: 80
- Optimizer: Adam

Hyperparameter	Value
Learning rate	2e-4
Weight decay	1e-4

Table 3.21: Hyperparameters for Original Backdoor Scheme Baseline Adam Optimizer - Learning Rate Test

- Criterion: Cross-entropy Loss
- Scheduler: StepLR

Hyperparameter	Value
Step size	10
Learning rate decay	0.1

Table 3.22: Hyperparameters for Original Backdoor Scheme Baseline Scheduler StepLR - Learning Rate Test

- Trigger Set: Randomly Mislabelled Samples CIFAR-10
- Trigger Set Size: 100
- Uniform Trigger Set Distribution: No

Watermark embedding for Uniform Watermarking Scheme

- Network: ResNet18
- Epochs: 80
- Optimizer: Adam

Hyperparameter	Value
Learning rate	2e-4
Weight decay	1e-4

Table 3.23: Hyperparameters for Uniform Watermarking Scheme Baseline Adam Optimize - Learning Rate Testr

- Criterion: Cross-entropy Loss
- Scheduler: StepLR

Hyperparameter	Value
Step size	10
Learning rate decay	0.1

Table 3.24: Hyperparameters for Uniform Watermarking Scheme Baseline Scheduler StepLR - Learning Rate Test

- Trigger Set: Randomly Mislabelled Samples CIFAR-10
- Trigger Set Size: 100
- Uniform Trigger Set Distribution: Yes

3.8.2 Incremental Training for Learning Rate Test

This section would be the core part of the Learning Rate Test, to see how the persistence of the Trigger Set would be affected by Incremental Training.

For the Incremental Training part, we would be using the same hyperparameters throughout to maintain fairness. This part would also be a simulation of how the models would be handled by clients, which aligns with ensuring robustness of watermarking schemes for MLaaS [4].

For the Incremental Training test with Variation in Learning Rates, the following specifications were used:

- Network: ResNet18
- Epochs: 100
- Optimizer: SGD

Hyperparameter	**Value**
Learning rate	0.0001 0.0005 0.001 0.005 0.01 0.05
Weight decay	1e-4
Momentum	0.9

Table 3.25: Hyperparameters for SGD Optimizer - Learning Rate Test

- Criterion: Cross-entropy Loss

Chapter 4

Results and Discussion

It is duly noted that this is the portion where we will discuss the results after Incremental Training is done on the models prepared as mentioned in the previous chapter. The models that have been trained to this point all have a **Trigger Accuracy of 100%**, and we will be observing the change in their Trigger Accuracy as a metric for denoting the persistence of watermarks, and hence evaluating the scheme's suitability for Incremental Training. The models that would be evaluating as mentioned in the previous chapter would be the Original Backdoor Watermarking Scheme [25], Certified Randomized Smoothing Watermarking Scheme [8], ROWBACK [7] and also a Uniform Distribution Watermarking 2.7.3 used as a control scheme. The control scheme is meant to act as a comparison between ROWBACK and the Original Backdoor Scheme. The reason for this can be depicted in the following table:

Scheme	**Trigger Set Type**	**Uniform Distribution**
ROWBACK	Adversarial Samples	Yes
Original Backdoor Watermarking	Randomly Mislabelled Samples	No
Uniform Distribution	Randomly Mislabelled Samples	Yes

Table 4.1: Control Scheme depiction

The difference between the Uniform Distribution against ROWBACK and Original Backdoor Watermarking is only by 1 variable respectively, which would prove as a useful scheme to experiment on to draw conclusions and hypothesis in our experiments.

The experiments carried henceforth are as such:

1. Incremental Training with Variation in Data Split Size
2. Incremental Training with Variation in Trigger Set Type
3. Incremental Training with Variation in Learning Rates

4.1 Incremental Training with Variation in Data Split Size

This experiment was done with different split size ratio of Train Set to Incremental Training Set. Each of the experiments were done on pre-trained models with their respective sizes of Train Set with the other hyperparameters kept constant throughout. The hyperparameters for the following experiment can be referred to here at Section 3.6.

4.1.1 90 to 10 Split

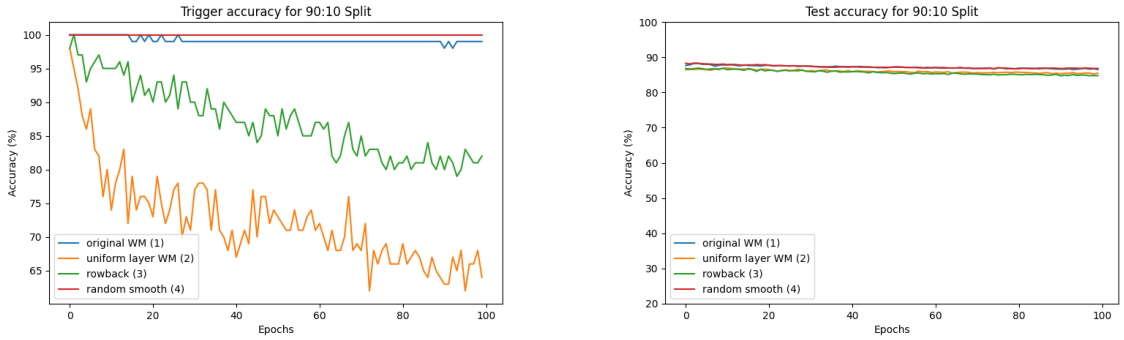

Figure 4.1: Trigger and Test Accuracy respectively for 90:10 Split

4.1.2 80 to 20 Split

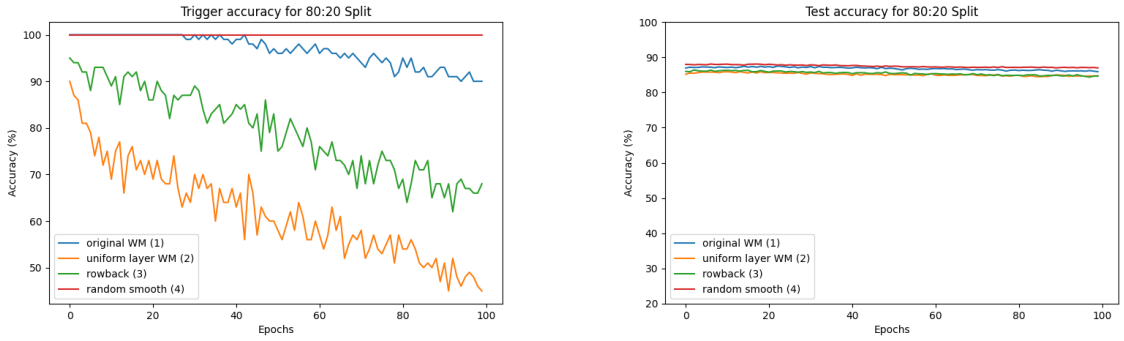

Figure 4.2: Trigger and Test Accuracy respectively for 80:20 Split

4.1.3 70 to 30 Split

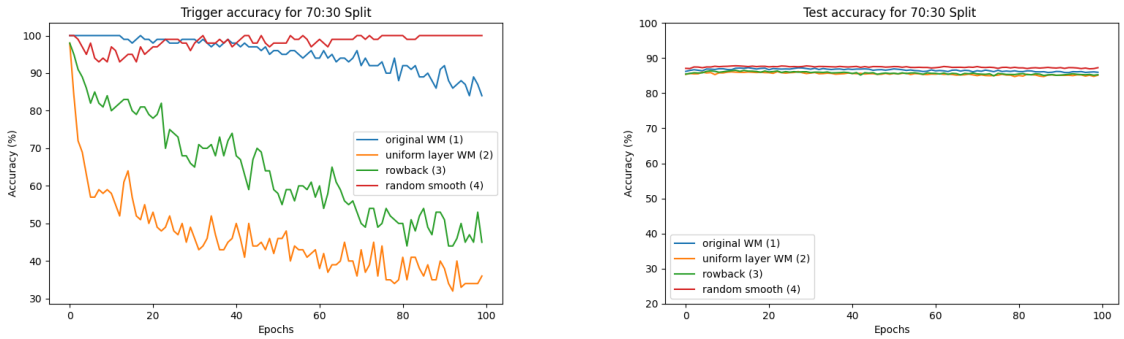

Figure 4.3: Trigger and Test Accuracy respectively for 70:30 Split

4.1.4 60 to 40 Split

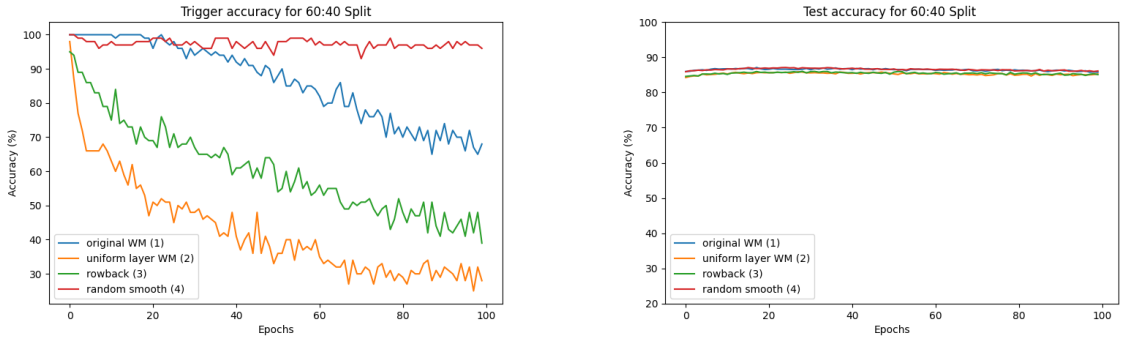

Figure 4.4: Trigger and Test Accuracy respectively for 60:40 Split

4.1.5 50 to 50 Split

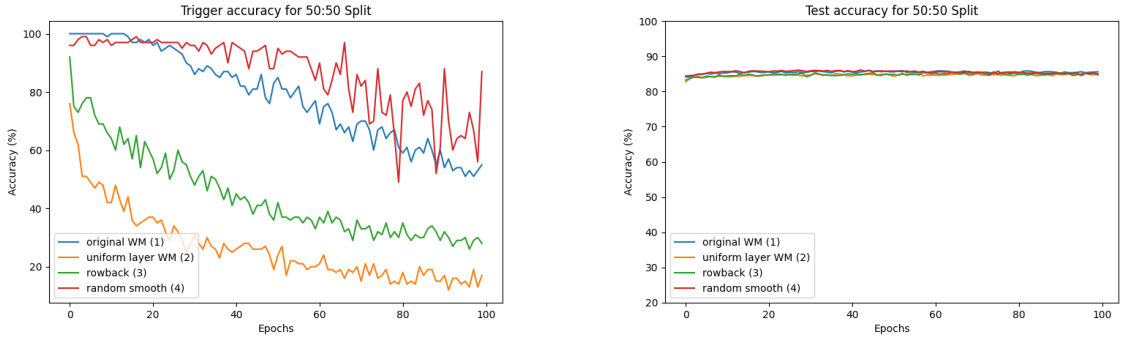

Figure 4.5: Trigger and Test Accuracy respectively for 50:50 Split

4.1.6 Summary of Results for Data Split Size Experiment

Split Ratio(T:I)	Random Smoothing	Orig. Backdoor	ROWBACK	Uniform WM
90:10	**100%**	99%	82%	64%
80:20	**100%**	90%	68%	45%
70:30	**100%**	84%	45%	36%
60:40	**96%**	68%	39%	28%
50:50	**75%**	55%	28%	17%

Table 4.2: Overview of Trigger Set Accuracy after Incremental Training

Overall, we can conclude that the Randomized Smoothing Scheme [8] and Original Backdoor Watermarking Scheme [25] supports Incremental Training. It is also observed that the Randomized Smoothing scheme maintains the highest Trigger Set Accuracy (bolded) in every Data Split Size. The schemes that leverage on uniform watermark distribution on the other hand have clear issues in maintaining high Trigger Set accuracy after Incremental Training. It is duly noted that the Test Set accuracy that denotes the accuracy of the model's original classification task is at least 80% throughout the Incremental Training.

There are also few interesting findings from the results:

1. By comparing the results between the Uniform Layer WM and ROWBACK, and knowing that the only difference between the two watermarking schemes lies within the type of Trigger Set, we can hypothesize that **Trigger Set Type plays a part in maintaining Trigger Set accuracy**. This will be further studied in the following section.

2. By comparing the results between the Uniform Layer WM and Original Backdoor Watermarking Scheme, and knowing that the only difference between the two watermarking schemes lies with the uniformity in watermark distribution, we can hypothesize that **uniform distribution causes an adverse effect towards Trigger Set accuracy when it comes to Incremental Training**.

3. It is intuitive that there is an **inverse correlation between the amount of Incremental Training done and persistence of watermarks**, e.g the more Incremental Training is

done, the more likely the watermarks would begin to fade. The results are aligned with this intuition.

4.2 Incremental Training with Variation in Trigger Set Type

Following up from the previous experiment, we will now study on the different Trigger Set Types with Incremental Training. For this experiment, we will use the Uniform Layer Watermarking, Certified Randomized Smoothing Scheme[8] and Original Watermarking Scheme [25] to test the different Trigger Set Types. The models were all pre-trained with the Trigger Set Types before the commencement of Incremental Training, where the results from the Incremental Training are visualized below. Due to the irregularity and fluctuations in the Trigger Set Accuracy during this test, it was repeated 3 times each and the mean would then be used to compare the results. The hyperparameters for the following experiment can be referred to here at Section 3.7.

4.2.1 Incremental Training Trigger Set Type Test - Randomized Smoothing

Randomized Smoothing Run 1

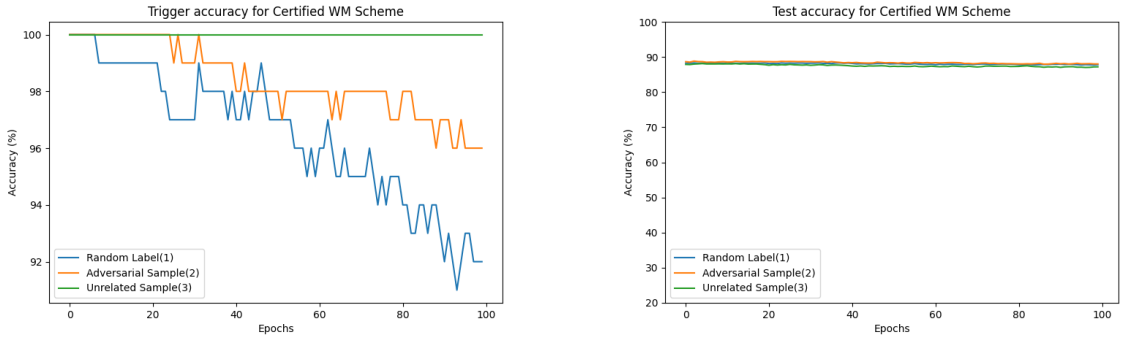

Figure 4.6: Randomized Smoothing Watermarking Trigger and Test Accuracy respectively for Run 1

Randomized Smoothing Run 2

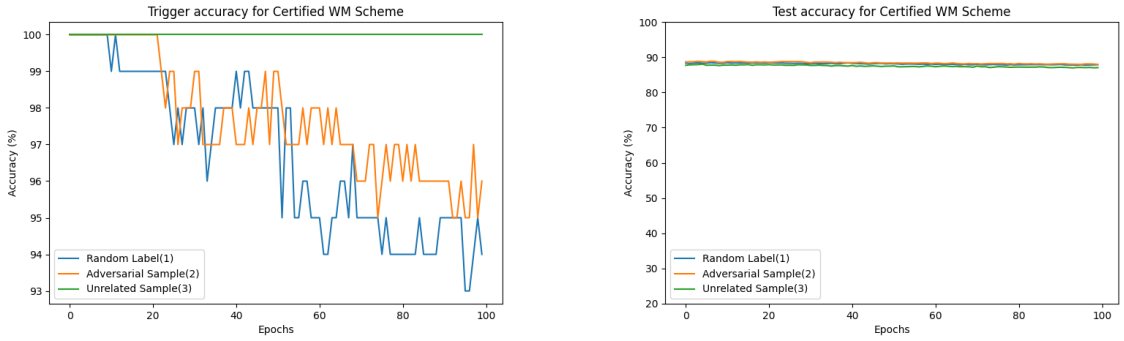

Figure 4.7: Randomized Smoothing Watermarking Trigger and Test Accuracy respectively for Run 2

Randomized Smoothing Run 3

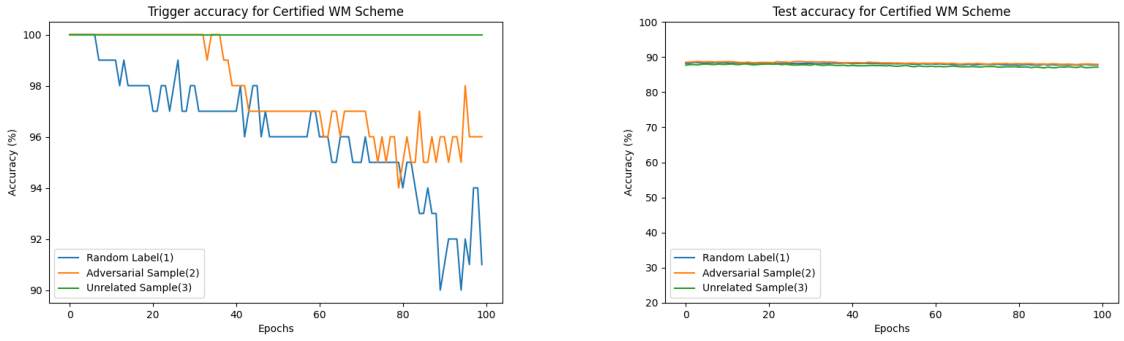

Figure 4.8: Randomized Smoothing Watermarking Trigger and Test Accuracy respectively for Run 3

4.2.2 Incremental Training Trigger Set Type Test - Uniform Watermarking

Uniform Watermarking Run 1

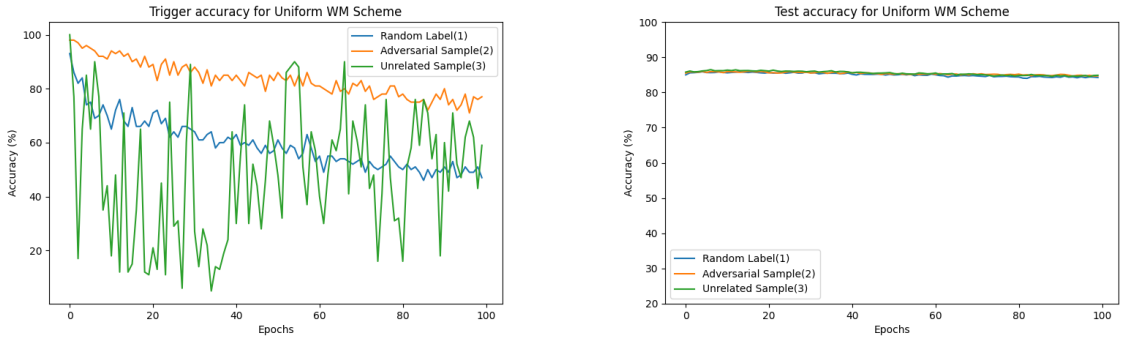

Figure 4.9: Uniform Watermarking Trigger and Test Accuracy respectively for Run 1

Uniform Watermarking Run 2

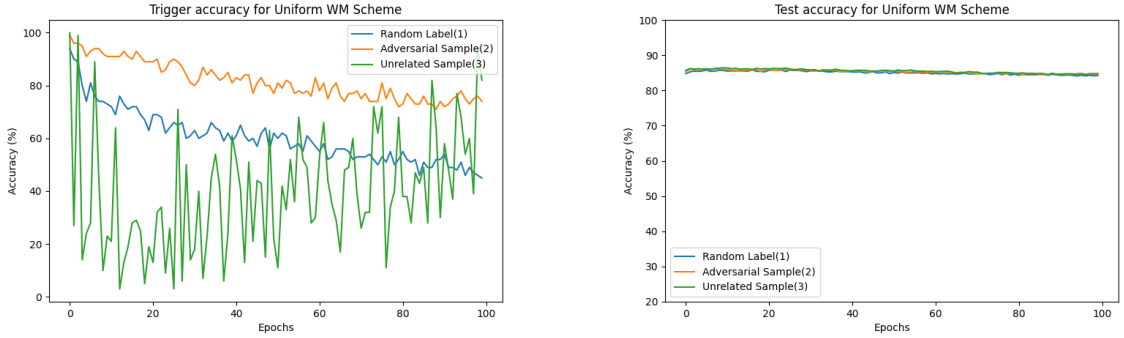

Figure 4.10: Uniform Watermarking Trigger and Test Accuracy respectively for Run 2

Uniform Watermarking Run 3

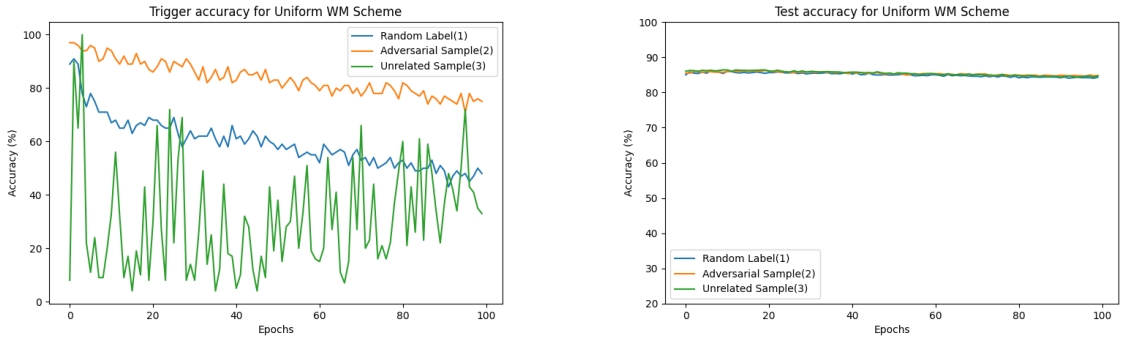

Figure 4.11: Uniform Watermarking Trigger and Test Accuracy respectively for Run 3

4.2.3 Incremental Training Trigger Set Type Test - Original Backdoor Watermarking Scheme

Original Backdoor Watermarking Scheme Run 1

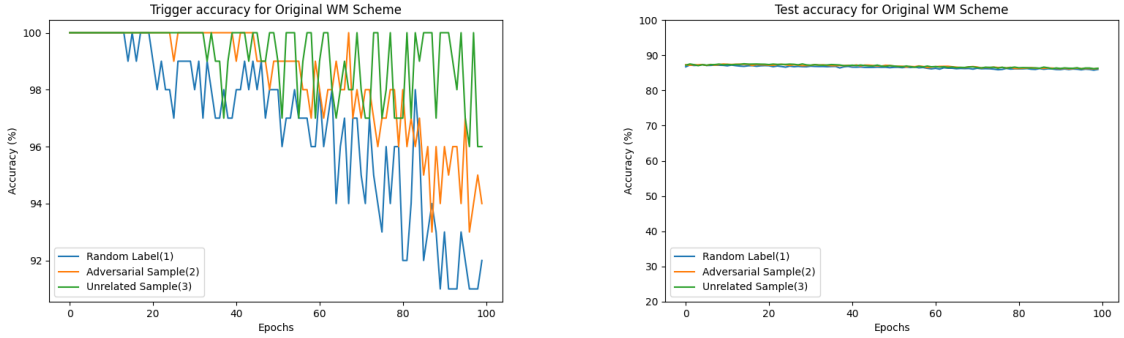

Figure 4.12: Original Backdoor Watermarking Trigger and Test Accuracy respectively for Run 1

Original Backdoor Watermarking Scheme Run 2

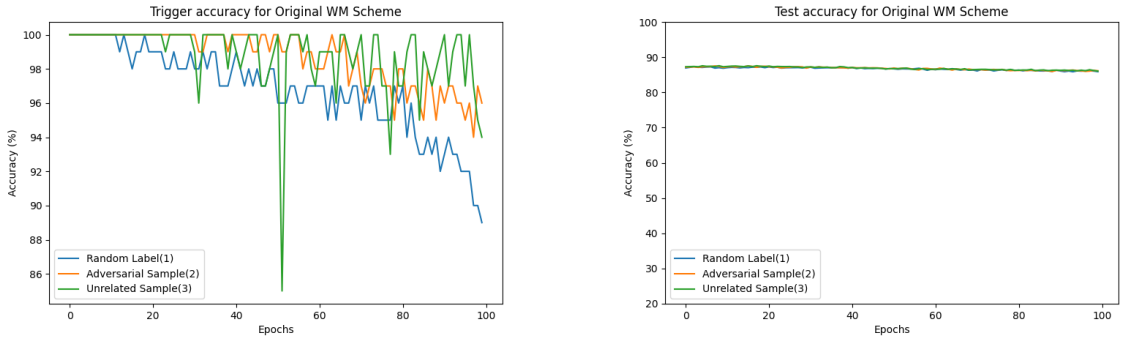

Figure 4.13: Original Backdoor Watermarking Trigger and Test Accuracy respectively for Run 2

Original Backdoor Watermarking Scheme Run 3

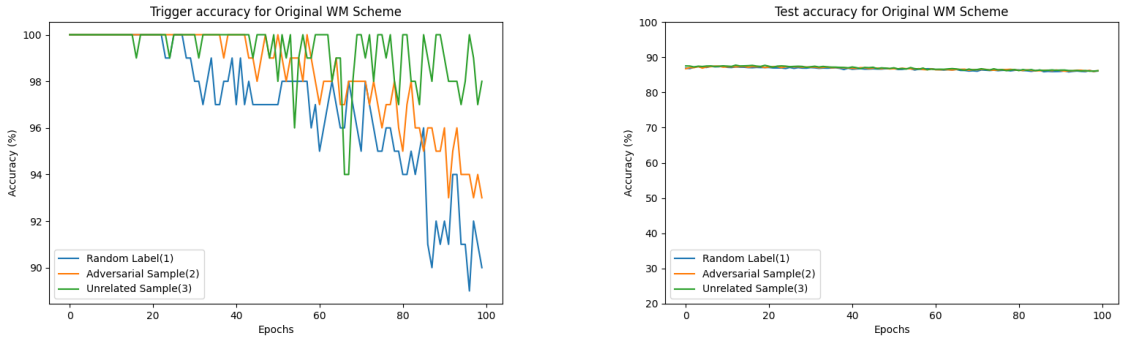

Figure 4.14: Original Backdoor Watermarking Trigger and Test Accuracy respectively for Run 3

4.2.4 Summary of Results for Trigger Type Test

Trigger Type	Random Smoothing	Uniform WM	Original Backdoor
RMS Samples R1	**92%**	47%	89%
Adversarial Samples R1	**96%**	77%	96%
Unrelated Samples R1	**100%**	59%	94%
RMS Samples R2	**94%**	45%	90%
Adversarial Samples R2	**96%**	74%	93%
Unrelated Samples R2	**100%**	82%	98%
RMS Samples R3	**91%**	48%	89%
Adversarial Samples R3	**96%**	75%	92%
Unrelated Samples R3	**100%**	33%	98%

Table 4.3: Tabulated Trigger Set Accuracy after Trigger Set Type Test

Trigger Type	Random Smoothing	Uniform WM	Original Backdoor
RMS Samples	92.3%	56.3%	89.3%
Adversarial Samples	96%	75.3%	93.7%
Unrelated Samples	100%	58%	96.7%

Table 4.4: Mean of Trigger Set Accuracy after Trigger Set Type Test

Based on the results, we can observe that between the Trigger Set of Adversarial Samples vs Randomly Mislabelled Samples (RMS), there is a slight improvement in performance throughout all the different watermarking schemes. However, it is duly noted that the main concept that maintains persistence of watermarks lies within the watermarking schemes itself, and that the Randomized Smoothing Scheme [8] the most robust scheme compared to the rest. This is evident from the Randomized Smoothing scheme maintaining the highest Trigger Set Accuracy (bolded) in every run.

4.3 Incremental Training with Variation in Learning Rates

This experiment was done with different learning rates to observe how much the persistence of watermarks would be affected by learning rates during the Incremental Training. The hyperparameters for the following experiment can be referred to here at Section 3.8.

4.3.1 Incremental Training Learning Rates Test - LR=0.0001

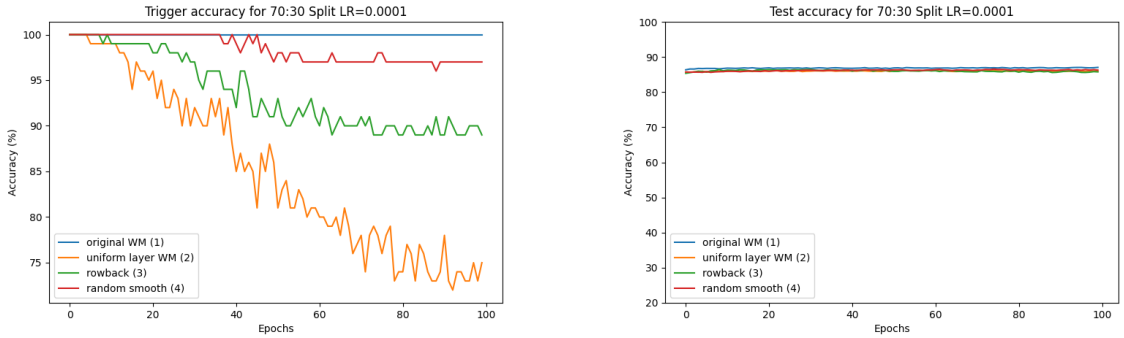

Figure 4.15: Left: LR=0.0001 Trigger Set Accuracy Right: LR=0.0001 Test Set Accuracy

4.3.2 Incremental Training Learning Rates Test - LR=0.0005

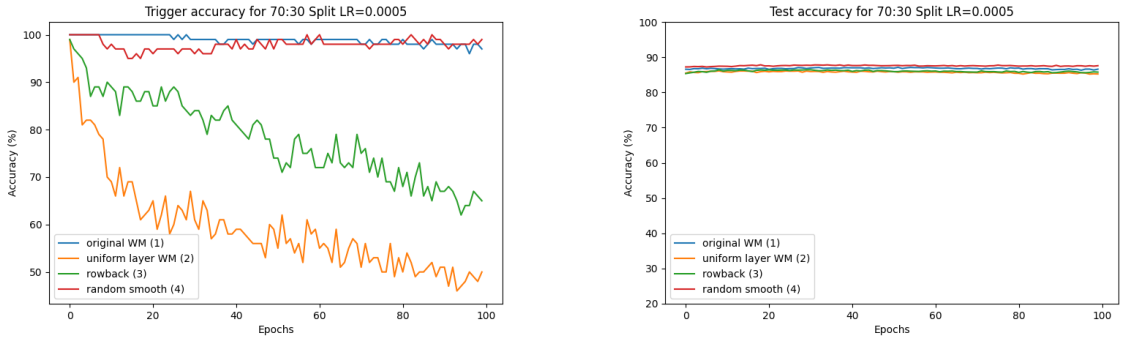

Figure 4.16: Left: LR=0.0005 Trigger Set Accuracy Right: LR=0.0005 Test Set Accuracy

4.3.3 Incremental Training Learning Rates Test - LR=0.001

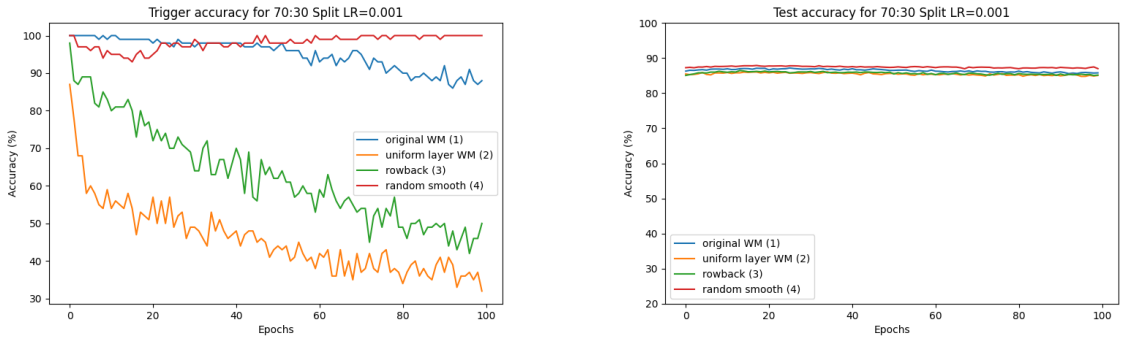

Figure 4.17: Left: LR=0.001 Trigger Set Accuracy Right: LR=0.001 Test Set Accuracy

4.3.4 Incremental Training Learning Rates Test - LR=0.005

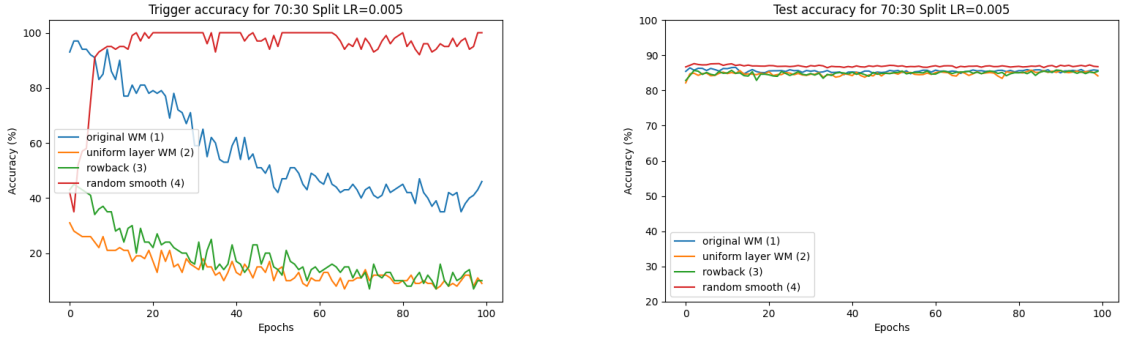

Figure 4.18: Left: LR=0.005 Trigger Set Accuracy Right: LR=0.005 Test Set Accuracy

4.3.5 Incremental Training Learning Rates Test - LR=0.01

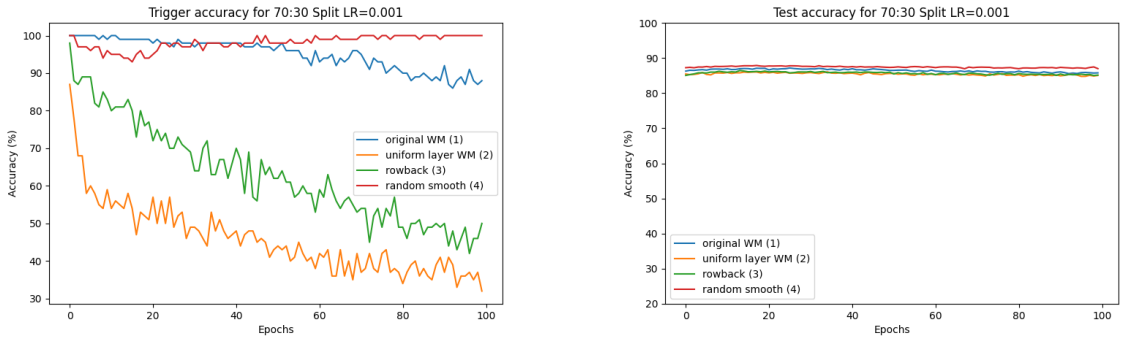

Figure 4.19: Left: LR=0.001 Trigger Set Accuracy Right: LR=0.001 Test Set Accuracy

4.3.6 Incremental Training Learning Rates Test - LR=0.05

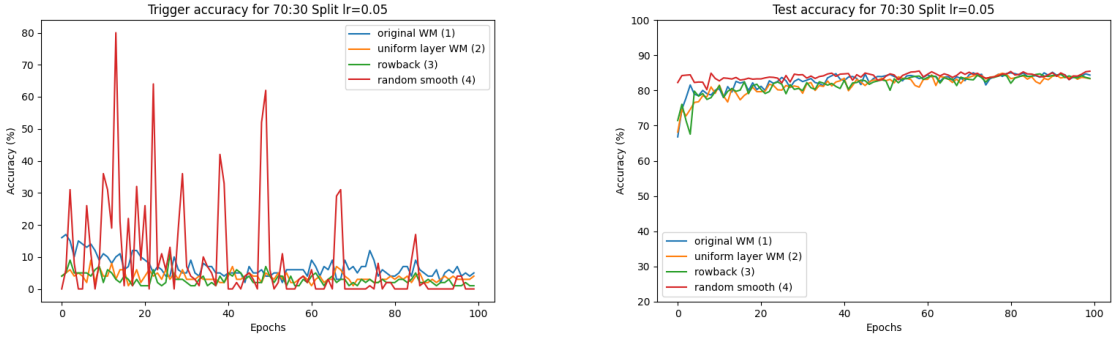

Figure 4.20: Left: LR=0.05 Trigger Set Accuracy Right: LR=0.05 Test Set Accuracy

4.3.7 Summary of Results for Learning Rate Test

Learning Rate	Random Smoothing	Original	ROWBACK	Uniform WM
0.0001	97%	**100%**	89%	75%
0.0005	**99%**	97%	65%	50%
0.001	**100%**	88%	50%	32%
0.005	**100%**	46%	10%	9%
0.01	**97%**	19%	6%	7%
0.05	0%	**4%**	2%	**4%**

Table 4.5: Tabulated Trigger Set Accuracy after Learning Rate Test

We can observe that the general trend for Trigger Set Accuracy is such that the learning rate is inversely proportionate to the persistence of the Trigger Set in each of the schemes, e.g. higher learning rates is detrimental towards Trigger Set persistence. However, the Randomized Smoothing Scheme [8] managed to withhold its persistence of Trigger Set with a high accuracy up to learning rates of 0.01, while all the other schemes have already failed around learning rates of 0.001. This further enhances the credibility of the Randomized Smoothing Scheme. If we focus in on Figure 4.3.6, we can see that the Trigger Set Accuracy has been pretty much destroyed, while the Test Accuracy still maintains at a high level of 80%. This information could prove to be malicious under the wrong hands. Adversaries could intentionally carry out Incremental Training at a learning rate of 0.05 to remove the Trigger Set, while maintaining the original model's classification ability.

Chapter 5

Conclusion

5.1 Conclusion

The experimental work presented here provides one of the first investigations on how Incremental Training would affect a watermark-embedded neural network. In this paper, we focused on 4 main watermarking schemes: ROWBACK [7], Certified Watermarking with Randomized Smoothing [8], Original Backdoor Watermarking Scheme [25] and also a Uniform Watermarking Scheme we used as a control scheme 2.7.3.

The Train Set and Incremental Set used was mainly the CIFAR-10 Dataset, while Trigger Sets were a combination of MNIST Dataset[19], adversarial samples [12] or CIFAR-10 Dataset with Randomly Mislabelled Samples.

The study has shown that persistence of watermarks is largely dependent on the scheme itself, and less on other variables such as Trigger Set Type or learning rates of the neural network models. We also hypothesized that uniform distribution of watermarks has ill effects on Incremental Training.

The findings here would be pivotal in developing a framework for a robust watermarking scheme for neural networks, ultimately preserving Intellectual Property Rights.

5.2 Future Works

5.2.1 Locking of specific layers

The current schemes that involve uniform distribution of watermarks involve locking every layer but one layer during each epoch of training, to simulate uniform distribution [7]. The idea was brilliant but might have been too aggressive, and looking into permanently locking specific layers that are essential in watermark detection **for the end user during Incremental Training** is something that could be considered.

5.2.2 Extending into other areas

The current schemes have only been tested on classification models. A possible enhancement to the scalability of the world of watermarking neural networks could be using it for regression models, forecasting models or even realms of unsupervised learning such as clustering or dimension reduction models.

5.2.3 Experimentation with more datasets

The experiments done in our study were all conducted with the CIFAR-10 Dataset [17]. There are lots of more datasets out there that could be potential candidates in testing the persistence of watermarks through Incremental Training. Possible candidates include CINIC-10 [36], ImageNet [37] or Visual Genome [38].

5.2.4 Experimentation with more networks

The experiments done in our study were all conducted with the ResNet-18 Architecture [34]. Future works could involve expanding the experiments into networks such as the ViT-H/14 [39], µ2Net [40] or EfficientNetV2 [41].

Bibliography

[1] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3):362–386, 2020.

[2] Geoffrey Hinton. Deep Learning—A Technology With the Potential to Transform Health Care. *JAMA*, 320(11):1101–1102, 09 2018.

[3] Daniel W. Otter, Julian R. Medina, and Jugal K. Kalita. A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*, 32(2):604–624, 2021.

[4] Mauro Ribeiro, Katarina Grolinger, and Miriam A.M. Capretz. Mlaas: Machine learning as a service. pages 896–902, 2015.

[5] Ricardo Ribani and Mauricio Marengoni. A survey of transfer learning for convolutional neural networks. pages 47–57, 2019.

[6] Dhaval Chudasama. Importance of intellectual property rights. 4:2021, 01 2022.

[7] Nandish Chattopadhyay and Anupam Chattopadhyay. Rowback: Robust watermarking for neural networks using backdoors. pages 1728–1735, 2021.

[8] Arpit Bansal, Ping-yeh Chiang, Michael Curry, Rajiv Jain, Curtis Wigington, Varun Manjunatha, John P Dickerson, and Tom Goldstein. Certified neural network watermarks with randomized smoothing. 2022.

[9] Huili Chen, Bita Darvish Rouhani, and Farinaz Koushanfar. Blackmarks: Blackbox multibit watermarking for deep neural networks, 2019.

[10] Nandish Chattopadhyay, Chua Sheng Yang Viroy, and Anupam Chattopadhyay. Re-markable: Stealing watermarked neural networks through synthesis. page 46–65, 2020.

[11] Sasha Targ, Diogo Almeida, and Kevin Lyman. Resnet in resnet: Generalizing residual architectures. 2016.

[12] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. 2016.

[13] IBM. Neural networks.

[14] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, jan 2015.

[15] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015.

[16] Poonam Sharma and Akansha Singh. Era of deep neural networks: A review. In *2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–5, 2017.

[17] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[18] Alex Krizhevsky. The cifar-10 dataset.

[19] Christopher J.C. Burges Yann LeCun, Corinna Cortes. The mnist database.

[20] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[21] Yue Li, Hongxia Wang, and Mauro Barni. A survey of deep neural network watermarking techniques. *CoRR*, abs/2103.09274, 2021.

[22] Huili Chen, Bita Darvish Rohani, and Farinaz Koushanfar. Deepmarks: A digital fingerprinting framework for deep neural networks, 2018.

[23] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. Deepsigns: A generic watermarking framework for ip protection of deep learning models, 2018.

[24] Sebastian Szyller, Buse Gul Atli, Samuel Marchal, and N. Asokan. Dawn: Dynamic adversarial watermarking of neural networks, 2019.

[25] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring, 2018.

[26] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. Embedding watermarks into deep neural networks. *CoRR*, abs/1701.04082, 2017.

[27] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2014.

[28] Jeremy M Cohen, Elan Rosenfeld, and J. Zico Kolter. Certified adversarial robustness via randomized smoothing, 2019.

[29] Ayush Gupta. A comprehensive guide on deep learning optimizers.

[30] Nagesh Singh Chauhan. Optimization algorithms in neural networks.

[31] Micheal Lanham. Learn arcore - fundamentals of google arcore.

[32] Simone. Stochastic gradient descent on your microcontroller.

[33] Kiprono Elijah Koech. Cross-entropy loss function.

[34] Datagen. Resnet: The basics and 3 resnet extensions.

[35] Yogesh Balaji, Tom Goldstein, and Judy Hoffman. Instance adaptive adversarial training: Improved accuracy tradeoffs in neural nets, 2019.

[36] Luke N. Darlow, Elliot J. Crowley, Antreas Antoniou, and Amos J. Storkey. Cinic-10 is not imagenet or cifar-10, 2018.

[37] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge, 2014.

[38] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A. Shamma, Michael S. Bernstein, and Fei-Fei Li. Visual genome: Connecting language and vision using crowdsourced dense image annotations, 2016.

[39] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2020.

[40] Andrea Gesmundo and Jeff Dean. An evolutionary approach to dynamic introduction of tasks in large-scale multitask learning systems, 2022.

[41] Mingxing Tan and Quoc V. Le. Efficientnetv2: Smaller models and faster training. 2021.