October 2022

Exploring the Vulnerability of A Neural Tangent Generalization Attack (NTGA) - Generated Unlearnable CIFAR-10 Dataset

Gitte Ost
University of South Florida

Exploring the Vulnerability of A Neural Tangent Generalization Attack (NTGA)-Generated Unlearnable CIFAR-10 Dataset

by

Gitte Ost

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Arts
Department of Mathematics & Statistics
College of Arts and Science
University of South Florida

Major Professor: Kaiqi Xiong, Ph.D.
Lu Lu, Ph.D.
Razvan Teodorescu, Ph.D.
Jiwoong Kim, Ph.D.

Date of Approval:
October 18, 2022

Keywords: Machine Learning/Deep Learning, Federated Learning, Neural Tangent Generalization Attack, Data Augmentation

Dedication

I dedicate this thesis from my heart to my beloved parents, Dirk Ost and Gerlinde Berckmoes, who have always supported me and given me constant moral, emotional and financial support. The dedication also includes my sister Saskia, brother Sybren, and brother-in-law Zahrak, who encouraged me to go further and further, with their humors.

Acknowledgments

First of all, I would like to thank my thesis and research supervisor, Dr. Kaiqi Xiong, who gave me the opportunity to conduct research in the field of machine learning/deep learning and computer security, despite my limited experience with deep learning. It was a pleasure working with Dr. Kaiqi Xiong, whose broad knowledge, rich research experience, and advice helped me a lot during my research exploration and thesis writing.

I would also like to thank Dr. Lu Lu, Dr. Razvan Teodorescu, and Dr. Jiwoong Kim for serving on my thesis committee.

In addition, I would like to thank Dr. Jing Lin, Thushari Happurachi, Dr. Mohamed Routi, and Shujun Zhao, who all aided me in advancing my knowledge and understanding of machine learning and python. I also would like to thank Long Dang for his patience to help me understand federated learning and resolve experimental issues I have encountered, and his feedback on my experiment results.

Moreover, I would also like to thank USF for providing the computing resources of Research Computing available so that I could implemented my research methodology and conduct my experiments to evaluate it.

I like to acknowledge the National Science Foundation (#1620871 and #1620868) for partially supporting this research.

Table of Contents

List of Tables

List of Figures

Abstract

Nowadays, a massive amount of data is generated and stored on servers and clouds from various applications daily. Preventing these data from unauthorized use often becomes necessary and critical in various real-world applications. Many researchers have studied this crucial problem and developed different methods for this purpose. Among them, Neural Tangent Generalization Attack (NTGA) is one of the most efficient methods to make a dataset unlearnable, which means that the dataset is not learnable by machine learning/deep learning methods. That is, the NTGA-generated dataset is protected against unauthorized use. In this thesis, we explore the vulnerability of an NTGA-generated unlearnable CIFAR-10 dataset. Specifically, we show that by employing data augmentation, we can still train a model using the CIFAR-10 dataset generated by NTGA in a centralized learning environment. That is, we experimentally prove that the NTGA-generated unlearnable CIFAR-10 dataset is vulnerable or learnable. To reduce the execution time of learning from the dataset, we further study the vulnerability of the unlearnable dataset in a decentralized learning (i.e., federated learning) environment. Our experiments demonstrate the efficiency of the proposed decentralized method. After all, we reduce the execution time for learning from the unlearnable dataset when using decentralized learning compared to centralized learning.

Chapter 1: Introduction

In the modern world, hundreds of exabytes of data are generated and stored on servers or clouds daily, waiting to be used by data scientists for machine learning and other applications. Therefore, the concern for data privacy and security has significantly increased, and as a response, new laws are being introduced to protect data. An example of this it the General Data Protection Regulation (GDPR), which made the data privacy laws in Europe consistent in 2018 [10]. One of the rules of this regulation is that the data owners have the right to object to having their data used by a third-party [9]. Around the same time, California also introduced a new law called the California Consumer Privacy Act (CCPA) [23]. Among the rules of this new law, California residents have the right to say that businesses are not allowed to sell their personal information [23]. Thus, other organizations cannot use personal information unless the California resident consents. The problem of privacy protection regarding personal data is not new. Hospitals, for example, store patient data to help them detect diseases and treat patients better; however, this information should also be kept private; therefore, the Health Insurance Portability and Accountability Act (HIPAA) was introduced in 1996 [13]. The new topic of discussion now revolves around the proper protection of data in accordance with the law. One way that is being explored is making data unlearnable to ensure that if someone obtains access to data that they are not

allowed to use, they will indeed be unable to use it.

Among the many existing studies on this issue, Neural Tangent Generalization Attack (NTGA), proposed by Yuan and Wu [28], is one of the most efficient methods to make a dataset unlearnable. In this thesis, we will explore the vulnerability of the NTGA-generated unlearnable dataset. Specifically, we will focus on studying the unlearnable CIFAR-10 dataset generated by the NTGA. Based on the results of our experiments, we can prove that the considered unlearnable dataset is converted into a learnable dataset by applying some simple data augmentation techniques. Hence, we showed that the CIFAR-10 NTGA-generated dataset is vulnerable. Furthermore, we propose a decentralized approach to train on the dataset to reduce the execution time of exploring the vulnerability of the NTGA-generated unlearnable dataset. When this approach is utilized, only the model information is transferred among organizations rather than the dataset itself. Our proposed decentralized approach uses Federated learning (FL) for decentralized training. While FL does provide some security and privacy advantages compared to centralized training due to its decentralized nature, there are still concerns about security/privacy associated with FL [21]. IBM has created a framework that includes libraries to address some of these concerns [18]. In this thesis, we will demonstrate that when using the CIFAR-10 NTGA-generated dataset on the IBM FL framework, the machine learning model can still learn from the dataset, even though it is supposed to be unlearnable. Thus, the NTGA does not protect the data in centralized nor decentralized learning environments.

1.1 Motivation

The need to secure data can be found in many real-life applications, where the use of private data is necessary for the proper functioning of the products or services provided. As is well-known, the use of smartphones has significantly increased over the last decade. With this increased usage of smartphones, there is also an increase in the need for data security. For example, when the next-word suggestion is on for the Google Keyboard, it will utilize the private data stored on that device to suggest which word to use next. Due to the privacy issues, FL is implemented [14].

The Google Keyboard application is a new phenomenon. Hospitals, on the other hand, have struggled for years with the patient data dilemma. On the one hand, the data is very sensitive and should not be available to anyone other than the necessary healthcare providers. On the other hand, that data is crucial for treating patients and researching causes, connections, treatment methods, and lifestyle. At this moment, hospitals need to be in accordance with HIPAA, which means that keeping the data secure is a crucial part of hospital management [13].

Another example to justify why data privacy is necessary is the energy usage data of households. Such data are typically stored by their energy providers since this information is essential to the provider's daily projections. In case this information falls into a bad guy's hands, the person could, with a high degree of certainty, predict when people will be in their homes or not.

In various industries, there is a need to use real-time data with machine learning, and it is, therefore, vital that the model is trained on the data in a time-efficient way. Self-driving cars is one of those industrial applications, getting much attention nowadays in real-time machine learning. Initially, the car learns from an enormous amount of data it collects to guide self-driving. For example, a self-driving car needs to learn traffic signs; it also needs to know how to react to obstacles and how to behave in traffic. To do these tasks as well as possible, the car will train on collected data in real-time, which means that the information obtained from real-time situations is being used in training for improving a car's self-driving performance. One of the real-time training tasks that has been introduced is to predict the steering angle of the car [12]. It is clear that not only the steering angle is essential, but that the reaction speed of the car is as important. Thus, training time is critical to avoid accidents.

The above examples have demonstrated the need to use private data and the need to protect against unauthorized use of the data. It is, therefore, essential that researchers continuously look for methods to protect these data. Nonetheless, it is even more important to investigate the reliability of these solutions. The main objective of this thesis is to explore the vulnerability of an NTGA-generated unlearnable CIFAR-10 dataset. This is important to help in secure real-world data in various applications.

1.2 Research Problems, Challenges, and Methodology

The research objective of this thesis is to explore the vulnerability of a CIFAR-10 dataset generated by NTGA, which results in the following two research questions: (1) Is an NTGA-generated CIFAR-10 dataset vulnerable or learnable by machine learning/deep learning methods? and (2) if yes, how can we find the vulnerability of a CIFAR-10 dataset generated by NTGA faster? “Learnable" means here that a deep neural network (DNN) could first train on given input and then output a model such that it can be used to make a prediction. When using the CIFAR-10 dataset, for example, such a model will be able to accurately predict which of the ten classes a new image belongs to. Thus, when we say “learnable," it means that the prediction of a model is correct most of the time. In this thesis, we will say the dataset is learnable if the model can achieve a test accuracy of at least 85%.

To answer our two research questions, we will start with utilizing a centralized learning environment and data augmentation on the NTGA-generated CIFAR-10 dataset to get a test accuracy of at least 85%. This section is part of the manuscript “Learn From the Unlearnable: Making Unlearnable Examples Learnable Through Data Augmentation”. We will use the same minimum test accuracy of 85% as set in the above-mentioned manuscript [17]. The second part of the research problem is to find out how we can improve the methodology and reduce the execution time of the training. We will again utilize data augmentation on the NTGA-generated CIFAR-10 dataset, but now in a decentralized learning environment.

We aim to get a test accuracy of at least 85% again. Hence, we will demonstrate that the dataset generated by the NTGA is vulnerable when used in both centralized and decentralized learning environments. Moreover, we will demonstrate through multi-party experiments that by using FL and thus increasing the number of parties, we can achieve the same test accuracy faster. As far as we know, there is currently no scientific literature in which the vulnerability of presumed protected data by NTGA on the IBM FL environment is tested.

As is well-known, a DNN model often has a complicated architecture, where there is a large number of parameters associated with the model. The complexity of the model makes it very difficult to conduct theoretical analysis on this model. Instead, we will study the above two questions by developing an experimental methodology. Such experiments will be conducted on real-world computing resources rather than simulations, which raises a computing resource challenge in this research. Another challenge is that when we increase the number of parties in the FL setting, the number of images in our dataset will decrease, likely decreasing our test accuracy. This would mean that we will have to utilize more data augmentation techniques with an increasing number of parties which would increase the execution time of the model training. These are the challenges we face while conducting our research experiments.

1.3 Contributions

To prove that the NTGA-protected dataset is vulnerable, we have developed a methodology to achieve a test accuracy of at least 85% in centralized learning. Moreover, in central-

ized learning, we are able to achieve a test accuracy of 87.04%. Thus, we demonstrate that a CIFAR-10 NTGA-generated dataset is still learnable; this was part of the manuscript "Learn From the Unlearnable: Making Unlearnable Examples Learnable Through Data Augmentation" [17]. In addition, we also show that our methodology, with slight modifications, can be used on decentralized learning to achieve a test accuracy of at least 85% while reducing the execution time of the model training. Thus, we show that a CIFAR-10 dataset generated by NTGA is vulnerable in a centralized and decentralized learning environment and that using a decentralized learning environment is faster than using a centralized learning environment.

1.4 Thesis Organization

The subsequent chapters are divided as follows. First, we go over some background information. We explain what the python code we used in Open Source Computer Vision (OpenCV) and Pillow do in our images, give machine learning/deep learning models and related information, and describe what NTGA and FL are. In the next chapter, we include our proposed methodology. The fourth chapter will include our experiments and evaluations. In our last chapter, we conclude our findings and future work.

Chapter 2: Background Information

As one of the deep learning models, a DNN model requires a lot of data to learn. A common way to increase the data is by doing data augmentation. One of the ways to do data augmentation is to make use of the built-in Keras function: ImageDataGenerator. Another way is to utilize the OpenCV and Pillow libraries in Python. For our experiments we used models that are based on models created by the Visual Geometry Group of the University of Oxford [25], called VGG models. The models will implement a loss function, an optimizer, and activation functions in training. Our dataset is an NTGA-generated dataset, so in this section, we will first describe how the NTGA works. We also used FL in our experiments, so we will end this section by going over how FL works and, more precisely, how IBM FL works.

2.1 OpenCV Data Augmentation

In this thesis, we used classes from the OpenCV library of Python to do data augmentation. In our experiments, we only used the color channels from OpenCV; therefore, we will only go over color channels in this section. Even though, more classes in OpenCV can do data augmentation, for example, Gaussian blur, dilate, and erode.

2.1.1 Color Channels [1]

Every digital image comprises many pixels, and a pixel has a specific value corresponding to the color intensity. When we have a grey image, the image will have a single channel, and the pixel has values between 0-255 [27], where a value of 0 gives a black image segment and a value of 255 a white image segment. The values between 0 and 255 give shades of grey; the lower the value, the darker the grey. In our case, the images are normalized; therefore, the pixel values range from 0 to 1. Colored images, on the other hand, use three channels, each representing the red, green, and blue color channels [27]. The pixel will therefore have three different values; where the first value represents information on the intensity of the red color, the second value then contains information on the intensity of the green color, and the third value provides information on the intensity of the blue color. Our images have the order: red, green, and blue, but it is also possible that the information is stored in the order: blue, green, and red. So, when we want to change the colors of an image, we need to adjust the color values of the pixels that make up the image. We decided to use color channels to do this. When creating a color channel, the images become grey since they will only have one channel now. We use cv2.merge((b,b,b)) in this research, where b stands for the blue color channel. We see that each pixel value is replaced by the pixel value of the blue channel. Since a pixel of (255,255,255) gives a white color and (0,0,0) gives a black color, we can see that for a higher value of the blue channel, the pixel will get closer to white and that for a lower value for the blue channel the pixel will get closer to black. The same

[1] This section is a modification from a section of the manuscript [17]

Figure 2.1: Example of the Unlearnable CIFAR-10 Image and Color Channels.

reasoning happens with the green and the red channels. Figure 2.1 shows the color channels for one CIFAR-10 image generated by NTGA.

2.2 Pillow Data Augmentation

When using the pillow library, we focused on implementing the enhancer module. We used all the available classes, namely, brightness, contrast, color, and sharpness. The python code we are going to utilize has two inputs: an image and a factor for every enhancer used. The first step is to create a new variable. The variable is often called an enhancer, which gives the type of enhancement that is applied and has as input the image. For example, enhancer = ImageEnhance.Brightness(image). The new image is created in the next step by using the command enhancer.enhance(factor). The enhancer.enhance will blend two images; the output will be the following [7]:

$$\text{(second image)} * \text{(1-factor)} + \text{(original image)} * \text{factor.} \tag{2.1}$$

Figure 2.2: Example of the Unlearnable CIFAR-10 Images with a Brightness Factor of 1.5.

The second image is different for each class. As we can see from equation 2.1, the original image will be returned if the factor is one. Figures 2.2-2.5 gives a visualization of the enhanced images of the unlearnable CIFAR-10 dataset generated by NTGA.

2.2.1 Brightness Enhancer

As the name suggests, this is implemented to change the brightness of an image. According to the source code [8], we can see that for this class, the second image is a black image. Therefore, when the factor is zero, it will return a black image. In our experiments, the factor was 1.5, which will brighten the images.

2.2.2 Contrast Enhancer

In this case, the contrast of the image will be adjusted. When using the contrast class, the second image will be created by using a specific value for the color of the original image, and this value is the mean of the original image plus 0.5 [8]. If the factor would be zero this time, we will receive a gray image. In our experiments, we used a contrast factor of 1.5, which increases the contrast of the image. If we would use a factor of 0.5, the image becomes more grayish.

Figure 2.3: Example of the Unlearnable CIFAR-10 Images with a Contrast Factor of 1.5

Figure 2.4: Example of the Unlearnable CIFAR-10 Images with a Color Factor of 1.5.

2.2.3 Color Enhancer

This class will affect the color output of an image. For this reason, the second image will be created by turning the original image into a monochrome image [8]. Hence, when the factor is zero a monochrome image will be returned. In our experiments, we increased the color of the images by using a factor of 1.5.

2.2.4 Sharpness Enhancer

When we want to change the sharpness of an image, we can make use of this class. The second image in equation 2.1 will be created by filtering the original image with a smooth filter [8]. In this case, a factor of zero will return a blurred image. In our experiments, we used a factor of two, which returns a sharpened image.

Figure 2.5: Example of the Unlearnable CIFAR-10 Images with a Sharpness Factor of 2.

2.3 Model Background

In our experiments, we used a VGG16 model and a VGG19 model, which are created by the Visual Geometry Group of the University of Oxford [25]; the difference between the two models is the number of layers. While Table 2.1 gives the architecture of a VGG16 model, Table 2.2 gives the architecture of a VGG19 model. Both models have an input shape of 32 x 32 x 3 to fit the CIFAR-10 dataset.

Layer	Output shape	Activation function
Input	32 x 32 x 3	
2 x Convolutional	32 x 32 x 64	ReLU
Max pool	16 x 16 x 64	
2 x Convolutional	16 x 16 x 128	ReLu
Max pool	8 x 8 x 128	
3 x Convolutional	8 x 8 x 256	ReLU
Max pool	4 x 4 x 256	
3 x Convolutional	4 x 4 x 512	ReLU
Max pool	2 x 2 x 512	
3 x Convolutional	2 x 2 x 512	ReLU
Max pool	1 x 1 x 512	

Table 2.1: The VGG16 Model Architecture with an Adjusted Input Size of 32x32x3 To Fit the CIFAR-10 Dataset. Source: Adapted from [17]

Layer	Output shape	Activation function
Input	32 x 32 x 3	
2 x Convolutional	32 x 32 x 64	ReLU
Max pool	16 x 16 x 64	
2 x Convolutional	16 x 16 x 128	ReLu
Max pool	8 x 8 x 128	
4 x Convolutional	8 x 8 x 256	ReLU
Max pool	4 x 4 x 256	
4 x Convolutional	4 x 4 x 512	ReLU
Max pool	2 x 2 x 512	
4 x Convolutional	2 x 2 x 512	ReLU
Max pool	1 x 1 x 512	

Table 2.2: The VGG19 Model Architecture with an Adjusted Input Size of 32x32x3 To Fit the CIFAR-10 Dataset. Source: Adapted from [17]

During the model training, a loss function will be implemented to determine how the model performs. The aim is to get model weights that give the lowest possible loss function. There are multiple loss functions that can be used while training, for example, the binary cross-entropy and the categorical cross-entropy. The binary cross-entropy is employed when working with a binary classification problem, i.e., the actual y value is one or zero. Nevertheless, since CIFAR-10 has ten classes, we will use the categorical cross-entropy loss in our experiments. The cross-entropy loss is defined as follows:

$$L_{CE} = -\sum_{i=1}^{n} y_i log(p_i) \tag{2.2}$$

where y_i indicates if the image belongs to the i-th class, p_i is the probability that the image belongs to the i-th class, and n is the total number of classes.

The categorical cross-entropy loss is employed when the true labels are presented as

one-hot encoding. For example, if there are three possible labels and the true label is number three, then it is represented as [0, 0, 1]. The sparse cross-entropy loss is employed when the true labels are presented as integers.

We will utilize an optimizer to reduce the overall loss and increase a test accuracy. Keras [6] has a few built-in optimizers that we can use. In our thesis, we used the Stochastic Gradient Descent (SGD), which is based on the gradient descent (GD). The GD is defined as:

$$\theta_{k+1} = \theta_k - \eta * \nabla_\theta \mathcal{L}(\theta_k) \tag{2.3}$$

where η is the learning rate, θ the model weights, and $\mathcal{L}(\theta_k)$ is the loss function we want to minimize.

The GD with momentum is defined as:

$$\begin{aligned} \text{velocity}_{k+1} &= \text{momentum} * \text{velocity}_k - \eta * \nabla_\theta \mathcal{L}(\theta_k) \\ \theta_{k+1} &= \theta_k + \text{velocity}_{k+1} \end{aligned} \tag{2.4}$$

The SGD uses only a small number of training examples instead of all the training examples for each step. Small changes are made to the model weights each time in order to bring the loss function to a minimum.

Using activation functions in neural networks helps to learn complex patterns from data and adds the ability to generate nonlinear mappings [22]. There are multiple activation functions available in Keras [5]. We used the Rectified Linear Unit (ReLU) and Softmax as activation functions for our experiments. The Softmax activation function will give us the

probability of each class for each image; therefore, we will implement it in our last layer.

$$\text{ReLU: } f(x) = \max(0, x) \tag{2.5}$$

$$\text{Softmax: } s(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{2.6}$$

where $\mathbf{x}$ is an input vector, and n is the number of classes.

In our experiments, we used dropout layers to decrease the overfitting of our model. A DNN uses feed forward, which can be described as follows [26]:

$$\begin{aligned} x_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \mathbf{y}^l + b_i^{(l+1)} \\ y_i^{(l+1)} &= f(x_i^{(l+1)}) \end{aligned} \tag{2.7}$$

where $l \in 1, .., L$ and L is the number of hidden layers in the DNN. The vector $\mathbf{x}^{(l+1)}$ denotes the inputs into layer $l+1$, and the vector $\mathbf{y}^{(l+1)}$ denotes the output of the layer $l+1$. The model weights are defined by $\mathbf{W}^{(l+1)}$ for layer $l+1$. Moreover, the biases are defined by $\mathbf{b}^{(l+1)}$ for layer $l+1$. The activation function is defined as $f(x)$.

When we included a dropout layer, the feed forward becomes [26]:

$$\begin{aligned} r_j^{(l)} &\sim \text{Bernoulli}(1-p) \\ \tilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)} \\ x_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \tilde{\mathbf{y}}^{l} + b_i^{(l+1)} \\ y_i^{(l+1)} &= f(x_i^{(l+1)}) \end{aligned} \tag{2.8}$$

When using the dropout layer in Keras, Keras will scale the inputs that are not set to zero by $\frac{1}{1-p}$ [4], where p stands for the probability that the random variable is equal to zero.

We also made use of the pooled standard deviation to determine how many trials we would do for each experiment. The pooled standard deviation for proportions is defined as follows:

$$\begin{aligned} \hat{p} &= \frac{x_1 + x_2}{n_1 + n_2} \\ \text{Standard Deviation} &= \sqrt{\hat{p}(1-\hat{p})\left(\frac{1}{n_1} + \frac{1}{n_2}\right)} \end{aligned} \tag{2.9}$$

where x_1, x_2 are found by using $p_i = \frac{x_i}{n_i}$ for $i = 1, 2$. In our case, n_i is the number of images in the test set, and p_i is the proportion of the images that the model predicts correctly.

2.4 Neural Tangent Generalization Attack

This thesis uses the CIFAR-10 unlearnable dataset that was generated by using the NTGA [28] on the clean CIFAR-10 dataset. The NTGA created by Yuan and Wu [28] implements the neural tangent kernel (NTK) to transform the clean label generalization attack into a single objective function.

2.4.1 Clean Label Generalization Attack

A generalization attack is used when the attacker wants to ensure that a trained network lacks generalizability [28], and therefore gives a low-test accuracy. The NTGA is made with the intention that legitimate users will be able to use the dataset; therefore, they focus on a clean label attack [28].

We can write the clean label generalization attack as the following bi-level optimization problem [28]:

$$\begin{aligned} &\arg\max_{P\in\tau} \mathcal{L}(f(X^m;\theta^*),Y^m) \\ &\text{subject to } \theta^* \in \arg\min_{\theta} \mathcal{L}(f(X^n+P;\theta),Y^n) \end{aligned} \tag{2.10}$$

where we denote the training set: $\mathbb{D} = (X^n \in \mathbb{R}^{n\times d}, Y^n \in \mathbb{R}^{n\times c})$ and the validation set $\mathbb{V} = (X^m, Y^m)$. Where n and m denote the number of examples for the training set and the validation set of the models, respectively. The dimensions of the training input are denoted by d, and the dimensions of the labels (the training output) by c. P is the perturbations added to the training set, $\mathcal{L}$ is the loss function between the predictions and the actual values, τ is a constraint on P, and $f(\cdot;\theta)$ is the model that is parameterized by θ [28].

2.4.2 Neural Tangent Kernel

It has been shown that the Gaussian Process (GP) can be used to estimate the distribution of a class of wide deep neural networks during training [2], [15], and [16]. The NTK is a kernel that becomes constant during training in the infinite-width limit at initialization [15]. Let $f(\cdot;\theta^{(0)})$ be the model that is parameterized by $\theta^{(0)}$, then we can define the NTK as follows [15]:

$$k(x^i,x^j) = \nabla_\theta f(x^i;\theta^{(0)})\nabla_\theta f(x^j;\theta^{(0)})^T \tag{2.11}$$

where x^i, x^j are two data points.

Lee et al. [16] showed that as the width grows of a neural network it can be approximated by its first-order Taylor expansion with respect to its initial parameters. The gradient descent minimizes the mean square error (MSE), and we write the learning rate of the MSE as η. Then, we can see that at time step t of the gradient descent, we can write the mean of the GP denote by $\bar{f}$ as follows [28]:

$$\bar{f}(X^m; K^{m,n}, K^{n,n}, Y^n, t) = K^{m,n}(K^{n,n})^{-1}(I - e^{-\eta K^{n,n} t})Y^n \tag{2.12}$$

where $K^{m,n}_{i,j} = k(x^i \in \mathbb{V}, x^j \in \mathbb{D})$, $K^{n,n}_{i,j} = k(x^i \in \mathbb{D}, x^j \in \mathbb{D})$, which shows that the predictions of $\bar{f}$ can be written in a closed form, hence, it eliminates the need to find θ^*. Therefore, equation 2.10 can be written in the following way:

$$\arg\max_{P \in \tau} \mathcal{L}(\bar{f}(X^m; \hat{K}^{m,n}, \hat{K}^{n,n}, Y^n, t), Y^m) \tag{2.13}$$

where $\hat{K}^{m,n}, \hat{K}^{n,n}$ are the kernel matrices built on the poison data set hence, $\hat{K}^{n,n}_{i,j} = k(X^n_{i,:} + P_{i,:}, X^n_{j,:} + P_{j,:})$ and $\hat{K}^{m,n}_{i,j} = k(X^m_{i,:}, X^n_{j,:} + P_{j,:})$ [28]. Now the project gradient descent (PGD) on the negative loss function can be applied to solve equation 2.13.

The PGD is a variant of the GD. The latter can only be used when solving an optimization problem without a constraint; when there is a constraint, as is the case for the NTGA, the PGD is used. The NTGA uses the multi-step variant of the fast gradient sign method to create the P [28][19].

Now we can write:

$$P_{k+1} = \text{Project}(P_k + \epsilon * \text{sign}\nabla_P(\mathcal{L}(\bar{f}(X^m; \hat{K}^{m,n}, \hat{K}^{n,n}, Y^n, t), Y^m)); \tau(\epsilon))) \quad (2.14)$$

where $\epsilon = \frac{8}{255}$, which was set by Yuan and Wu [28]. This means that P will be projected on $\tau(\epsilon)$ if P is greater than ϵ [28].

2.5 IBM Federated Learning Framework

In our experiments we decided to use FL to implement decentralized learning.

2.5.1 Federated Learning

As indicated earlier, we will also investigate the vulnerability of the NTGA-generated unlearnable dataset by using decentralized training. During this research we will also investigate whether decentralized training is faster than centralized training and, therefore, more efficient. The main difference between centralized and decentralized training is that the research data are kept at their locations, and only the model parameters are shared. This allows large amounts of data from different locations to be trained in a secure and fast manner without data transfer, and due to the larger amount of data from different locations, better results can also be obtained. Decentralized training, therefore, offers both advantages in terms of data privacy preservation and performance. However, in our case, we did not increase the amount of data compared to centralized learning, which permits us to understand how FL, a decentralized learning framework, can help us explore the vulnerability of the NTGA-generated unlearnable dataset in terms of training time.

There are different applications where FL has been introduced. One of the applications is to implement FL in the banking world. When a bank makes decisions on, for example, small and micro enterprise (SME) loans, it is essential to have as much data as possible. For instance, WeBank created an FL application called FedRiskCtrl [1]. It is designed in such a way that it is possible to extract useful information from data coming from different sources without the need to have this data centralized during the training process.

As we mentioned in the introduction, FL has also been implemented by Google to secure the data needed for getting the next word suggestion on the Google keyboard [14]. Google's researchers use FL since the next word suggestion needs a lot of different messages from different people, so this should stay private.

The goal of implementing the FL framework is to train a model among different parties that do not share their private data. The empirical risk minimization problem that needs to be solved is the following [20]:

$$\min_{\theta \in \mathbb{R}^d} F(\theta) \text{ where } F(\theta) = \sum_{p=1}^{P} \frac{n_p}{n} F_p(\theta) \text{ where } F_p(\theta) = \frac{1}{n_p} \sum_{(x,y) \in \mathcal{D}_i} \mathcal{L}_i(x, y; \theta), \tag{2.15}$$

where d is the dimension of the machine learning model, P is the number of parties, $\mathcal{D}_i$ is the data that belongs to party i, and $n_p = |\mathcal{D}_p|$. The total number of examples is denoted by n, and θ are the model parameters that needs to be learned. The function $\mathcal{L}_i(x, y; \theta)$ denotes the loss function that belongs with the data pair (x_i, y_i) and the model with parameters θ [20].

Since the data used in FL are not stored centrally, each party will not have information

about a dataset from a different party. This increases the data security and privacy of each party. As said before, there are still some privacy challenges when using FL. In addition, implementing FL will most likely reduce the time it takes to train a model since the dataset is split over different parties and the training time for each party is most likely less on a smaller dataset. This is one of the main reasons that we used FL to reduce our training time while demonstrating that the CIFAR-10 NTGA-generated dataset is vulnerable.

2.5.2 IBM Federated Learning

To consider the privacy and security challenges in using the existing machine learning library, IBM researchers created the IBM federated learning framework [18]. In this thesis, we used this framework to create a model that can learn from the unlearnable CIFAR-10 dataset generated by NTGA. The IBM FL framework has been adapted by Dang [11] to be used on the computing resources of the Research Computing (RC) [24], which the University of South Florida provides. The IBM FL framework uses an aggregator to communicate with all parties, and each party has only knowledge of its own dataset.

To solve equation 2.15, the aggregator and parties use an FL algorithm that is visualized in Figure 2.6. As always, we want to train a model by finding the parameters (weights) that gives the minimal loss. As noted, the training is done in a decentralized situation, where we have p number of parties. Furthermore, each party has a dataset D_i such that the total dataset is $\sum_{i=1}^{p} D_i = D$. Ludwig et al. [18] described the federated learning algorithm as follows. In the first step of the FL algorithm, the aggregator generates the model informa-

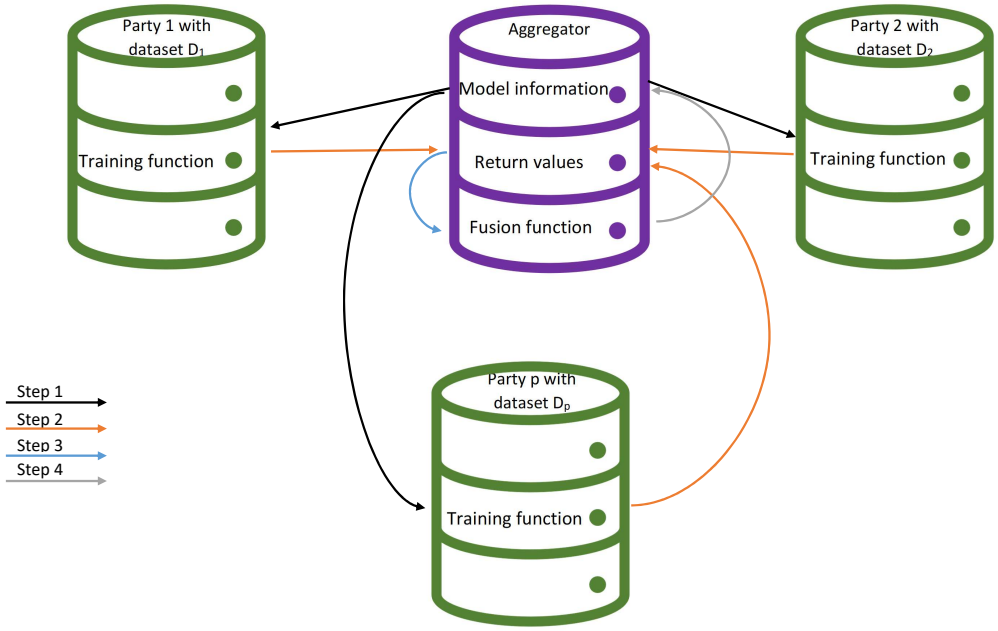


Figure 2.6: The Federated Learning Algorithm.

tion, for example, the model weights. Then, this model information is sent to each party. At this step, each party will use this model information and its dataset as the inputs for its local training function, which will return the model updates, and all parties' returns are sent back to the aggregator. At that time the aggregator will collect all the returned information together. Moreover, the aggregator will use its fusion function to generate a new model, based on the updated model information. These steps will keep being repeated until the manually set number of rounds has been reached.

The IBM FL framework is designed for an easy setup while supporting the federated learning algorithm [18]. To address the privacy concerns, several multi-party computations (SMC) are included in the IBM FL framework [18].

As said before, the IBM FL framework works with an aggregator and parties, each with different components.

The role of the aggregator is to synchronize the FL process. For this reason, we need to execute the fusion function and preserve the meta-data of the FL process. This means that we need to keep track of the registered parties, their current status, all the additional information that is logged, and the aggregated models [18].

In order for us to use the IBM FL framework, we first need to define three configuration files: one for the aggregator, one for the parties, and one for the runner. The runner configuration file will have the important information that is given to the aggregator and parties files as well. For example, the IP address will be manually inserted in the runner configuration file, and it will then be used in the corresponding aggregator and party configuration files. It will also contain the number of rounds and the number of epochs.

The aggregator's configuration file contains the necessary information for the three components that the aggregator needs to function. These components are the connection, the fusion handler, and the protocol handler [18]. The connection component has information about the aggregator's IP address and ports. The protocol handler defines how the messages between the aggregator and the parties are exchanged. It uses the connection component for

information about the connection. Lastly, the fusion handler will acquire each party's model updates and determine how this is combined to generate the new model information. The fusion handler will also reference the model architecture. In addition to the components, the aggregator configuration file also has hyperparameters that are defined. This will include a learning rate and a batch size.

The configuration file of the parties exists of the three components that the parties need to function correctly. These components are the connection, the party protocol handler, and the local training handler [18]. For the first one, the connection, both the IP address and information of the aggregator and the party are needed to connect the two. The communication is again handled by the party protocol handler. The third component is the local training handler, which will decide which training function is used for each party. The local training handler generates the model updates that will be sent to the aggregator. It will need information about what model is being trained, and how the data should be processed. The model information is, for example, whether a TensorFlow model or a PyTorch model is used. It will also need to know the architecture of the model. For the process of the data, a data handler is used.

The process of the federated learning in the IBM FL framework goes as follows. First, the user fills in the information about their IP addresses and model hyperparameters in the runner configuration file. Then, the runner configuration file reads this information and fills it in the configuration files for the parties and the aggregator. If needed, the learning rate

and the batch size will be updated in the aggregator file. The first one to register will be the aggregator since the aggregator communicates with the parties and keeps the logger's information. After the aggregator has successfully started, the parties will register one by one. For the parties to register successfully, they will need to process their data and require the model information. When all the parties are registered, the aggregator will send out a message to start training. After training, the parties send the information about the weight of the last model. Then, the aggregator will use the fusion handler to combine the weights from each party. Furthermore, the updated weights will be sent back to the parties so that the training starts again. These steps will repeat until the number of rounds has been met. At that moment, the aggregator will let the parties know to stop.

When using FL, we also want to see if the test set influences the test accuracy or if the test accuracy is changed due to some sample errors. For this reason, we use the Chi-square test on the test accuracy, where our null hypothesis would be that the distribution over the parties is the same. Moreover, our alternative hypothesis is that the distribution over the parties is not the same. The Chi-square test has two assumptions that need to be met before it can be used. First, the data should be selected at random. Second, each expected count should be at least five. The Chi-square test is defined as followed:

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i} \tag{2.16}$$

where O_i is the observed value and E_i is the expected value.

Chapter 3: Methodology

In this chapter, we will present the methodology we have used in conducting our experiments. Our methodology consists of two major parts, namely, a model training via central learning which is shown in Figure 3.1 and via decentralized learning which is shown in figure Figure 3.2. First, we started training the model using centralized learning. To achieve our target test accuracy of 85% or higher, we applied the following approach. We train our full model every time.

- First, we inserted the dataset generated by the NTGA and included all the libraries that we needed, i.e. pandas, cv2, numpy, etc.
- Subsequently, we inserted a pre-trained VGG19 model with ImageNet weights.

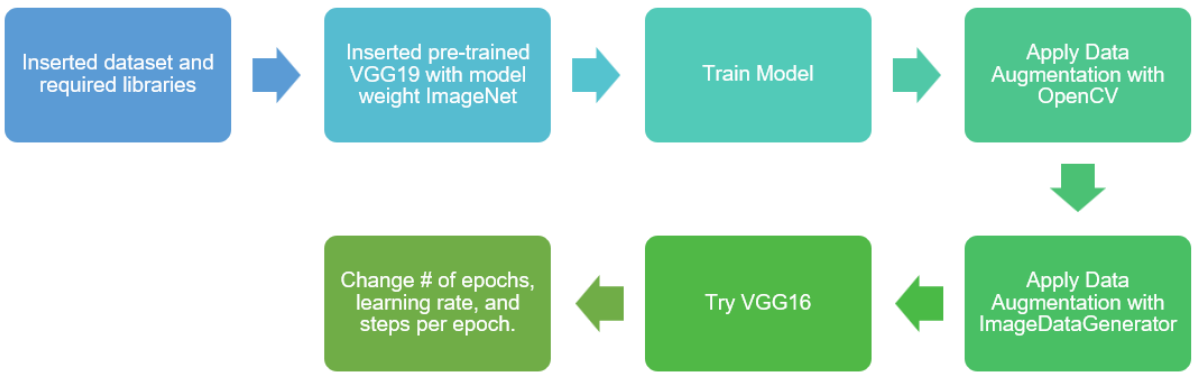


Figure 3.1: Methodology Used During Centralized Learning.

We also added five fully connected layers (FC) of 1024, 512, 256, 128, 10 units, respectively. The first four FC layers had ReLU as an activation function, and the last layer had Softmax as an activation function.

- During this step we trained our model for the first time, the output gave us a starting validation accuracy. Since the provided test dataset did not include the correct predictions. We needed to use our model to predict the class for each image in the test dataset and then upload this to Kaggle, which is a machine learning community, so that we were able to get a test accuracy. Therefore, we first looked at our validation accuracy instead of our test accuracy.
- Then, we started to apply data augmentation classes from OpenCV, such as color channel and Gaussian blur. To know the influence per class, we applied them one by one. If our validation accuracy increased, we moved on to another class. If the validation accuracy did not increase, we removed it and tried with a different class again. We repeated this process until we had used all the following classes: color channel, dilate, erode, Gaussian blur, threshold, and threshold inverse.
- After getting the best combination of classes, we switched to the Keras ImageDataGenerator. Again, we decided to add only one argument at a time and see how the change affected our validation accuracy. If our validation accuracy did not increase, we removed the argument used and proceeded to add another argument. If our validation accuracy did increase, we changed the value for the argument to see if we could increase the validation accuracy more. This step was

repeated until the following arguments were all tested: rotation range, width shift range, height shift range, shear range, zoom range, flip horizontal, flip vertical, and validation split.

- During the training, in the search for the best data augmentation method, we also checked whether our model was overfitting or underfitting. We added a FC layer on under-fitted models and checked if it helped to reduce under-fitting. If not, we omitted the FC layer, and instead, increased the number of units in existing FC layers. On an overfit model, we added dropout layers. We started with a 10% drop-out and increased this as needed. If the model is still overfitting, we removed FC layers. If our validation accuracy was higher or equal to 85%, we decided to use the test dataset to get our test accuracy as well.
- We further tried VGG16 and compared the results with the VGG19 model. If the test accuracy was higher, we moved to VGG16; if not, we stayed with VGG19. We also tried one experiment with ResNet50.
- In the last step, we made changes on the number of epochs, the learning rate and the steps per epoch. The number of epochs was increased when we saw that the training accuracy was still increasing. The number of epochs was decreased when we saw that the training stopped increasing in the last few epochs. For the learning rate, we tried different values; we first started with 0.001 and then decreased the learning rate. If the change increased the test accuracy, we decreased it more. If it decreased the test accuracy, we increased the learning rate. We kept repeating

this step until we found the best learning rate. The number of epochs was also updated to see which gave us the best accuracy.

As indicated, our methodology consists of two parts. In the second part, we used decentralized training. In this case, our approach was very similar to the centralized learning. But the initial model used is different compared with centralized learning. This time, we did not start with a general model without data augmentation; instead, we started with the model and the data augmentation that gave us the best test accuracy in the centralized training. Our goal is to get a test accuracy about the same as the centralized training but faster. For this step, we would keep track of the test accuracy instead of the validation accuracy. In addition, we also changed the number of rounds to see if the experiment gave us the desired test accuracy. In the step where we decided which data augmentation would be used, we also included the Pillow classes since we observed that only using the OpenCV options did not give us a test accuracy of 87%.

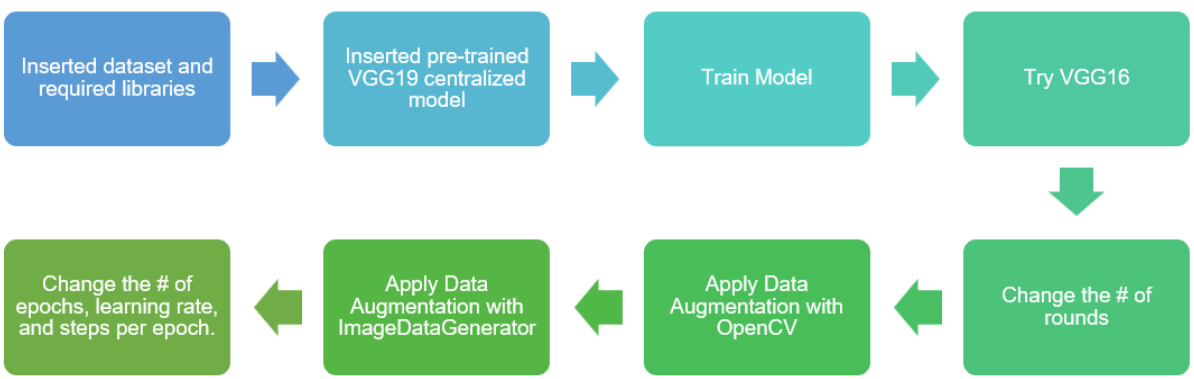


Figure 3.2: Methodology Used During Decentralized Learning.

Chapter 4: Experimental Evaluation

In this chapter, we will present our experiments with experimental result discussions for centralized and decentralized training, respectively.

4.1 Experiments on Centralized Data [1]

As said before, the goal of the NTGA is to reduce the test accuracy significantly when using the CIFAR-10 NTGA-generated dataset to train a DNN. Nevertheless, in this section, we used data augmentation techniques to demonstrate that by using CIFAR-10 NTGA generated dataset and centralized training, it is still possible to achieve a test accuracy of at least 85%. Hence, this experimental result has demonstrated that using the NTGA alone is not a guarantee to protect data from unauthorized use. This experiment was also part of the research paper "Learn From the Unlearnable: Making Unlearnable Examples Learnable Through Data Augmentation" [17], where the goal of at least 85% was set in the manuscript as well [17].

The training dataset has 40,000 poisoned images, and the validation and test datasets both have 10,000 images. Keras [3] provided the test dataset for CIFAR-10, and Yuan and Wu [28] provided the poisoned images and the validation dataset. Originally, we would use

[1] This section is a modification from a section of the manuscript [17]

Figure 4.1: Example of CIFAR-10 Images Generated by NTGA.

the provided test dataset by Yuan and Wu [28]. We evaluated our model initially on this test dataset, but we also needed to ensure that we could evaluate the decentralized learning. Therefore, we decided to use the test dataset provided by Keras for our final test accuracy. The train and validation datasets were normalized; therefore, we also normalized the test dataset. As usual, the CIFAR-10 dataset contains ten classes. The resolution of the images is 32 x 32 x 3. Figure 4.1 gives a visualization of an image of each class of CIFAR-10 generated by NTGA.

The research computing (RC) that the University of South Florida provides is used for the experiments. We used one GPU and partition simmons_itn18 [24] for this experiment.

We utilized some simple data augmentation techniques to achieve the goal of a test accuracy of 85%. To get the best combination of data augmentation we made use of the methodology as described in chapter 3. First, we used OpenCV methods to increase our dataset. For the color channels, we used the blue, green, and red channels; see section 2.1.1 for more details. We included the original unlearnable dataset as well. In addition, we also employed the built-in Keras data augmentation function, namely, ImageDataGenerator. We

used this function on our increased dataset. For the ImageDataGenerator, we used different arguments. The images were flipped horizontally, rotated by 7 degrees, and zoomed in by 0.3. Furthermore, we created images with a 35% fraction of the width and a 35% fraction of the height. Moreover, we skewed the images using a shear range angle of 0.4. We also used a validation split argument of 0.5.

Our final model was a VGG19 model with pre-trained weights from ImageNet in Keras. We tried ResNet50 and VGG16, but VGG19 performed the best among these models. Our model had four fully connected (FC) layers and two dropout layers. Each of these dropout layers would randomly drop 30% of the weights of the FC layer before the dropout layer. And they were added after the first and the second FC layer. The units of the FC layers are 1024, 512, 64, and 10, respectively. For the activation function, we used ReLU for every FC layer except for the last layer. In the last layer, since our output needs to be as a probability distribution over the ten classes, we used the Softmax function for this case. For details on the function, please see section 2.3. The learning rate was 0.008, the batch size was 100, and we used 40 epochs and 1600 steps per epoch. Furthermore, we utilized the SGD with momentum as an optimizer, and we applied the categorical cross-entropy loss as a loss function.

The details of the architecture of our model are given in Table 4.1. We can utilize Figure 4.2a and Figure 4.2b to illustrate the validation and training accuracy of the model based on the number of epochs and the validation and training loss of the model based on

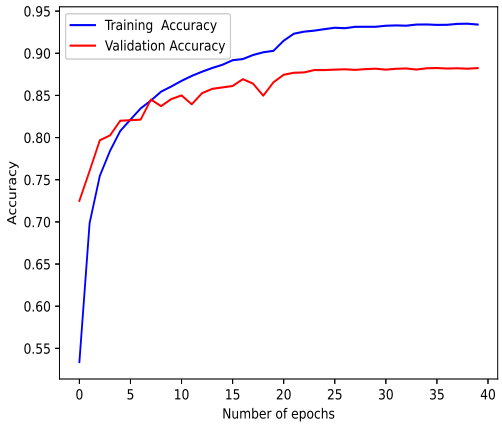
Layer	Output Shape	Activation Function
VGG19		
Flatten	512	
FC	1024	ReLU
Dropout (0.3)	1024	
FC	512	ReLU
Dropout (0.3)	512	
FC	64	ReLU
FC	10	Softmax

Table 4.1: The Model Architecture Used for Training the Unlearnable CIFAR-10 Images Generated by NTGA. The Model Achieves an Average Test Accuracy of 87.10 %. Source: Adapted from [17]

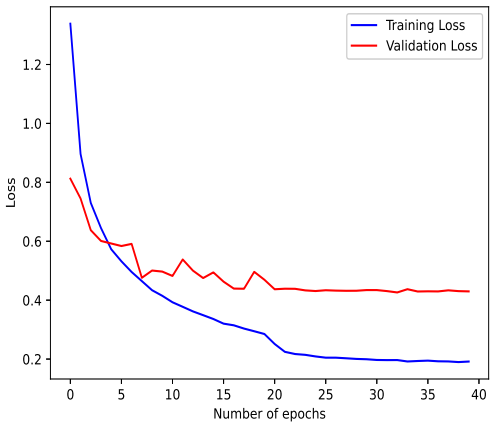
the number of epochs, which conclude that our model is slightly overfitting. We made use of the standard deviation to determine how many trials we should do. Our first trial gave a test accuracy of 86.25 % and our second trial gave a test accuracy of 87.13 %. When using both these test accuracies, we receive a standard deviation of 0.004726, so we decided to do five trials. Each trial took around 1 hour and 12 minutes.

4.2 Experiments on the IBM Federated Learning Framework

Dang [11] has modified the IBM FL framework so that it can be used on RC. Therefore, we will utilize this modified version in our decentralized learning experiments. We made use of the FL based on the speculation that decentralized learning will most likely decrease the execution time of the model training. We trained the model for 2, 4, 8, and 16 parties. Unfortunately, at the moment, we were only able to do one experiment with 16 parties since it is very difficult to reserve 16 nodes at RC.



(a) Training and Validation Accuracy of Our Model Trained with Poisoned CIFAR-10 Data.



(b) Training and Validation Loss of our Model Trained with Poisoned CIFAR-10 Data.

Figure 4.2: Training and Validation Accuracy and Loss Show that the Model is Overfitting.

The settings for the aggregator are given as follows. We used a flask connection as the connection component. We further employed an iterated average fusion handler. This means that when the model applies the fusion function, it requests the local model weights from all the parties, and the fusion function averages the weights. The protocol handler used was the one provided by the IBM FL.

Furthermore, the following gives the settings for the parties. Again, we used a flask connection for the connection component. The local training handler has been slightly modified, just like the data handler. We modified the local training handler to make use of a validation dataset during the model's training. The data handler had been modified to do some data augmentation. The TensorFlow model has also been modified to implement the ImageDataGenerator from Keras.

Moreover, the training images that we used in every experiment are the ones generated by NTGA [28]. In addition, we used the validation images provided by Yuan and Wu [28]. For the test images, we used the ones that Keras [3] provided. Both the training and validation images were in a normalized form, so we also normalized the test images. We utilized one GPU while training for each party. For the parties and aggregator, we utilized the partition simmons_itn18. When we trained 16 parties, we had to utilize both simmons_itn18 and snsm_itn19 since there were never 16 computers free with the simmons_itn18 partition. To achieve a test accuracy close to the test accuracy achieved in the central training, i.e., around 87 %, we had to increase the dataset by data augmentation again. The data augmentation that we ended up with was based on our methodology as described in chapter 3.

4.2.1 Experiments with Two Parties

We initially used the same architecture and data augmentation as the final model we got in the centralized experiments. Afterward, we updated the architecture and the data augmentation as described in chapter 3. We uniformly split the datasets into two subsets for the two parties. Now, each party had 20,000 poisoned train images, 5,000 test images, and 5,000 validation images. Each party had different images for the train, test, and validation datasets.

We first implemented the OpenCV package in Python. This information was added to the data handler of the parties. We increased our dataset 4 times, the same way as for

the centralized experiment. Again, we utilized color channels from OpenCV three separate times. We applied the color channels to get three monochrome, which are based on the color channels blue, green, and red, datasets. For the ImageDataGenerator, we utilized six arguments. The images from our increased dataset were zoomed in by 0.08, flipped horizontally, and rotated by 7 degrees. In addition, we created images that had a 35% fraction of the width and a 35% fraction of the height. We implemented the shear range arguments of 0.4, which means we skewed out images. Moreover, we again used a validation split argument of 0.5.

After extensive experiments, we found that a VGG16 model with initial weights from ImageNet could provide the best accuracy for us. We added four FC layers to the VGG16 model with 1024, 512, 64, and 10 units, respectively. We further added two dropout layers, each with 30% after the first and second FC layers. The learning rate was chosen as 0.008, and the number of epochs for each round was ten. We did not specifically defined our steps per epoch this time. We applied a sparse categorical cross-entropy loss as the loss function and a batch size of 32. As the activation function, we used ReLU for each FC layer except for the last layer; we also employed the Softmax activation function in the model. Furthermore, we did five rounds for each party. We again utilized SGD with momentum for the optimizer. We compared the average test accuracy over the 2 parties for our first 2 trials to determine how many trials are needed to ensure that the test results are reliable. Our first trial gave an average test accuracy of 87.22%, and our second trial gave an average test accuracy of

86.83%. Based on these values, we calculated the standard deviation. We received a low standard deviation of 0.006721, so we decided to do five trials. Table 4.2 shows the average test accuracy for each party over the five trials. Each trial took around one hour to train. Figure 4.3 illustrates how the test accuracy changes over the number of rounds for one trial;

Party	Average Test Accuracy (%) over 5 Trials
0	87.14
1	87.12

Table 4.2: Average Test Accuracy for Two Parties

this shows that the test accuracy of both parties is almost the same on most rounds. After the third round, the test accuracy of the two parties are different. Nevertheless, they converge again after the fifth round. Table 4.3 shows the architecture of the model gave us the best test accuracy. Figure 4.4a illustrates the validation and training accuracy in the last round for each party. Figure 4.4b illustrates the validation and training loss for the last round for each party. From these figures, we can conclude that our model is overfitting.

We applied the Chi-square test to verify if the test accuracy does not depend on the test dataset. Hence, the parties' distributions are the same. Since the test dataset is randomly selected for each party from the complete test dataset. The assumption of random selection holds. When we calculate the expected value, we get an expected value of 4356.5 (for number of correct predictions) and 643.5 (for number of incorrect predictions) for each party. Thus, all our expected values are more than 5, so we can use the Chi-square test. For our null hypothesis, we assume that the distribution is the same for the two parties. Our

Layer	Output shape	Activation function
VGG16		
Flatten	512	
FC	1024	ReLU
Dropout (0.3)	1024	
FC	512	ReLU
Dropout (0.3)	512	
FC	64	ReLU
FC	10	Softmax

Table 4.3: The Model Architecture Used for Training the Unlearnable CIFAR-10 Images Generated by NTGA Among Two Parties. Source: Adapted from [17]

alternative hypothesis says that the distributions are not the same for each party. We used R to calculate the p-value. We got a p-value of 1, which is more than 0.05. This means that we failed to reject the null hypothesis, so we cannot say that the distribution is not the same for the two parties. Thus, the test accuracy does not depend on the test dataset.

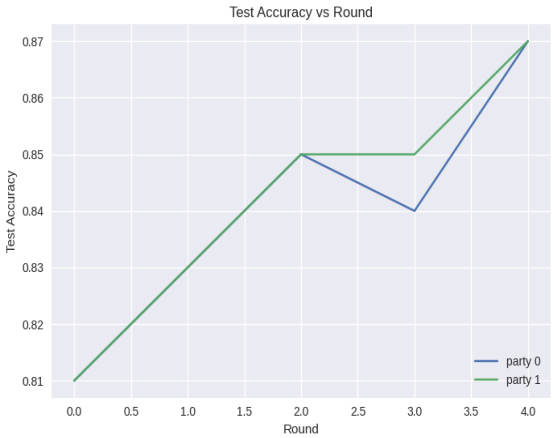
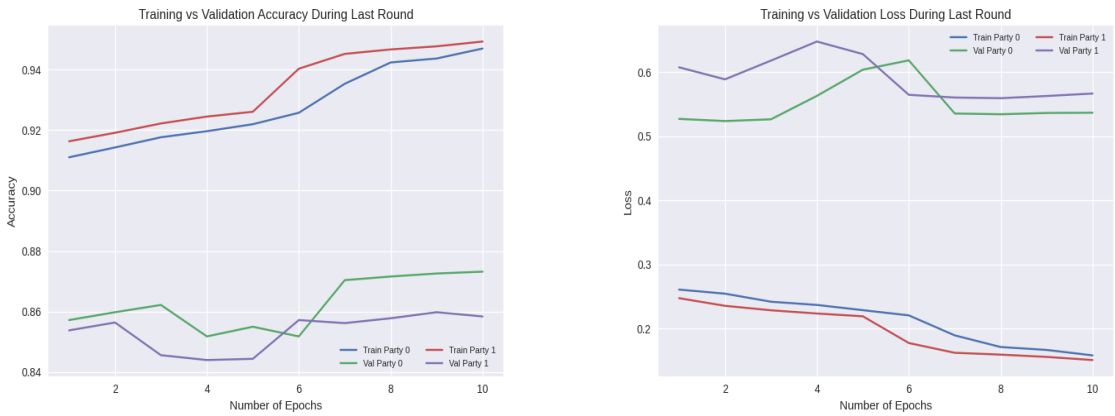


Figure 4.3: Test Accuracy of Party Zero and Party One over the Number of Rounds.



(a) Training Vs. Validation Accuracy in the Last Round.

(b) Training Vs. Validation Loss in the Last Round.

Figure 4.4: Training and Validation Accuracy and Loss for Two Parties Show that the Model is Overfitting.

When we evaluated the five final global models on the entire test dataset, we got an average test accuracy of 87.17%. Which is close to the average of all the parties, namely 87.13%.

4.2.2 Experiment with Four Parties

In our first experiment, we wanted to see how our test accuracy changes when we increase the number of parties. So, for the first part of this experiment, we applied the same hyperparameters, model architecture, and data augmentation for our experiment with 2 parties. But, instead of splitting the data over 2 parties, we now split it over 4 parties. Again, each party had different images for their test, train, and validation dataset. Hence, the test dataset had 10,000 poisoned images, and the validation and dataset had 2,500 images each.

Party	Average Test Accuracy (%) over 5 Trials
0	84.98
1	85.92
2	86.48
3	85.65

Table 4.4: Average Test Accuracy for Four Parties

Again, we compared the average test accuracy over the 4 parties for 2 trials to determine how many trials are needed for the test results to be considered reliable. Our first trial gave an average test accuracy of 85.88%, and our second trial gave an average test accuracy of 85.37%. When calculating the standard deviation, we got 0.009923, which is low. Thus, we decided again to do five trials. Table 4.4 shows the average test accuracy for each party. Each trial took about 30 minutes to train the model.

Figure 4.5 illustrates the test accuracy over the number of rounds for one trial. In this case, we can see that, at first, the test accuracy is different for each party. Nevertheless, most parties converge to the same test accuracy after the fourth round. Figure 4.6a shows the validation and training accuracy in the last round for each party. Figure 4.6b shows the validation and training loss for the last round for each party. From these figures, we can conclude that our model is again overfitting.

When we evaluated the five final global models on the entire test dataset, we got an average test accuracy of 85.71%. Which is close to the average of all the parties since this was 85.74%.

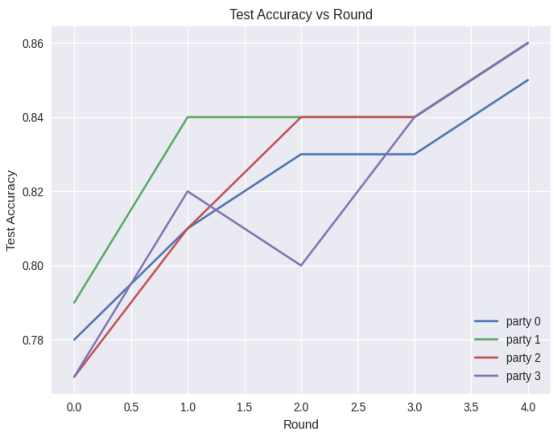
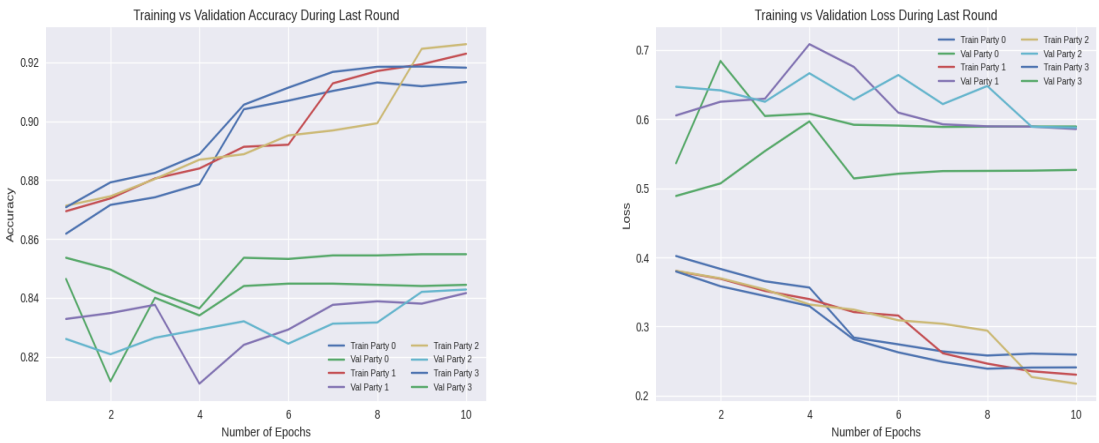


Figure 4.5: Test Accuracy of the Four Parties Vs. the Round.

Although we had a test accuracy of at least 85% to demonstrate that the NTGA-generated CIFAR-10 dataset is learnable, we also wanted to see if we could increase our test accuracy to an average of about 87% as we had for the two parties and the centralized training. To achieve this test accuracy, we used the same model architecture as before. In addition, we applied the brightness Pillow class for data augmentation so that we could increase our dataset five times. We used a factor of 1.5 to increase the brightness of our images, and we increased the number of rounds from five to eight. We removed the validation split argument in the ImageDataGenerator. This gave us an average test accuracy of 86.8%. Again, we used the standard deviation to determine how many trials we should do. Our first trial gave an average test accuracy of 86.62%, and our second trial gave an average test accuracy of 86.85%. This gave us a standard deviation of 0.009594, so we again chose to do



(a) Training Vs. Validation Accuracy in the Last Round.

(b) Training Vs. Validation Loss in the Last Round.

Figure 4.6: Training and Validation Accuracy and Loss for Four Parties Shows that the Model is Overfitting.

Party	Average Test Accuracy (%) over 5 Trials
0	86.17
1	87.35
2	87.59
3	86.54

Table 4.5: Average Test Accuracy for Four Parties for an Average Test Accuracy of 86.91%

five trials. Table 4.5 shows the average test accuracy for each party over the five trials. One trial took about one hour and one minute to train the model.

Figure 4.7 illustrates the test accuracy over the number of rounds for one trial. From this, we can see that, at first, the test accuracies are different for each party. Nevertheless, most parties converge to the same test accuracy after round eight. Figure 4.8a shows the validation and training accuracy in the last round for each party. Figure 4.8b shows the

validation and training loss for the last round for each party. From these figures, we can conclude that our model is still overfitting.

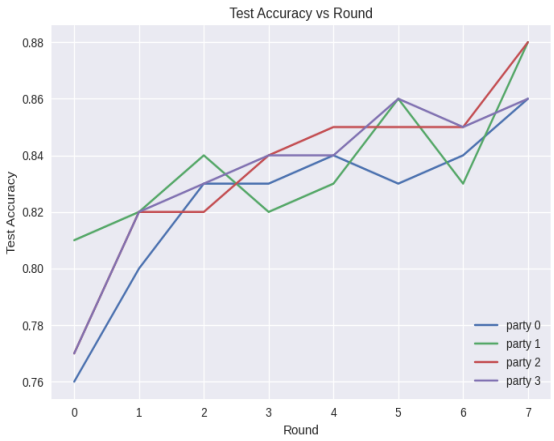
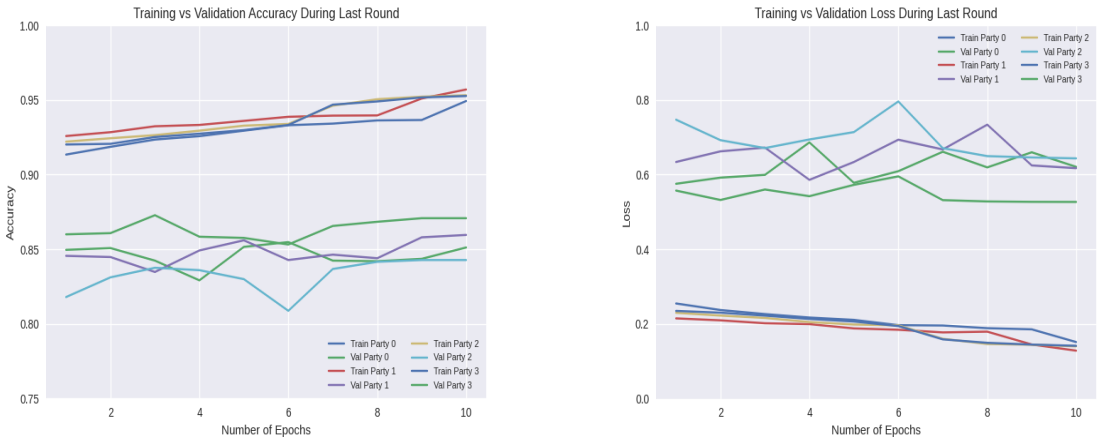


Figure 4.7: Test Accuracy of the Four Parties Vs. the Number of Rounds.



(a) Training Vs. Validation Accuracy in the Last Round.

(b) Training Vs. Validation Loss in the Last Round.

Figure 4.8: Training and Validation Accuracy and Loss for Four Parties Showing that the Model is Overfitting.

Furthermore, we applied the Chi-square test for four parties to see if the test accuracy depended on the test dataset. The assumption of random selection holds. When we calculate the expected value we get an expected value of 2173 (for number of correct predictions) and 327 (for number of incorrect predictions) for each party. Hence, all our expected values are more than 5, so we can apply the Chi-square test. We defined the null hypothesis the same as before but now for four parties. By making use of R, we received a p-value of 0.392, which is more than 0.05; hence, we failed to reject the null hypothesis. So, the test accuracy does not depend on the test dataset.

When we evaluated the five final global models on the entire test dataset, we got an average test accuracy of 86.73%, which is close to the average of all the parties, namely 86.91%.

4.2.3 Experiment with Eight Parties

For the first part of this experiment, we again implemented the same hyperparameters, model, and data augmentation as we used in the experiment with 2 parties. This time we split the data over 8 parties instead of two as discussed in the previous trial setting. Now, our training dataset consisted of 5,000 poisoned images, and the validation and test dataset had 1,250 images each. The datasets for each party were unique.

Moreover, we compared the average test accuracy among two trials for 8 parties to determine how many trials are needed for the test results to be considered reliable. Our first trial gave an average test accuracy of 83.35%, and our second trial achieved an average test

accuracy of 83.55%. When calculating the standard deviation, we got 0.014865, which is low. Thus, we decided to do five trials. Table 4.6 showed the average test accuracy for each party. Each trial took around 16 minutes to complete the model training.

Party	Average Test Accuracy (%) over 5 Trials
0	83.57
1	82.77
2	85.09
3	83.81
4	84.78
5	84.90
6	82.98
7	82.98

Table 4.6: Average Test Accuracy Among Eight Parties

Figure 4.9 illustrates the test accuracy over the number of rounds for one trial. From this, we can see that the test accuracies are scattered at the beginning. Nevertheless, after the fourth round, some of the parties' test accuracies start to converge. Figure 4.10a shows the validation and training accuracy in the last round for each party. Figure 4.10b shows the validation and training loss for each party in the last round. From these figures, we can conclude that our model is again overfitting.

When we evaluated the five final global models on the entire test dataset, we got an average test accuracy of 83.64%, which is close to the average of all the parties, namely 83.86%.

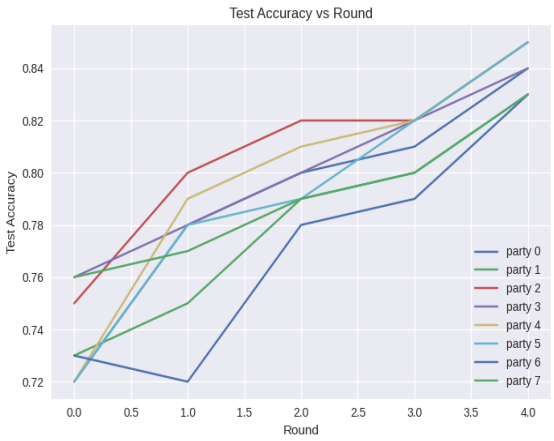
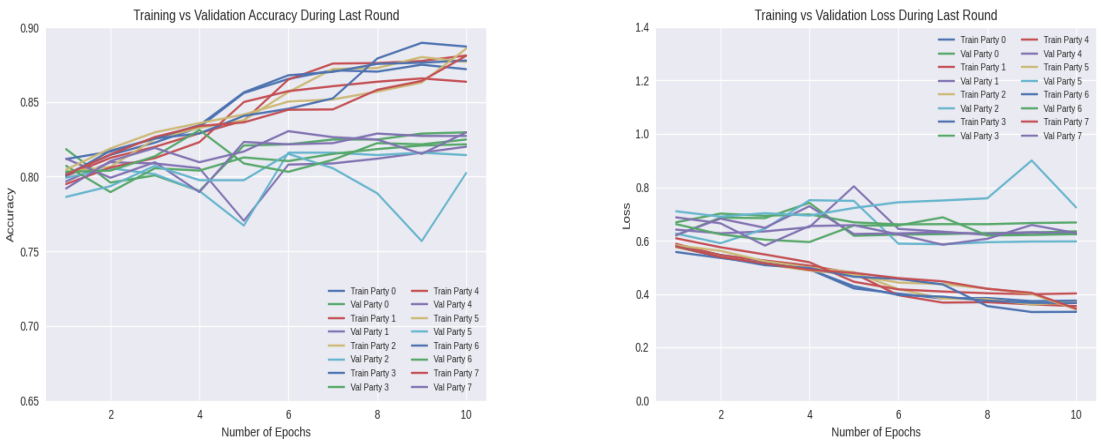


Figure 4.9: Test Accuracy of the Eight Parties Vs. the Number of Rounds.



(a) Training Vs.. Validation Accuracy in the Last Round.

(b) Training Vs. Validation Loss in the Last Round.

Figure 4.10: Training and Validation Accuracy and Loss for Eight Parties Shows that the Model is Overfitting.

The next part of the experiment was to determine if we could achieve an average test accuracy of around 87%. Unfortunately, at the moment, the highest average test accuracy we were able to achieve is 86.07%. To get this test accuracy, we increased our dataset

Party	Average Test Accuracy (%) over 5 Trials
0	86.75
1	85.57
2	86.34
3	85.55
4	86.42
5	86.48
6	85.23
7	86.11

Table 4.7: Average Test Accuracy for Eight Parties

seven times. For this purpose, we created three datasets based on the color channels. From Pillow, we implemented the enhancer to change the image's brightness, contrast, color, and sharpness. For contrast, color, and brightness, we used a factor of 1.5; for sharpness, we used a factor of 2. We utilized the same arguments in the ImageDataGenerator for the two parties, except for the validation split. We now had a value of 4 for the shear range.

Furthermore, we applied the standard deviation to determine how many trials we needed to run. Our first trial gave an average test accuracy of 85.95%, and our second trial gave an average test accuracy of 86.22%. The standard deviation was 0.013844. So, we decided to run five experiments as before. We also increased the number of rounds to ten rounds. Table 4.7 shows the average test accuracy for each party over five trials. The average execution time over the five trials is one hour and one minute.

Figure 4.11 illustrates the test accuracy over the number of rounds for one trial. From this result, we can see that the test accuracies are scattered at the beginning. Nevertheless,

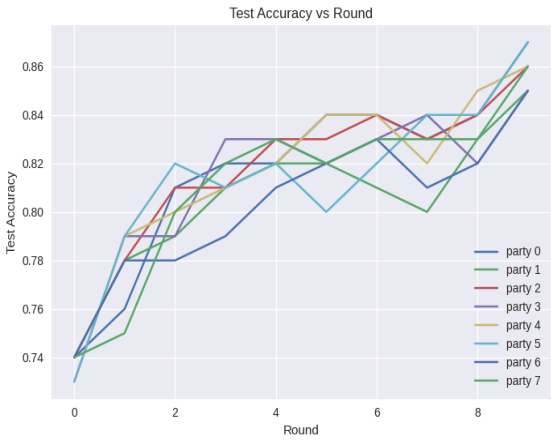


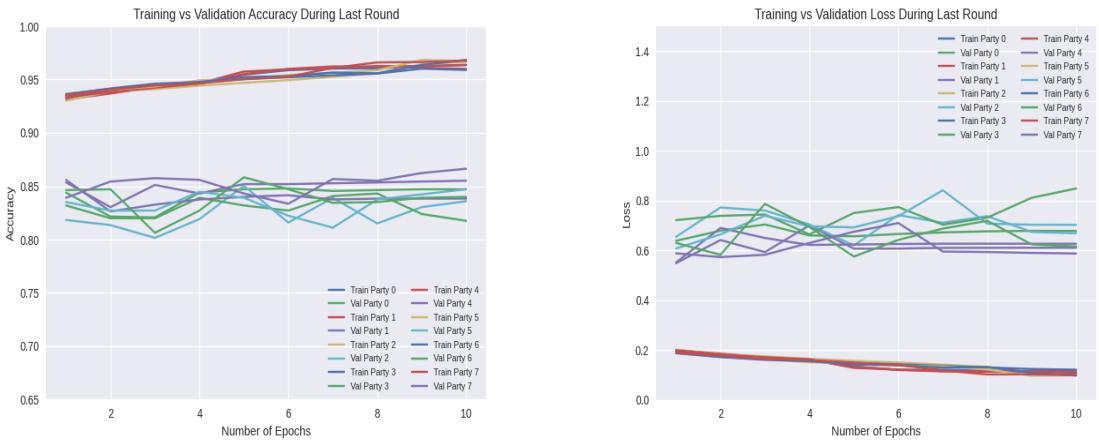
Figure 4.11: Test Accuracy of the Eight Parties Vs. the Round.

after the tenth round, some of the parties' test accuracies start to converge. Figure 4.12a shows the validation and training accuracy in the last round for each party. Figure 4.12b shows the validation and training loss for the last round for each party. From these figures, we can conclude that our model is again overfitting.

When we evaluated the five final global models on the entire test dataset, we got an average test accuracy of 86.31%, which is a little bit higher than the average of all the parties, which was 86.06%.

Again, we applied the Chi-square test to see if the distribution was the same for eight parties. Since the test dataset is random, and every expected value is greater than 5. For our null hypothesis, we have that the distribution is the same for the eight parties. Our alternative hypothesis says that the distributions are not the same for each party. We used R to calculate the p-value. We got a p-value of 0.953, which is more than 0.05; that is, we

failed to reject the null hypothesis, so we cannot say that the distribution is different.



(a) Training Vs. Validation Accuracy in the Last Round.

(b) Training Vs. Validation Loss in the Last Round.

Figure 4.12: Training and Validation Accuracy and Loss for Eight Parties Showing that the Model is Overfitting.

4.2.4 Experiments with Sixteen Parties

In this experiment we implemented the same model and hyperparameters as for the experiment with two parties. We also did five rounds and ten epochs. In our experiment, each party had 2,500 poisoned training images, 625 validation images, and 625 test images. Furthermore, we applied standard deviation to determine how many times we should train the model. Our first average test accuracy was 81.33%, and our second average test accuracy was 81.05%. Therefore, we got a standard deviation of 0.022107, which is again low; so, we decided to train the model five separate times. Table 4.8 shows the average test accuracy for each party. The average time it took for each trial is 12 minutes. However, for the 16 parties we needed to use a combination of two partitions: simmons_itn18 and snsm_itn19, which

are the computing resources at RC. This is because we have observed that snsm_itn19 is in general slower we can make the assumption that if we would be able to use simmons_itn18 for all the parties, the training time would reduce.

Party	Average Test Accuracy (%) over 5 Trials
0	78.94
1	81.86
2	82.72
3	80.86
4	82.05
5	81.22
6	81.92
7	83.52
8	82.88
9	81.09
10	79.17
11	81.25
12	80.16
13	81.52
14	81.22
15	79.39

Table 4.8: Average Test Accuracy for Sixteen Parties

We can use Figure 4.14 to illustrate the test accuracy over the number of rounds for one trial. From this figure, we can see that the test accuracies start to converge in the end. Figure 4.14a shows the validation and training accuracy in the last round for each party, whereas Figure 4.14b shows the validation and training loss in the last round for each party. From the results, we can conclude that our model is slightly overfitting.

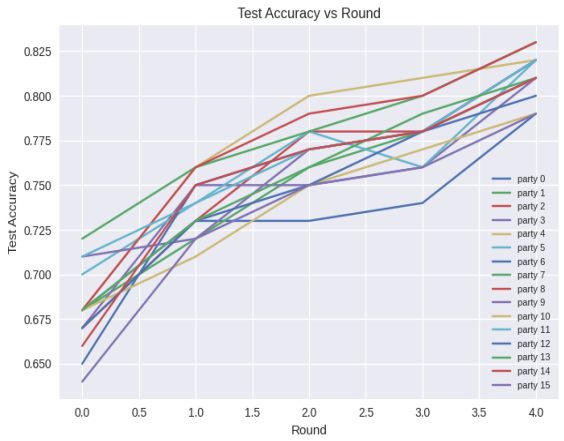
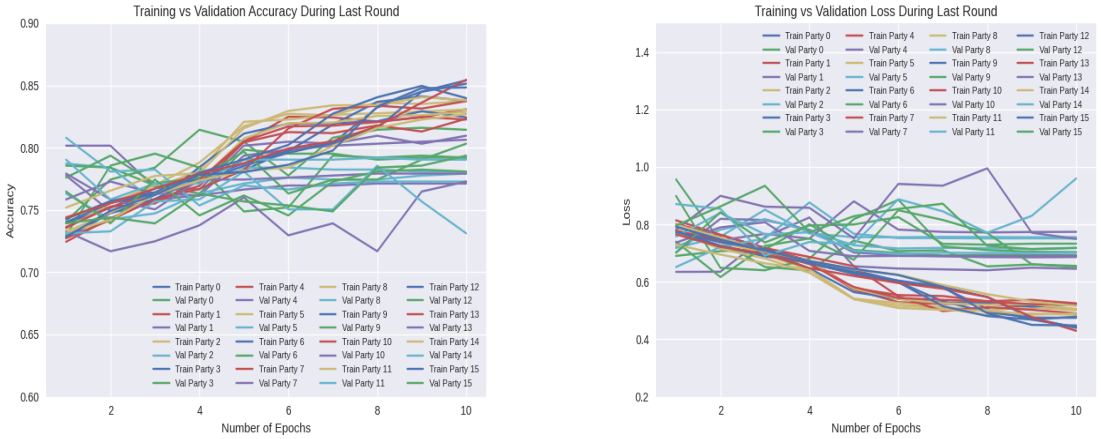


Figure 4.13: Test Accuracy of the Sixteen Parties Vs. the Number of Rounds.



(a) Training Vs. Validation Accuracy in the Last Round.

(b) Training Vs. Validation Loss in the Last Round.

Figure 4.14: Training and Validation Accuracy and Loss for Sixteen Parties Showing that the Model is Slightly Overfitting.

When we again evaluated the five final global models on the entire test dataset, we got an average test accuracy of 81.12%, which is close to the average test accuracy over all the parties, 81.21%.

Even though the test accuracy did not reach our goal, we still applied the Chi-square test to see if the distribution over the parties is the same. The assumptions are still met since the samples are selected at random and none of the expected values are less than five. If our null hypothesis is the same as before, i.e., the distribution over the parties is the same. We can find that when we use R to calculate the Chi-square, we get a p-value of 0.4973; if we again use a significance level of 5%, we can see that we fail to reject the null hypothesis.

Unfortunately, we were not able to do more experiments using 16 parties at this moment since research teams compete for computing resources at RC and we have the difficulty to 16 notes for the 16 parties at one time, which is needed in the experiment. Thus, we were not able to increase the test accuracy to at least 85%.

4.3 Comparison between Number of Parties

In this section, we examine the influence of the number of parties on the test accuracy and the training speed. When we increase the number of parties and keep the same model architecture and data augmentation techniques, the time it takes to train a model is reduced. Unfortunately, this also reduces the average test accuracy. We could get around the same test accuracy for 1, 2, and 4 parties. The smallest amount of time to get a test accuracy of 87% was around one hour for two and four parties. We have not gotten a test accuracy of 87% using less time training the model. At the moment, we were not able to get an average test of 87% for 8 parties. We were also not able to get a test accuracy of 85% for 16 parties, which was due to not enough available resources.

Number of Parties	Number of Rounds	Average Test Accuracy (%) over 5 Trials	Average Time
1	1	87.10	1 hour and 12 minutes
2	5	87.17	1 hour
4	5	85.71	30 minutes
4	8	86.73	1 hour and 1 minute
8	5	83.64	16 minutes
8	10	86.31	1 hour and 1 minute
16	5	81.12	12 minutes

Table 4.9: Average Test Accuracies and Average Time with Number of Rounds

Table 4.9 shows the average of all parties over five trials and the number of rounds and the average time it takes to achieve the accuracy.

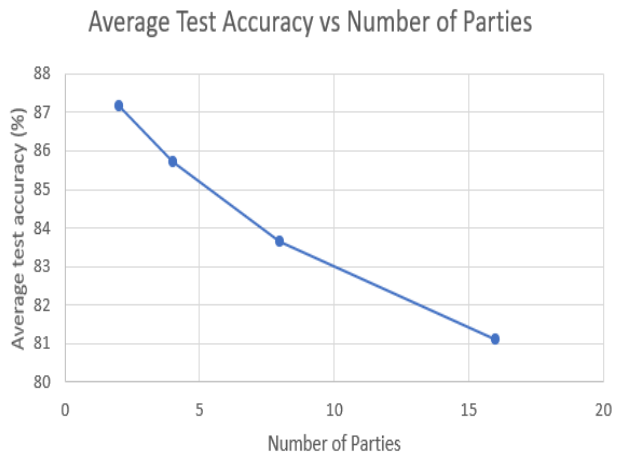


Figure 4.15: Average Test Accuracy Over the Number of Parties.

From these experiments, we have seen that we got an average test accuracy of at least 85% for centralized learning and decentralized learning with 2, 4, and 8 parties. Hence, we have showed that the unlearnable CIFAR-10 dataset is vulnerable. In addition, we have also noticed that the test accuracy seemed to drop when we increased the number of parties. So,

for more than eight parties, additional data augmentation and an increase in the number of rounds are most likely needed to get an average test accuracy of at least 85%. All our models were overfitting, so more research needs to be done to see if this can be solved. Only the model for 16 parties is slightly overfitting, but this model did not achieve a test accuracy of at least 85% yet. More experiments need to be done before we can make any conclusion on this overfitting matter. Figure 4.15 shows how the test accuracy drops, when we keep the model and hyperparameters the same. From this figure we can see that there is a negative relationship between the increase in the number of parties and the test accuracy. This negative relationship is most likely due to the use of more parties while the total amount of data stays the same. This means that each party has less data to be trained on, which results in a lower test accuracy.

Chapter 5: Conclusions and Future Work

The need for data security has significantly increased over the past decades, and many studies have been suggested to deal with data security. Among them, the NTGA technique has become one of the most efficient methods to generate unlearnable datasets. Nevertheless, in this research, we have shown that a DNN in a centralized setting can still learn from the unlearnable CIFAR-10 dataset generated by an NTGA. In addition, we have further studied to see if we could get to the same test accuracy but faster, which may be critical to some time-sensitive applications in the real world. For this purpose, we have utilized decentralized training to investigate NTGA-generated unlearnable CIFAR-10 dataset. This decentralized learning technique, generally speaking, is predicted to decrease the execution time because the model trains on smaller datasets since the dataset is split among the parties, and the model training on the smaller dataset for each party should take less time. By using extensive experiments, we have demonstrated that the NTGA-generated dataset is still vulnerable to data augmentation and have confirmed the above prediction. Specifically, we were able to decrease the time by 12 minutes, which is a 16% decrease compared to centralized learning when two parties were used in the decentralized learning. We have also seen some execution time reductions for four and eight parties, respectively. At this moment, we have not been able to conclude what will happen when we utilize decentralized learning for more than eight

parties since it is very difficult to reserve a large amount of computing resources at USF's Research Computing (RC). For this reason, we will continue to conduct more experiments for more than eight parties in our future research. In addition, it is also interesting to see what happens when we start with something other than the centralized model when utilizing decentralized training. So, we will research this as well. As said before, more research will also need to be done to see if we can reduce the overfitting of the models we used. Moreover, more and more data are generated daily. The demand for better data protection will also increase. We will also explore new research methods to protect data in the future.

References

[1] Yong Cheng, Yang Liu, Tianjian Chen, and Qiang Yang. Federated learning for privacy-preserving ai. *Communications of the ACM*, 63(12):33–36, 2020. https://doi.org/10.1145/3387107.

[2] Lenaic Chizat, Edouard Oyallon, and Francis Bach. On lazy training in differentiable programming. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPs)*, pages 2933–2943, 2019.

[3] François Chollet et al. Cifar10 small images classification dataset. https://keras.io/api/datasets/cifar10/. Accessed: October 14, 2022.

[4] François Chollet et al. Dropout layer. https://keras.io/api/layers/regularization_layers/dropout/. Accessed: October 12, 2022.

[5] François Chollet et al. Layer activation functions. https://keras.io/api/layers/activations/. Accessed: October 14, 2022.

[6] François Chollet et al. Optimizers. https://keras.io/api/optimizers/. Accessed: October 14, 2022.

[7] Alex Clark and Contributors. Image module. https://pillow.readthedocs.io/en/stable/reference/Image.html. Accessed: September 22, 2022.

[8] Alex Clark and Contributors. Source code for pil.imageenhance. https://pillow.readthedocs.io/en/stable/_modules/PIL/ImageEnhance.html#_Enhance. Accessed: September 22, 2022.

[9] Intersoft Consulting. Artical 21 GDPR right to object. https://gdpr-info.eu/art-21-gdpr/. Accessed: September 18, 2022.

[10] Intersoft Consulting. General data protection regulation GDPR. https://gdpr-info.eu/. Accessed: September 18, 2022.

[11] Long Dang. Federated adversarial training (FAT) in high performance computing (HPC) cluster. Technical report.

[12] Truong-Dong Do, Minh-Thien Duong, Quoc-Vu Dang, and My-Ha Le. Real-time self-driving car navigation using deep neural network. In *Proceesings of the International Conference on Green Technology and Sustainable Development (GTSD)*, pages 7–12. IEEE, 2018.

[13] Centers for Disease Control and Prevention. Health insurance portability and accountability act of 1996 (HIPAA). https://www.cdc.gov/phlp/publications/topic/hipaa.html. Accessed: September 18, 2022.

[14] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018. http://arxiv.org/abs/1811.03604.

[15] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPs)*, pages 8580–8589, 2018.

[16] Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPs)*, pages 8570–8581, 2019.

[17] Jing Lin, Thushari Hapuarachchi, Gitte Ost, Mohamed Rahouti, and Kaiqi Xiong. Learn from the unlearnable: Making unlearnable examples learnable through data augmentation. Technical report.

[18] Heiko Ludwig, Nathalie Baracaldo, Gegi Thomas, Yi Zhou, Ali Anwar, Shashank Rajamoni, Yuya Ong, Jayaram Radhakrishnan, Ashish Verma, Mathieu Sinn, et al. Ibm federated learning: an enterprise framework white paper v0. 1. *arXiv preprint arXiv:2007.10987*, 2020. https://arxiv.org/abs/2007.10987.

[19] Aleksander Mądry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2018.

[20] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the Artificial Intelligence and Statistics (AISTATS)*, pages 1273–1282. PMLR, 2017.

[21] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pages 691–706. IEEE, 2019.

[22] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018. `http://arxiv.org/abs/1811.03378`.

[23] State of California Department of Justice Office of the Attorney General. California consumer privacy act (CCPA). `https://oag.ca.gov/privacy/ccpa`. Accessed: September 18, 2022.

[24] Univeristy of South Florida. Circe hardware. `https://wiki.rc.usf.edu/index.php/CIRCE_Hardware`. Accessed: September 22, 2022.

[25] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[26] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014. `https://dl.acm.org/doi/10.5555/2627435.2670313`.

[27] Kexin Ye, Lei Zhang, and Tianran Li. Design of a picture-seeing and talking system based on attention mechanism. *American Journal of Electrical and Electronic Engineering*, 10(1):6–23, 2022. `http://doi.org/10.12691/ajeee-10-1-2`.

[28] Chia-Hung Yuan and Shan-Hung Wu. Neural tangent generalization attacks. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 12230–12240. PMLR, 2021.