May 2021

Privacy-Preserving Image Classification Using Convolutional Neural Networks

David Karl Langbehn
Clemson University, dklangbe@gmail.com

Privacy-Preserving Image Classification Using Convolutional Neural Networks

---

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Computer Engineering

---

by
David Karl Langbehn
May 2021

Accepted by:
Dr. Melissa Smith, Committee Chair
Dr. Adam Hoover
Dr. Jerome McClendon

Abstract

The process of image classification using convolutional neural networks (CNNs) often relies on access to large, annotated datasets and the use of cluster or cloud-based computing resources. However, many classification applications such as those in healthcare or defense introduce privacy concerns that prevent the collection of such data and the use of pre-existing large scale computing systems. Although many solutions to privacy preserving machine learning have previously been explored, the added computational complexity incurred with training on encrypted values inhibits these systems from executing in real-time. One of the most promising solutions that facilitates secure machine learning is secure multi-party computation (MPC), which relies on segmenting data across multiple devices such that the original data cannot be reconstructed without recombining each of the data segments.

This thesis explores the efficacy of training CNNs on encrypted data using MPC techniques and utilizes several optimization techniques to lessen the computational and communication overheads incurred from doing so. The goals are to create a privacy-preserving CNN framework that achieves testing accuracy similar to a non-secure model while introducing the least amount of computational overhead. To this end, a multi-party encryption scheme was used to encrypt all floating point values used in training, and federated learning was incorporated to reduce the effects of the computational overhead by parallelizing the training of the network.

The developed secure CNN was able to achieve validation accuracy within 1.1-2.8% of a baseline CNN on the MNIST dataset and 9.9-19.4% on the CIFAR-10 dataset. This

decreased accuracy is caused by rounding errors incurred by performing multiple continuous arithmetic computations in the secure domain during training, however the accuracy results of the secure CNN indicate that training can be performed on encrypted values. The cost of performing training on encrypted values was found to range from between 8 - 21$\times$ more computation time in comparison to a non-secure baseline implementation due to the added computational complexity and communication overhead required to perform training on secure values. This additional training time, however, was shown to be able to be mitigated through the use of federated averaging by performing training on multiple devices in parallel.

Dedication

I dedicate this work to my loving wife as without your continuous support, none of this would have been attainable. Your encouragement has allowed me to overcome challenges that I never thought possible, my gratitude for which could never be fully expressed in words.

Acknowledgments

I would like to express my gratitude to Dr. Jerome McClendon for his support and guidance throughout the completion of this research. I would also like to thank my advisor Dr. Melissa Smith for all of her steadfast support throughout my graduate career and Dr. Adam Hoover for agreeing to serve on my advisory committee.

I would like to also thank my friends and family for their support and belief in me, and all of the members of the FCTL group for their valuable help and advice throughout this research.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

The field of computer vision has seen major improvements in recent years with the advent of deep learning architectures such as convolutional neural networks and recurrent neural networks. These advancements have pushed the boundary of what is possible when it comes to classifying the objects present in images and tracking moving images through multiple frames of video as well as have proven useful in other related fields such as speech recognition and text classification. However, the drawback of these artificial intelligence (AI)-based methods compared to previous hand-tuned techniques is the large datasets required to train these networks. Access to these datasets and to the underlying features learned from them poses a significant challenge in many fields where human privacy and other security concerns are a priority.

Previous research has explored multiple approaches for addressing this challenge, with the primary goal being to perform computations directly on the encrypted data. This capability opens up the possibility to train these complex networks on non-secure devices such as largescale cloud computing servers without the unencrypted data being available to third parties. With the scope of AI-based recognition and classification solutions broadening to more fields and access to high performance computing (HPC) resources becoming easier

to obtain, the need for privacy-preserving algorithms is growing substantially.

Currently, two emerging encryption schemes include the capability to perform arithmetic computations on encrypted data, each with its own limitations. While the first of these approaches explored in this research, homomorphic encryption (HE), facilitates solving of Boolean or arithmetic circuits between two encrypted values without access to a secret key, its disadvantage is the added computational complexity required to perform any operation in the encrypted domain as well as the inability to perform an arbitrary number of computations without increasing the errors in the result. Multiplication between two encrypted values is its primary weak point as the output is prone to errors accruing from small amounts of noise added through the encryption scheme. As many image classification systems perform thousands to millions of multiplications in a single training or validation cycle, these errors compound and lead to meaningless outputs. This issue limits the use less computationally expensive versions of this scheme as we need an encryption scheme that can handle multiplications with less overhead.

The second encryption scheme that explored here is secure multi-party computation (MPC), which trades computational overhead for communication overhead. Various encryption protocols have been developed that support MPC, each based on the basic idea of splitting shares of a single encrypted value across multiple devices such that if each device performs the same set of instructions on its share, a later recombination of all shares will have the same computations performed on the final output. However, while this approach again adds complexity when dealing with multiplication, there are optimization techniques that allow multiplication between two encrypted values in only a few rounds of communication.

1.2 Contributions

In addressing these issues, this research offers several contributions which can be summarized as follows. First, a basic convolutional neural network was developed as a

baseline for measuring the most time-consuming computations that would benefit from running on a distributed architecture. Next, an MPC scheme was created to perform the floating-point arithmetic required for these expensive computations to run on a non-secure device or server. Doing so involved minimizing the total rounding error incurred by doing each secure computation so that it had the least total affect on the desired output. Minimizing these errors is important as they can compound significantly through many repeated computations in the secure domain. Once a complete MPC scheme that could handle each type of operation required to train the network was developed, it was integrated into a secure network that uses encrypted values for all inputs, outputs, and intermediate values needing to be accessed by a third party during training. Finally, optimization techniques such as federated learning were introduced to limit the added communication and computational overhead incurred through the use of the MPC scheme.

The secure architecture developed in this work was evaluated against the baseline architecture to determine the feasibility of performing the backwards phase of learning in the encrypted domain. This backpropagation training phase was chosen because through testing it was determined to be the most computationally expensive and as such would benefit from running on a third party computing platform. The goal of this network is to achieve accuracy results similar to the baseline architecture, with the additional overhead not offsetting the potential improvements gained from the utilization of more computational power. To test the feasibility of such a network, the proposed MPC scheme was simulated since implementing the communication protocols required to run the network on multiple devices was outside of the scope of this research.

1.3 Thesis Outline

This thesis is organized as follows. Chapter 2 provides background information on image classification techniques and the calculations involved in convolutional neural networks in addition to detailing federated learning and the encryption protocols used in this

research. Chapter 3 summarizes the methods used and the results from past research relevant to secure machine learning. Chapter 4 details the experimental design, including the implementation details and the network layout of both the baseline and secure architectures as well as the measurements used to test the efficacy of the secure implementation. Chapter 5 presents the results of the privacy preserving CNN, comparing them to the baseline architecture as well as to other methods used in literature. Chapter 7 concludes with observations regarding the capability of the proposed architecture as well as detailing suggested areas for future work.

Chapter 2

Background

This chapter introduces key concepts used in image classification as well as provides details on the algorithms used in this research, beginning with the image classification techniques and datasets commonly used in this field. The second section details convolutional neural networks (CNNs) including the layer types commonly used in them, followed by an introduction into federated learning and the various strategies for distributed learning. The fourth section covers secure multi-party computation (MPC) including implementation details as well as highlighting the advantages and disadvantages of this encryption approach. This chapter concludes with details about the MPC encryption protocol used in this research.

2.1 Image Classification

Image classification, which entails analyzing an entire image in order to assign it labels, typically refers to cases where only one object appears in an image, whereas object detection refers to classifying multiple objects within a single image. Multiple classification methods have been developed over the years, with the main goal being to reduce the number of images a system classifies incorrectly [29]. Most of these methods rely on some form of feature extraction, which allows a system to analyze the most relevant pixels in an image

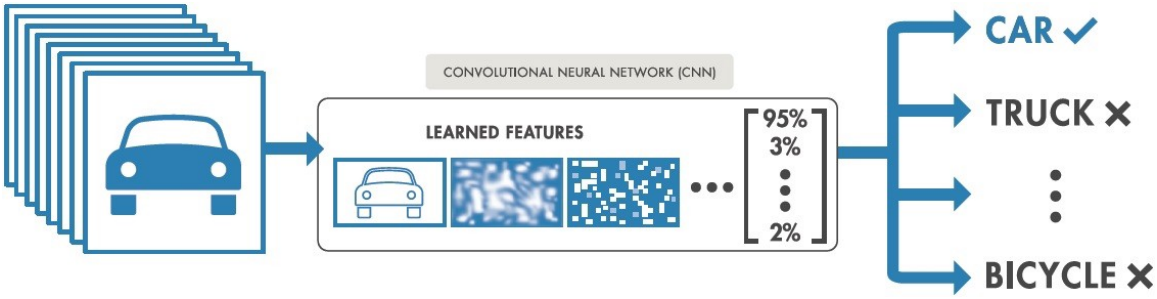


Figure 2.1: Example CNN used for image classification

rather than analyzing each pixel individually. This approach also results in a reduction in the dimensionality of the image data, subsequently reducing the computational complexity of the classification. In image classification, for example, these features, which can take the form of shapes, edges, or patterns that occur within the image, can be either hand tuned for a given dataset or learned directly from the data through the process of machine learning.

Early classification techniques were based on supervised learning and statistical analysis of a dataset and relied on user input to hand-tune model parameters [26]. These algorithms include support vector machines, k-nearest neighbor, linear regression, and naive Bayes. While these earlier methods were able to achieve high accuracy across numerous datasets, each implementation requires the fine tuning of multiple parameters, meaning they require much effort to create and are not generalizable.

The field of image classification has seen major improvements in recent years with the introduction of neural network-based approaches [31]that rely on algorithms that allow the system to learn features without the need for human input and, as such, can determine features that are more meaningful for a given dataset than might be decided by a user. Neural network-based techniques such as CNNs have proven to be more robust across multiple datasets than previous statistical learning methods, surpassing previous methodologies in classification accuracy for many widely used datasets [16].

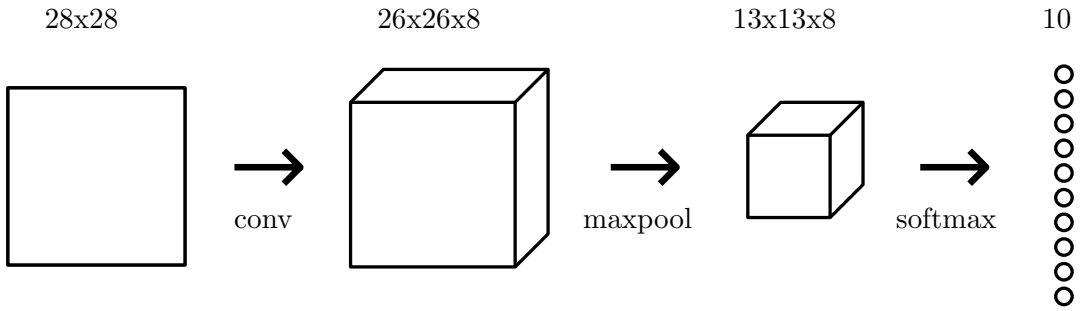


Figure 2.2: Basic CNN Layer Configurations

2.2 Convolutional Neural Networks

Convolutional Neural Networks were first proposed as a method of image classification in 1989 [21], but it wasn't until recently that their performance outpaced current statistical learning techniques [20]. A typical CNN uses different layers to learn features from a set of input data. These features are typically stored as local weights that are used by the layer to calculate its output. For a given layer function f, the output of the layer $y^l_{i,j,k}$ at location (i, j) of the k-th feature map and layer l is calculated by:

$$y^l_{i,j,k} = f(w^l_k, x^l_{i,j}) + b^l_k \tag{2.1}$$

where w^l_k is the weight vector and b^l_k is the bias term of the k-th filter of the l-th layer, and $x^l_{i,j}$ is a region of the input to layer l centered at location (i, j) [14].

A typical CNN begins with a convolutional layer that performs a convolution between an input image and a set of filters consisting of trainable weights used to extract features from the input data. The output of this layer is then used as input for a pooling layer, which reduces the dimensionality of the extracted features because neighboring pixels in an image often have similar values, meaning a convolutional layer will produce similar values for neighboring pixels in its output. The output of the pooling layer is then fed into either additional convolutional layers or an output layer that determines the class the input

image belongs to [34]. These layers are described in more detail in Sections 2.2.2 - 2.2.4.

Many modern CNNs make use of multiple layer types, with the best performing CNNs typically utilizing multiple layers totaling to hundreds of thousands of features present in the network. As such the computational complexity of these networks can make training times prohibitively long when running on a CPU. However, many of the algorithms used by individual layers have proven to be parallelizable on GPUs, and overall training times can be drastically shortened using such methods [8]. In addition, pre-processing the training data has been found to increase classification accuracy, as well as limit the over-fitting of weights to the training data [7].

2.2.1 Backpropagation

As a network trains on a set of data, both the weights and biases of each layer are updated using a loss function to minimize classification error by propagating the error of the prediction backwards through the network and updating the weights of each layer relative to their impact on the final output of the network [21]. This backpropagation of error through the network is ultimately what enables the learning of features to occur as the initial weights used for each layer are typically randomized at the beginning of training.

The classification error of the network is calculated through the use of a loss function. While many such functions exist, a cross-entropy loss function is detailed here as it was used in this research. The goal of a loss function is to describe how sure a network is of each prediction: If a network is confident it has identified the correct class, then it can be assumed that it has correctly learned the underlying features of that class. Cross-entropy loss can be calculated as:

$$L = -\ln(p_c) \tag{2.2}$$

where the loss L is the value we wish to minimize, c is the value of the correct class, and p_c is the predicted probability for class c. The value of L, which is calculated during the

forward pass of the network, is subsequently propagated backwards through the network so that each layer may update its weights based on whether the prediction was accurate or not.

Stochastic gradient descent (SGD) is one loss optimization method used to back-propagate this loss through the network. In SGD, the act of minimizing the loss is performed after each image is passed through the network. As a result, a higher number of iterations are typically required to find global minima than for other techniques such as batch gradient descent; however this is offset computationally by requiring each image to pass through the network only once. SGD, an iterative algorithm, is generally written as:

$$w = w - \frac{\eta}{n} \sum_{i=1}^{n} \nabla L_i(w) \tag{2.3}$$

where w is the weight parameter that is to be updated, η is the learning rate of the network, ∇ is the loss gradient of the previous layer, and $L_i(w)$ is the value of the loss function for the i-th image. The gradient ∇, which can be viewed as the input to a layer during the backwards pass through the network, describes the loss of the network relative to a given layer's output during the forward pass. To calculate this gradient, a layer must temporarily store its input during forward propagation.

2.2.2 Convolutional Layer

CNNs get their name from the use of convolutional layers in the network. These convolutional layers take an image as input, and performs a set of convolutions between that image and a set of filters that store the layer's weights. This set of convolutions can be written as:

$$y_{i,j,k} = \sum_{n=1}^{N} \sum_{m=1}^{M} x_{i+n,j+m} * w_{n,m,k} \tag{2.4}$$

where an output pixel $y_{i,j,k}$ represents a convolution between an $N \times M$ region of the input x and filter w_k. Given k filters and no padding of the input, this results in an output of size

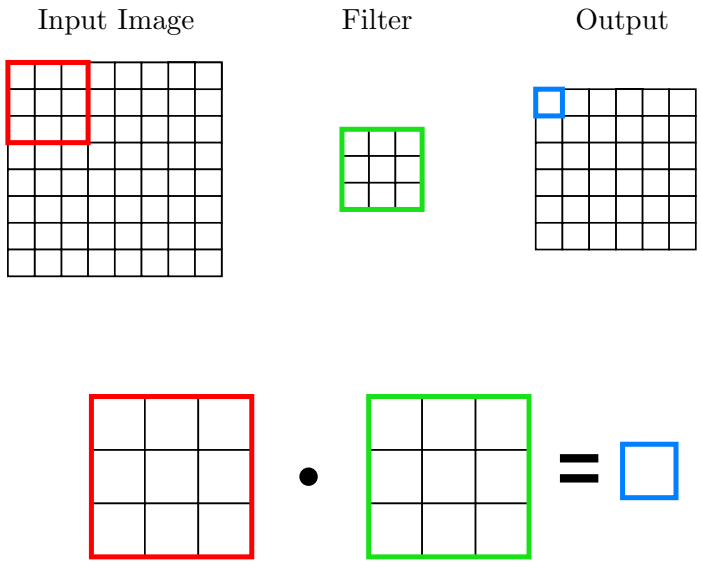


Figure 2.3: Convolutional layer functionality during the forward propagation of data through the network using a 3×3 filter and a stride of 1

$I - N - 1 \times J - M - 1 \times K$ where the input to the layer is of size $I \times J$. When no padding is used it is important to note the shrinking of the output relative to the input, as the use of multiple convolutional layers in a network is limited by the size of the filters.

The goal of performing convolutions on an image with a set of filters is to extract features from that image, with each filter responsible for extracting a different type of feature. These features, which can range from edges, shapes, colors or textures, are used to help the network classify an image as one of multiple classes. For example, if the network is designed to distinguish between different types of animals, features relating to fur or color may be learned.

The act of learning these features occurs during the backpropagation of loss gradients through the network, and the weight values are updated based on their impact on the overall loss of the network as in Equation (2.3) using:

$$\frac{\partial L}{\partial w_{n,m,k}} = \sum_{i=1}^{I} \sum_{j=1}^{J} x_{i+n,j+m} * \nabla_{i,j,k} \tag{2.5}$$

to find the effect of a single weight on the network's loss where ∇ is the loss gradient

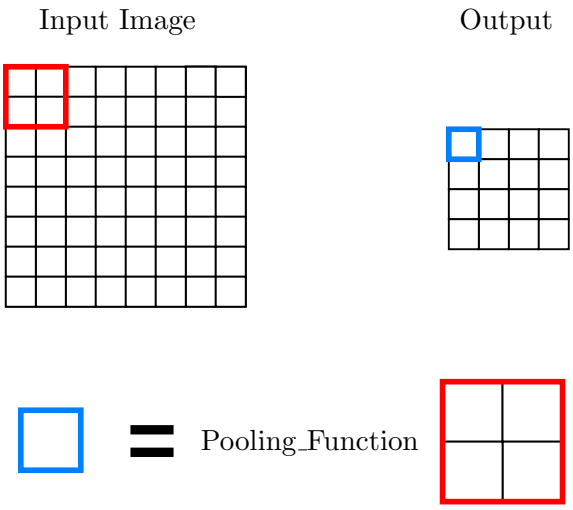


Figure 2.4: Pooling layer feature reduction using 2 × 2 filters and a stride of 2

propagated from the previous layer and x was the input to the layer during forward propagation. If multiple convolutional layers are used in a network, the gradient $\frac{\partial L}{\partial input}$ has to be propagated to the previous layers as well, calculated as:

$$\frac{\partial L}{\partial input_{i,j}} = \sum_{n=1}^{N} \sum_{m=1}^{M} \nabla_{i+n,j+m,k} * w_{n,m,k} \tag{2.6}$$

2.2.3 Pooling Layer

Pooling layers are used to reduce the size of their input, and as such decrease the number of weights required in subsequent layers in the network. Typically, a pooling layer is placed after a convolutional layer in the network, with many types of pooling functions being found in literature. As these functions simply reduce the dimensionality of an input, they do not require any trainable weights.

Two of the most common types of pooling layers are max pooling and average pooling. Max pooling layers propagate the maximum value of each $P \times P$ region of the input, where P is a predetermined pooling size. For example, using a value of $P = 2$ leads to a reduction of the width and height of the input by a factor of two. During backpropagation the loss gradient is passed through to the maximum pixel in each region

as it was the only value that influences the output of the network. While average pooling layers follow the same reduction principle, they propagate the average value of a $P \times P$ region of the input. The loss gradient for the layer is calculated by multiplying the gradient by $\frac{1}{P \times P}$ and assigning the error to every pixel in a pooling block. Using max pooling layers rather than average pooling layers generally leads to higher classification accuracy for a given network as the maximum value of a pooling region is often more meaningful than the average.

2.2.4 Softmax Layer

The softmax layer is a fully connected layer that uses the softmax function as its activation function. The purpose of this layer is to create a probability distribution to determine the network's confidence in a given prediction. This is accomplished in part by calculating an intermediate value corresponding to the totals of the network, t as follows:

$$t = w * input + b \tag{2.7}$$

where w and b are the weights and biases of the softmax layer, respectively. To create a probability distribution such that all values of t_i sum to one, the softmax activation function is applied:

$$\text{softmax}(t_i) = \frac{e^{t_i}}{\sum_{j=1}^{n} e^{t_j}} \tag{2.8}$$

the output of which should be highest for $t_i = c$, signifying a correct classification.

To update the weights and biases of the softmax layer during training, its effect on the total loss of the system is calculated. The loss function for the cross-entropy function described in (2.2) can be written as:

$$\frac{\partial L}{\partial out_s(i)} = \begin{cases} 0 & \text{if } i \neq c \\ -\frac{1}{p_i} & \text{if } i = c \end{cases} \tag{2.9}$$

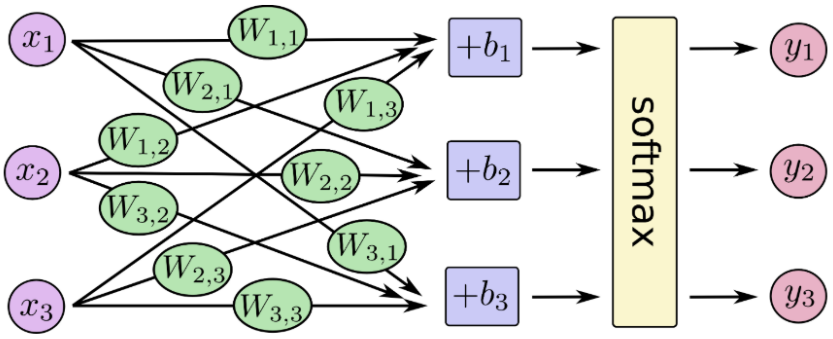


Figure 2.5: Softmax layer functionality during forward propagation

where $out_s(i)$ is the output of the softmax layer for class i in the forward direction and p_i is the predicted probability of class i. The result is the initial gradient of the network and is fed as input to the softmax layer during backpropagation.

Multiple local gradients are calculated in order to calculate the output gradient that will be used as input for the previous layer in the network as well as to update the weights of the softmax layer as in Equation (2.3). The first of these intermediate gradients is the loss gradient of the output with respect to the totals, which is used to calculate the local gradients and is written as:

$$S = \sum_i e^{t_i}$$
$$\frac{\partial out_s(i)}{\partial t} = \begin{cases} \frac{-e^{t_c} e^{t_i}}{S^2} & \text{if } i \neq c \\ \frac{e^{t_c}(S - e^{t_c})}{S^2} & \text{if } i = c \end{cases} \tag{2.10}$$

Next the gradients $\frac{\partial L}{\partial w}$, $\frac{\partial L}{\partial b}$, and $\frac{\partial L}{\partial input}$ are calculated, with gradient $\frac{\partial L}{\partial w}$ being used to update the softmax layer's weights, $\frac{\partial L}{\partial b}$ to update the layer's biases, and $\frac{\partial L}{\partial input}$ returned

to be used as input to the previous layer during backpropagation. These are calculated as:

$$\begin{aligned} \frac{\partial L}{\partial w} &= \frac{\partial L}{\partial out} * \frac{\partial out}{\partial t} * \frac{\partial t}{\partial w} \\ \frac{\partial L}{\partial b} &= \frac{\partial L}{\partial out} * \frac{\partial out}{\partial t} * \frac{\partial t}{\partial b} \\ \frac{\partial L}{\partial input} &= \frac{\partial L}{\partial out} * \frac{\partial out}{\partial t} * \frac{\partial t}{\partial input} \end{aligned} \tag{2.11}$$

where

$$\begin{aligned} \frac{\partial t}{\partial w} &= input \\ \frac{\partial t}{\partial b} &= 1 \\ \frac{\partial t}{\partial input} &= w \end{aligned} \tag{2.12}$$

2.3 Federated Learning

Federated learning has been recently proposed as a learning strategy that leverages distributed computing across multiple devices in order to keep a user's training data private [27]. The idea behind this approach is that each participating device, often referred to as clients, uses its local training dataset to compute updates to a global model accessible by all clients. Instead of each client sharing its data with a global server during training, these clients share only their updates to the model, which can be combined to create a more accurate global model.

Federated learning has proven to be a successful strategy for training models across a large number of devices, with multiple examples found in mobile applications [3], including various image classification applications, language model applications used in voice recognition and next-word-prediction for touch-screen keyboards. Federated learning can be leveraged in these fields to create accurate global models and more personalized local models, which together can provide provide localized predictions on non-IID data [39].

However, federated learning has grown to encompass more than mobile machine learning and has proven useful as a parallelization method in use cases where data are collected from multiple sources by the same client [17]. In such cases where data privacy is not a concern, federated learning can be utilized to leverage distributed computing. In such a system, individual data collecting nodes can perform training on their local models in parallel, drastically reducing training times.

Different strategies have been proposed for combining local models into a global model [23], one example being federated stochastic gradient descent (FedSGD), where all the data from a random sample of nodes are used to compute a set of gradients. These gradients are then averaged into a global gradient by a server, proportional to the number of data samples provided by each node. A second approach is federated averaging, a generalization of FedSGD in which each node performs training on a single batch of training data and then exchanges the updated weights of its local model with the server to be averaged. In this strategy since the local model for each node is initialized the same, averaging the changes to the weights of each is the same as averaging the local gradients.

One of the important concerns with federated learning is the communication overhead incurred through transmitting network parameters between the clients and a server holding a global model. To address this issue, additional strategies for communicating updates that look to reduce the communication cost of transmitting large quantities of data between devices have been proposed [18]. Such strategies include restricting the dimensionality of shared parameters through the use of quantization and subsampling to limit the size of communications between clients and server.

2.4 Secure Multi-Party Computation

The privacy of user-held data is of major concern in many fields such as healthcare and banking, and as the prevalence of machine learning algorithms continues to grow, machine learning techniques that guarantee privacy are becoming increasingly more important.

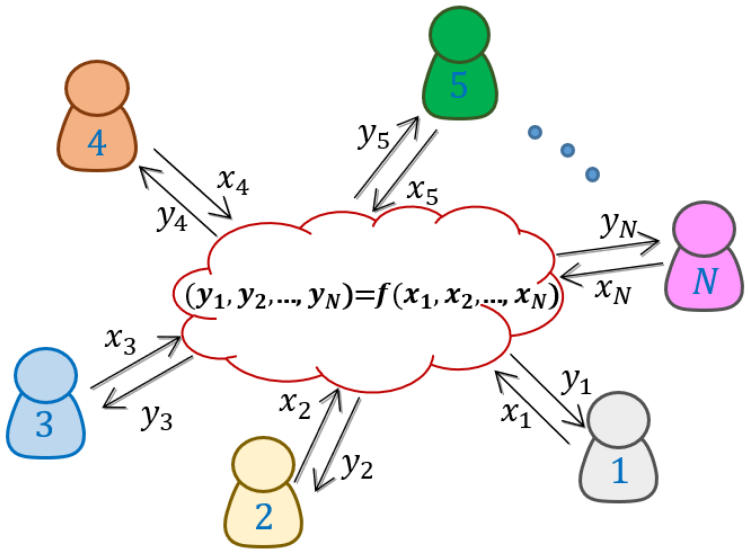


Figure 2.6: Example MPC secret-sharing scheme [22]

The ultimate goal of a privacy-preserving machine learning model is to keep all data used in the model encrypted such that no information about the training data or labels can be obtained without proper authentication nor indirectly accessed through model parameters such as the weights and biases used for individual layers. A privacy-preserving model should also be resistant to the influence of unauthorized outside forces through methods such as inference attacks.

Multiple methods have been proposed to create privacy-preserving machine learning models, with the basic principle of most solutions focusing the ability to perform computations between two encrypted values. By enabling computations in the encrypted domain, these strategies can leverage computation resources such as large cloud-based clusters as no plain-text is used in running the machine learning model. This approach, unlike other methodologies such as using AES encryption to send encrypted training values to a server running the model, requires that both parties need to be trusted with the data.

One encryption method that allows for arithmetic computations between encrypted values is secure multi-party computation (MPC). MPC enables encrypted computations through the use of a secret sharing scheme where a single piece of data is mapped to m

shares and split among m parties [13]. These shares are created such that these parties can perform arithmetic functions between two shares, and when recombined, the output will have the identical operations performed on it. Typically, these shares are created as random elements in a finite field that add up to the secret value modulo the size of the field. Each individual share, thus, appears randomly distributed in the field, and data security is maintained in an MPC system as long as all parties do not collude to reconstruct the original data.

2.5 Encryption Protocols

Multiple MPC encryption protocols have been developed over the years, each attempting to reduce the communication and computational overhead required for performing computations in the encrypted domain. Most of these protocols are based on a secret sharing scheme as described earlier rather than the circuit garbling protocols commonly found in two-party computation schemes [2]. While circuit garbling protocols such as Yao's [40] benefit from few rounds of communication between parties during computation, they are inherently limited to Boolean circuit evaluation.

One secure computation protocol that has received significant scientific attention recently is the SPDZ protocol [10], which has proven adaptable to multiple uses of secure computation as it supports as few as two parties. In addition, many optimization techniques have also been proposed to decrease the communication rounds between parties during computation. One such technique for decreasing inter-party communication is for certain values needed during computation to be pre-computed during an offline phase by the secret-sharing client [9]. This technique is explored further in Section 6.2.

Secrets used in this protocol exist in a finite ring of integers based on a large prime

Q and are created as:

$$s_i = \begin{cases} \text{random\_in\_range}(0, Q) & \text{if } 0 \leq i < N-1 \\ (d - \sum_{i=0}^{N-1} s_i) \ \% \ Q & \text{if } i = N-1 \end{cases} \tag{2.13}$$

where s_i is the ith share of data d for N total shares. To reconstruct plain-text data from N shares, the following is performed:

$$d = \left(\sum_{i=0}^{N-1} s_i \right) \ \% \ Q \tag{2.14}$$

As this form of secret generation is based in a finite set of integers, extra steps are needed to implement the SPDZ protocol for computations dealing with floating point values. One method is to convert these floating point values into fixed point values and then scale those into integers over a predefined range. Extra overhead, however, is added in this method as multiplication between two of these values requires re-scaling the output to ensure the correct decimal placement. Because of this technique, the SPDZ protocol has been proven adaptable for use in various machine learning algorithms [6].

Other protocols have also been proposed to deal with floating point computations in the secure domain without using fixed point representations for the encrypted data [1], one example being utilizing the binary representation of floating point values to create integer representations of a single floating point value. Implementation details for the encryption protocol used in this research are included in Section 4.3.

Chapter 3

Related Work

This chapter explores literature that relates to this research area, beginning with the various implementations for integrating secure data in machine learning, including the advantages and disadvantages of each. Next, the federated learning strategies that have been applied to secure learning algorithms are analyzed, followed by a comparison of the architecture implementations that utilize secure multi-party computation and details the shortcomings of previous works. The final section summarizes the chapter and provides comparisons between these related works and the work completed in this research.

3.1 Secure Machine Learning Techniques

The multiple use cases for secure machine learning exemplify the various approaches taken by researchers to implement secure machine learning algorithms. CryptoDL [15] adopted CNNs to facilitate homomorphic encryption (HE) by approximating activation functions with low-order polynomials. While the network was able to achieve accuracy results within 0.04% of the baseline model, CryptoDL relies on models trained on unencrypted data and only performs predictions in the encrypted domain. A second scheme found in [24] proposes using different HE schemes to identify images for intelligent transportation systems using cloud computing devices. This framework doubly encrypts each

image such that a secure server fully decrypts each image for training, and a non-secure cloud server subsequently performs classification on a partially encrypted image. Similar to CryptoDL, this method relies on using non-encrypted images for training but has the added complexity of requiring two separate servers to run the forwards and backwards phases of training/validation.

CryptoNets [12] also uses HE to encrypt a neural network similar to [15] and [24]. The disadvantages of this proposed method, however, involve efficiency and security as it assumes a server already has a trained model. Only the inputs and intermediate values are encrypted in this work, while the weights of the network are not encrypted to take advantage of more efficient HE multiplication schemes. However, this poses a privacy risk because according to [30] features of a network can be used to aid in adversarial attacks against a network as well as to provide details about the inputs and outputs of a network. SHE [25] utilizes the leveled fast HE over torus (LTFHE) encryption scheme to implement the ReLU activation function as well as the max pooling layers. This was an improvement over CryptoNet as it supports these extra layer types and, thus, can run on the ImageNet dataset, which, at the time it was developed, no other leveled HE model could.

These methods were all able to achieve high validation accuracy results; however, the use of non-secure pre-trained models impacts the overall security of a network if they are used on non-trusted devices. Utilizing encryption for classification only also limits the possible use cases of such networks as being able to train a model on encrypted data would allow for the utilization of larger non-secure computation sources such as pre-existing cloud computing solutions.

More recent work such as CryptoNN [38] look to address these previous limitations by performing both training and prediction using encrypted data. This proposed framework uses functional encryption which allows for partial decryption of data unlike HE where the data is either fully hidden or fully revealed. Using functional encryption, CryptoNN is able to perform both forward and backward passes through the network with encrypted data, enabling training on encrypted inputs and labels. While the ability to perform training

in the encrypted domain was a first for crypto based approaches, as opposed to secure multi-party computation based approaches covered in 3.3, a limitation of this work is that there are only two rounds of secure computations in the network. The first of these rounds occurs with the encrypted image during the forward propagation and the second with the encrypted label during backpropagation, meaning all resulting computations of the network are in plaintext. Thus, though the computations are more efficient, it results in privacy concerns as discussed previously.

3.2 Federated Learning with Encrypted Data

As federated learning requires exchanging model parameters among multiple participants, the possibility of leaking information about the training data is a security concern that needs to be addressed in systems where each participant prefers to keep its own data private. While different approaches to privacy-preserving federated learning have been explored previously in literature using a number of different encryption techniques such as differential privacy, HE, and secure multi-party computation, these solutions aim to guarantee data privacy, support high communication efficiency, and be resilient to participant dropout and multiple types of inference attacks.

One implementation of secure federated learning [35] uses secure multi-party computation techniques to encrypt the local weights of each device before they are aggregated into a global model. This weighted average of the encrypted weights is calculated by the data aggregator, with the added security of a differential privacy protocol being used to ensure that the resulting average cannot be used by colluding clients to re-create the weights of others.

A second approach [4] explores using secure aggregation, a class of MPC where a group of mutually distrustful parties collaborate to compute an aggregate value, to train a deep neural network with federated learning. More specifically, this work investigates possible solutions to such challenges faced by federated learning systems as participant dropout

and malicious users and defines a theoretical solution for a practical secure aggregation protocol. [5] extends this work by analyzing the computation and communication costs incurred by implementing a secure aggregation protocol to minimize the effect of user dropout and the overhead associated with the proposed model.

The framework proposed in [36] utilizes differential privacy (DP) to prevent information leakage of client data. This DP scheme adds artificial noise to each client's data before aggregation to ensure that the aggregator cannot learn the true nature of each client's local models. However, this method involves a trade-off between the performance of the network and the level of privacy protection gained from DP. As a result, its accuracy suffers in comparison to other approaches when ensuring the same level of security against differential attacks.

Hybrid Alpha [37] employs a functional encryption scheme to perform secure multi-party computation. This implementation aims to lessen the communication overhead incurred through both federated learning and MPC while at the same time being robust against honest-but-curious and colluding participants. The use of functional encryption compared to other schemes such as DP or HE was shown to reduce the training time of the model by 68%, while maintaining accuracy results similar to other MPC implementations.

3.3 Secure Multi-Party Computation in Machine Learning

The use of secure multi-party computation (MPC) protocols for distributed machine learning has been researched less frequently than similar HE implementations; however, the benefits of MPC compared to HE for certain applications has led to an emergence of additional research being conducted in this field. These benefits include the use of secret sharing that prohibits an attacker from re-creating the encrypted data without compromising every device in the system as well as lower error rates when performing computations in the secure domain. One of the primary limitations for HE-derived architectures is that most current implementations require training using plain-text data. If a model is trained on

encrypted data, however, the same encryption keys used during training must be reused during the inference stage, resulting in one of two scenarios, neither ideal: Either the keys are shared with a third party during the inference stage, meaning all data can be decrypted by a non-secure source, or the trained model can be used only by a data owner, limiting the scalability and use cases of such an architecture.

An additional disadvantage to using HE schemes for machine learning is the performance loss experienced when creating deeper architectures, caused by their limitations in performing concurrent arithmetic computations between two encrypted values without the error of each computation growing exponentially as well as the vanishing gradient problem introduced by using the easier to implement sigmoid activation function compared to ReLU.

One of the earlier implementations of using MPC for privacy-preserving machine learning is detailed in [32]. The architecture proposed in this work utilizes the SPDZ encryption protocol to train a CNN using two-party MPC connections. The model implements the convolutional, pooling and dense layers using SPDZ as well as an approximation of the ReLU activation function. The softmax layer was not implemented, however as it requires exponentiation, which SPDZ does not support. The disadvantages of the approach proposed in this work are the security risks due to not encoding the entire network as well as the loss of accuracy and immense overhead incurred from the secret-sharing scheme.

The framework created in [28] uses a specialized 3-party protocol that is tailored towards functions commonly used in neural networks as opposed to more general 3-party and MPC frameworks based on solving arithmetic or Boolean circuits. This MPC protocol was used in implementing a CNN that achieved state-of-the-art classification accuracy while achieving a 6-533$\times$ communication overhead reduction compared to approaches found in previous work. This reduction in communication overhead is attributed primarily to an improvement in computational complexity for non-linear functions as well as the decrease in the number of communications required after the offline phase.

3.4 Summary

This chapter analyzed the advantages and disadvantages of various implementations of privacy-preserving neural networks. This analysis of past research influenced the design and implementation of the model developed in the research presented here. The work presented in this thesis aims to build upon these works as follows:

1. Expanding upon the MPC scheme presented in [32] to facilitate the entirety of back-propagation to occur in the secure domain with no plain-text values visible to the network.

2. Implementing a basic version of floating point scheme presented in [1] to facilitate the necessary arithmetic functions needed to perform the backpropagation phase of training on secure values.

3. Building off of the ideas presented in [35] regarding the encryption of local weights before aggregating them into a global model to enable federated learning between encrypted values

Chapter 4

Experimental Design

This chapter covers the design and implementation details of the work conducted for this research. It begins by establishing the goals for this study as well as the hypothesis it intends to test. Then it details the datasets used in testing the validity of this research, followed by the details on the security protocol implemented in this work. The fourth and fifth sections provide the architecture designs used to test the baseline and secure versions of the image classification model, and it concludes with the types of measurements used to prove the validity of this study.

4.1 Hypothesis and Research Goals

The goal of this research is to enable the training of a CNN in an encrypted domain through the use of simulated MPC techniques. The use case of such a system would allow untrusted third-party servers to perform the computationally expensive parts of training, while trusted parties can make use of the decrypted model to perform testing on plaintext. To this end, a basic CNN was created in C++ to use as a baseline for testing, and a second version of the network was created that could make use of an encrypted floating point scheme in order to perform the backpropagation phase of training on encrypted values. Federated learning techniques were also incorporated into each model in order to weigh the

possible parallelization benefits for training against the validation accuracy losses incurred through distributed learning.

The research question posed in this work is whether or not the proposed security protocols are conducive to performing the large amounts of computations necessary for training in an encrypted setting. We propose that integrating the proposed security model into a CNN will have a slight negative impact on the accuracy of the network, with the majority of the impact coming from increased training times. We test this through comparing the validation accuracy achieved by the secure model against that of the baseline, as well as modeling the communication overhead in order to measure its impact on training times.

4.2 Datasets

Multiple datasets have been created over the years to test the efficacy of image classification techniques. These datasets are typically composed of a number of different classes, with each image belonging to a single class, for example an image of a cat or dog. Image classification datasets typically contain thousands to millions of individual images and may contain tens to thousands of distinct classes.

Images in a dataset are divided into either a training set or a validation set for testing, with each containing examples of all classes. The reasoning for dividing the dataset into these two subsets is for the image classification system to have access to the training set while learning to prevent the system from learning directly from the validation set. Doing so reinforces the system's ability to classify novel images as opposed to memorizing a solution for every given input. The size of the validation set relative to the size of the training set is a ratio that needs to be considered when tuning an image classification system as an overabundance of training samples can cause over-fitting to occur in the trained model. This over-fitting leads to the system learning only features present in the training set instead of broader features present in the entire dataset, thus causing a decrease in the validation accuracy of the system.

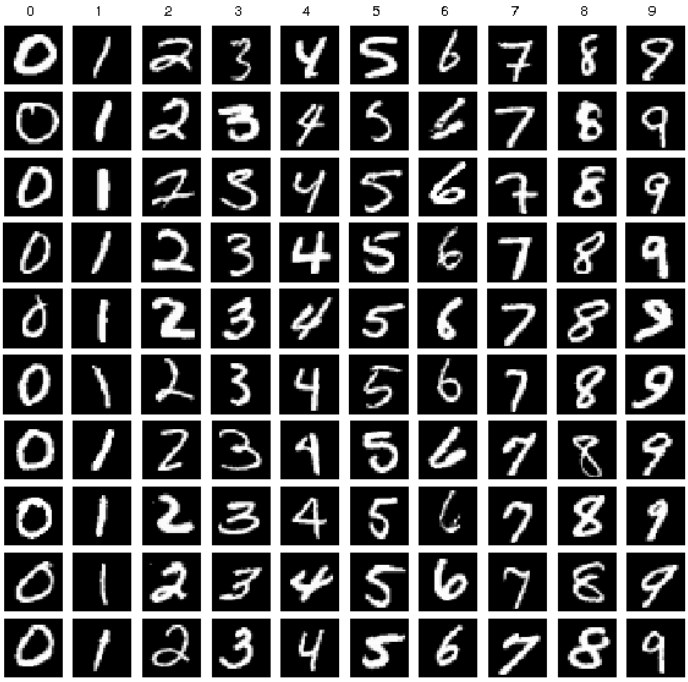


Figure 4.1: Example images from the MNIST dataset

The MNIST dataset is widely used in the field of image classification and is often used as a baseline to compare the performance of different image classification systems. The MNIST dataset contains 60,000 grayscale images of handwritten digits between zero and nine, with each image being normalized to 28×28 pixels [11]. It is typically split into 50,000 training images and 10,000 validation images when training a classifier. Both the baseline and secure architectures developed in this research were tested primarily using the MNIST dataset, and are discussed in sections 4.4 and 4.5.

Another popular image classification dataset, CIFAR-10, contains the ten classes of airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck [19]. It includes 60,000 32×32 pixel RGB images and is split into 50,000 training and 10,000 test images. While the classification networks created in this research were primarily created based on the MNIST dataset, the CIFAR-10 dataset was also used to test the validity of the proposed classifiers on more complex data.

Figure 4.2: Example images from the CIFAR-10 dataset

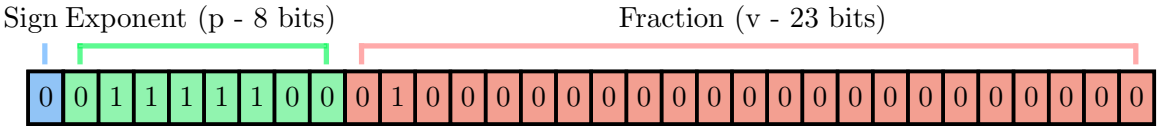


Figure 4.3: Standard 32-bit floating point representation

4.3 Security Model

The security model used in this work was significantly influenced by the protocol defined in [1], with several modifications made to fit the scope of this research. The work of Aliasgari et al. introduced methods for performing secure computations on floating point numbers using a secret sharing scheme similar to the protocols discussed in Section 2.5. This framework is based in a multi-party setting where each party is given an encrypted share of an input and jointly calculates the encrypted output of a given function.

These input shares are created from the standard representation of a floating point number as shown in Figure 4.3, which consists of a significand or base value v and an

exponent p, combined as follows to create the floating point number x:

$$x = (1 - 2s) * (1 - z) * v * 2^p \tag{4.1}$$

where s is a bit representing the sign of the number and z is the zero bit which is set to 1 when $x = 0$. Each floating point value x can thus be represented as a 4-tuple integer (v, p, z, s), where v is limited to l-bits and p is limited to k-bits such that $l + k$ equals the precision of the floating point value (i.e. 32 or 64-bit). For a 32-bit floating point value the range of v and p is thus limited to $v \in [2^{l-1}, 2^l)$ and $p \in (-2^{k-1}, 2k - 1)$. If $l = 24$ and $k = 8$, then the 4-tuple will have the same range as standard single-precision floating point values. The values of the 4-tuple (v, p, z, s) are calculated from an input x as:

$$p = \lfloor log_2(|x|) \rfloor - l + 1 \qquad v = \lfloor |x| * 2^{-1*p} \rfloor$$
$$s = \begin{cases} 1 & \text{if } x < 0 \\ 0 & \text{if } x >= 0 \end{cases} \qquad z = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

In the proposed encryption scheme, the 4-tuples are created by the data holder and then secret-shared to each participating party. As the values of the 4-tuple are all integers, the MPC encryption protocols previously explored such as SPDZ can be applied to facilitate encrypted integer arithmetic on the shared values. For the scope of this research, this added encryption was not implemented; however, each 4-tuple is treated as containing private values for the secure architecture detailed in Section 4.5.

The floating point operations detailed in [1] were simplified to eliminate the need to use the secure bit-wise operations necessary for fully secure arithmetic functions; however, the base algorithms for floating point addition, subtraction, multiplication and division were maintained to ensure computational correctness. While these simplified algorithms can still make use of added integer encryption schemes, certain functions such as exponentiation need to be modified further in order to work with such schemes.

The CNN used for the secure architecture requires only addition, subtraction, multiplication, and division to occur in the encrypted domain. As such, these are the only functions created to work with the encrypted floating point scheme. Of the four basic arithmetic functions, only addition, subtraction, and multiplication between two encrypted values are required as the only division that occurs in the network uses a constant value as the denominator. For this reason, division was implemented as simply a multiplication with the inverse of the denominator as the constant value is public and unencrypted.

These basic arithmetic functions were implemented to work for two encrypted inputs, i.e. $\langle [v_1], [p_1], [z_1], [s_1] \rangle + \langle [v_2], [p_2], [z_2], [s_2] \rangle$ (throughout this work variables denoted with square brackets can be further encrypted using a secret sharing scheme). When an operation is performed between an encrypted 4-tuple value and a non-encrypted floating point value, the non-encrypted value is first transformed into 4-tuple form before the arithmetic operation is performed.

Multiplication between two encrypted floating point values $\langle [v_1], [p_1], [z_1], [s_1] \rangle$ and $\langle [v_2], [p_2], [z_2], [s_2] \rangle$ to produce $\langle [v], [p], [z], [s] \rangle$ can be expressed mathematically as:

$$(1 - 2(s_1 \oplus s_2)) * (1 - (z_1 \vee z_2)) * v_1 * v_2 * 2^{p_1 + p_2} \tag{4.3}$$

However, extra steps are needed so that neither v nor p exceed its allowed ranges for a given l and k. This is to ensure that any given 4-tuple value may be converted back into a standard floating point value without overflowing the allocated bits used to store both the base and exponent values.

To ensure that the base value of the output fits within l bits, the product of v_1 and v_2 is shifted to the right l bits as shown in lines 1-2 below. Doing so can limit the precision of multiplications as the least significant bits are discarded; however, it is necessary to ensure that the encrypted value can be properly reconstructed. The value of p also needs to take this l-bit shift into account, and as a result l is added to the sum of p_1 and p_2 (Line 3). The sign and zero bits of the product are determined by the corresponding inputs as shown on

lines 4-5.

$\langle [v], [p], [z], [s] \rangle \leftarrow$ Multiply($\langle [v_1], [p_1], [z_1], [s_1] \rangle, \langle [v_2], [p_2], [z_2], [s_2] \rangle$)

---

1. $[t] \leftarrow [v_1][v_2]$
2. $[v] \leftarrow [t] >> l$
3. $[p] \leftarrow [p_1] + [p_2] + l$
4. $[s] \leftarrow [s_1]$ XOR $[s_2]$
5. $[z] \leftarrow [z_1]$ OR $[z_2]$

Addition between two encrypted 4-tuples $\langle [v_1], [p_1], [z_1], [s_1] \rangle$ and $\langle [v_2], [p_2], [z_2], [s_2] \rangle$ can be expressed as:

$$((1 - 2s_1) * (1 - z_1) * v_1 * 2^{p_1}) + ((1 - 2s_2) * (1 - z_2) * v_2 * 2^{p_2}) \tag{4.4}$$

Similar to multiplication, extra care needs to be taken to ensure that the values of v and p fit within their respective ranges. The value of the output exponential p should be scaled to either p_1 or p_2, whichever is greater, to ensure that the largest number of significant digits are retained through the summation (Line 3 below). Thus, the base value corresponding to the input with the smaller exponent needs to be scaled to match the larger exponent (Line 6 or 11, with the scaled base being stored in $[b]$). This scaled base is then checked to make sure it is within the range (INT_MIN, INT_MAX) determined as INT_MAX $= 2^l$. If the value of $[b]$ is outside of this range, then it needs to be re-scaled to fit within the range of appropriate values by modifying its corresponding exponent value as shown in lines 8-9 and 13-14. This updated base value $[t]$ is then added to the base of the other 4-tuple input while taking the signs of the original inputs into account (Lines 10 and 15).

$\langle [v], [p], [z], [s] \rangle \leftarrow$ Add($\langle [v_1], [p_1], [z_1], [s_1] \rangle, \langle [v_2], [p_2], [z_2], [s_2] \rangle$)

1. if ($[z_1]$) return $\langle [v_2], [p_2], [z_2], [s_2] \rangle$
2. if ($[z_2]$) return $\langle [v_1], [p_1], [z_1], [s_1] \rangle$
3. $[p] \leftarrow$ Max($[p_1], [p_2]$)
4. $[s] \leftarrow [s_1]$ if ($[p_1] \geq [p_2]$) else $[s_2]$
5. if ($[p_1] < [p_2]$) goto 11.
6. $[b] \leftarrow [v_2]$ / Pow($2, [p_1] - [p_2]$)
7. $[t] \leftarrow [b]$ if (INT_MIN $< [b] <$ INT_MAX), goto 10.
8. $[t] \leftarrow [v_2]$ / Pow($2, [p_1] - [p_2] + 1$)
9. $[p] \leftarrow [p] + 1$
10. $[v] \leftarrow |(1 - 2 * [s_1]) * [v_1] + (1 - 2 * [s_2]) * [t]|$, goto 16.
11. $[b] \leftarrow [v_1]$ / Pow($2, [p_2] - [p_1]$)
12. $[t] \leftarrow [b]$ if (INT_MIN $< [b] <$ INT_MAX), goto 15.
13. $[t] \leftarrow [v_1]$ / Pow($2, [p_2] - [p_1] + 1$)
14. $[p] \leftarrow [p] + 1$
15. $[v] \leftarrow |(1 - 2 * [s_2]) * [v_2] + (1 - 2 * [s_1]) * [t]|$
16. $[z] \leftarrow 0$

Subtraction between two 4-tuples in this scheme is performed as an addition between the same values by simply inverting the sign bit in the second operand. This is performed by creating a new 4-tuple $\langle [v_3], [p_3], [z_3], [s_3] \rangle$ such that $[v_3] = [v_2]$, $[p_3] = [p_2]$, $[z_3] = [z_2]$, $[s_3] = 1 - [s_2]$, and then adding $\langle [v_1], [p_1], [z_1], [s_1] \rangle$ and $\langle [v_3], [p_3], [z_3], [s_3] \rangle$.

4.4 Baseline Architecture

4.4.1 Design

The baseline architecture used for this research, as visualized in Figure 4.4, is a simple CNN created in C++ consisting of a single convolutional layer followed by a pooling

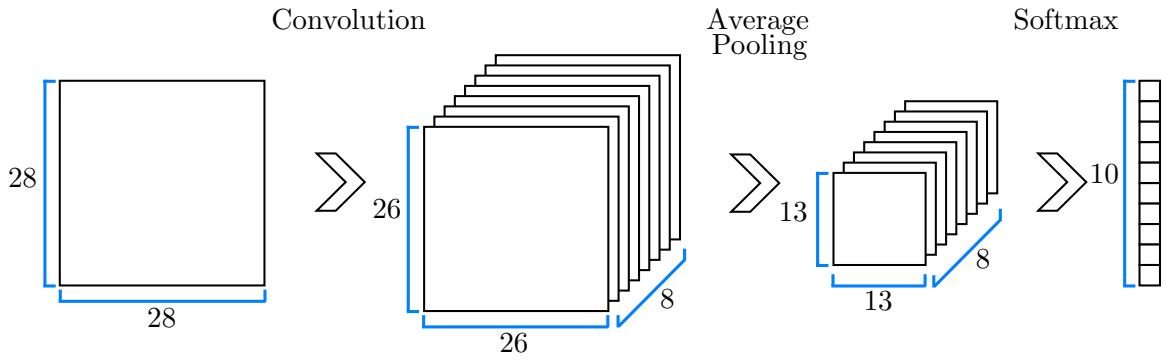


Figure 4.4: Baseline CNN architecture with 1 convolutional layer, 28 × 28 input image and eight 3 × 3 convolutional filters pictured

layer and a softmax layer as the output activation layer. This CNN was originally designed to be as simple as possible while still achieving relatively high validation accuracy on the MNIST dataset. The decisions to base this research on a basic CNN as well as to develop all of the network functions instead of using pre-made frameworks such as Tensorflow or Pytorch were made for a few key reasons.

First, the simplicity of this architecture facilitates an easier integration of the security model than if the secure architecture was based on current state-of-the-art CNNs. Second, while the validation accuracy of this architecture is lower than that of state-of-the-art models, the smaller size of the proposed model requires far fewer computations and as such requires only a fraction of the training time of more complex models. This decreased training time led to a more rapid testing cycle while creating and tuning the network.

The baseline architecture was built from the ground up to facilitate the intricate changes needed to integrate the security model described described in Section 4.3. Doing so instead of using a pre-existing machine learning framework also allowed for testing of individual components from the baseline and secure architectures to better measure the accuracy and timing differences between them. A preliminary version of the baseline model was created in Python; however, a speedup in both training and testing times of approximately 100× was noticed after porting this model to C++.

The limitations of the simplistic CNN became apparent when testing the CIFAR-10 dataset. Unlike the MNIST dataset, which contains fewer features present in a given

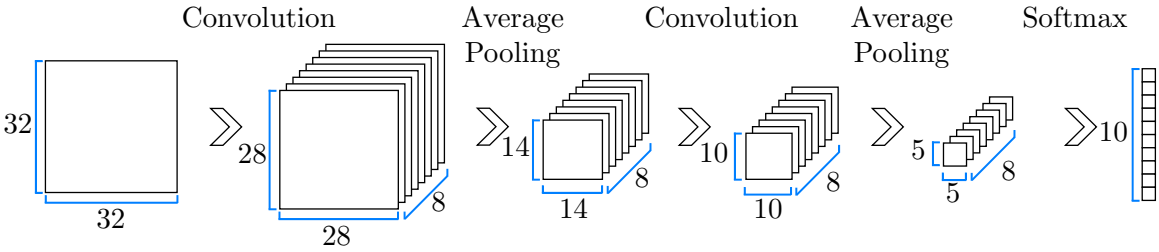


Figure 4.5: Modified baseline CNN architecture with 2 convolutional layers, 32 × 32 input image and eight 5 × 5 convolutional filters pictured

image, images in the CIFAR-10 dataset cover a much broader spectrum of possible features. Because of this added complexity and the reduction in validation accuracy incurred by using such a sparse network, a modified version of the 3-layer network was created as pictured in Figure 4.5. This modified network makes use of a second convolutional layer placed after the output of the first pooling layer, with a subsequent pooling layer placed before the final softmax layer. The performance and limitations of these two baseline architectures are discussed further in Chapter 5.

Multiple parameters used in the convolutional and pooling layers had to be initially decided upon to determine the shapes of each layer's inputs and outputs. These differ from the tuned parameters, or hyperparameters, that can be tested and modified in order to achieve the highest validation accuracy results for a given network configuration. The testing of these various hyperparameter configurations for each network is discussed throughout Chapter 5.

Convolutional layers extract features from an input through convolving said input with a set of filters. The number of convolutions performed between the input and a single filter is determined by the size of the filter and the stride, or number of pixels that the filter traverses between convolutions. Both the size and number of filters used in the convolutional layers were treated as hyperparameters in this research, and their effects on the training times and accuracy of the network are shown in Chapter 5. The stride of these filters, however, was selected to be a fixed parameter, and a stride length of 1 was selected so that every possible sub-region of an input image was convolved with the set of filters.

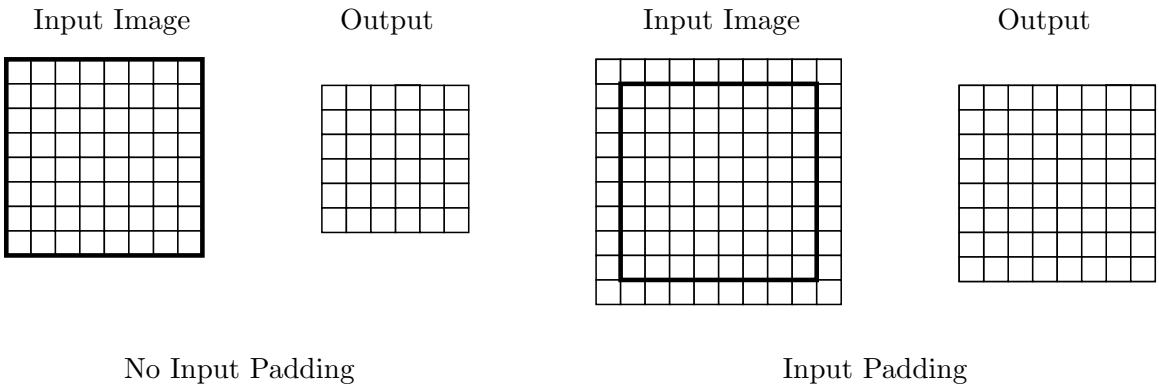


Figure 4.6: Effects of input padding for a 3x3 convolutional filter

Another key decision that had to be made when designing the convolutional layers for the network were whether or not padding should be used on the layer's input. Performing convolutions on an $M \times N$ image with a $F_M \times F_C$ filter results in the output being size $M-(F_M-1) \times N-(F_N-1)$. If this reduction in output dimensionality is not desired, then the input can be padded on all sides with F_M pixels, given the filter is of shape $F_M \times F_M$. Doing so leads to the output of the convolution having the same $M \times N$ shape as the original unpadded input, as seen in Figure 4.6. This similarity is preferred if the network includes a large number of layers as a continual shrinking of intermediate layers due to the inclusion of multiple convolutional layers limits the feature dimensionality subsequent layers as well as the total number of layers that can be utilized.

A pooling layer reduces the dimensionality of its input by propagating a single value calculated from a group of pixels forward to its output layer. These groups of pixels are determined by two layer parameters, the filter size and the stride of the filter. The filter size determines the amount of data reduction the pooling layer performs; for example, a 2×2 filter will reduce the output dimensions to one half of the input dimensions given a stride of 2 is used. The stride determines how the pooling filter moves across the input. If the stride matches the size of the filter, then each sub-region of the input is reduced a single time with no overlap in pixels. In this research, all pooling layers utilized 2×2 filters and a stride of 2 to achieve a 4:1 feature reduction throughout the network.

As discussed in Section 4.3, only the basic arithmetic functions are enabled in the

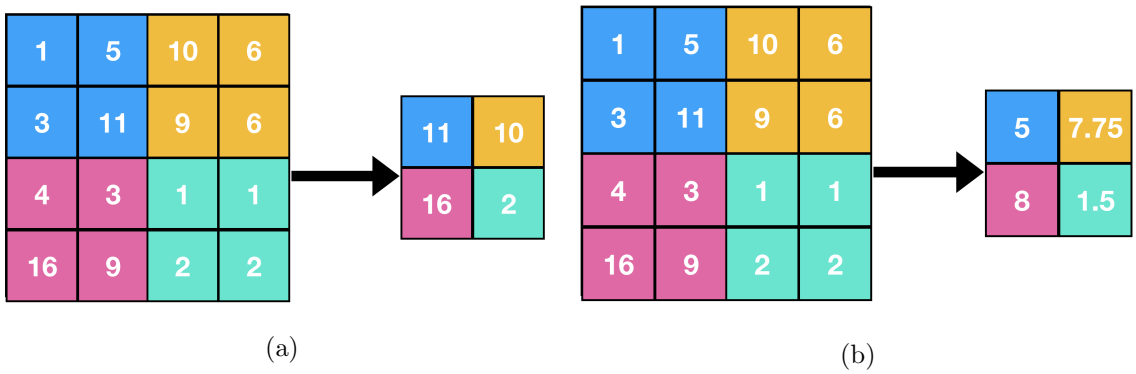


Figure 4.7: (a) Max pooling function with 2 × 2 filter and a stride of 2, (b) Average pooling function with 2 × 2 filter and a stride of 2 [33]

proposed security model. While this is adequate for implementing the backpropagation functions for convolutional and softmax layers, it limits the types of pooling layers that can be used in the network. A max pooling layer was originally used for the 3-layer architecture due to its well documented success for use in CNNs as often the maximum value in a group of pixels carries more meaning for a network than other metrics about that group. For example, if one pixel in that region has a value well outside the range of the other pixels, then often it correlates to a prominent feature present in the input.

As comparisons between two 4-tuple encrypted values are not possible in the proposed security model due to possible data leakage, a different pooling function needed to be implemented in the baseline architecture to ensure the layout of the secure and non-secure versions of the architecture were kept identical. Because of this an average pooling layer was used as it maintains the same feature reducing ability as a max pooling layer, the only difference being that the average of a group of pixels is propagated rather than the maximum. As a result, less important features have more impact on the network as a whole as opposed to the most prominent features, resulting in a slight loss in validation accuracy of the network. Examples of these two pooling functions are shown in Figure 4.7a and 4.7b, and the differences in validation accuracy resulting from the use of max pooling versus average pooling in the baseline architecture are shown in Chapter 5.1.

The initialization of the weights used by both the convolutional and softmax layers

often determines how well a network is able to learn features and, as such, how well the network performs in classifying images. In this work, each weight was initialized to a random floating point value between 0 and 1, and then scaled based on the number of weights used in that layer. This scaling is necessary as not every weight should contribute equally to the performance of a network, and having too many weights close to 1 can cause instability in the network. To address this issue, the following scaling was applied:

$$\begin{aligned} w_{conv} &= \text{random\_in\_range}(0,1)/(F_N * F_M) \\ w_{softmax} &= \text{random\_in\_range}(0,1)/(N_{filters} * M_{softmax} * N_{softmax}) \end{aligned} \tag{4.5}$$

where the convolutional layer uses $N_{filters}$ total filters of size $F_N \times F_M$, and the input to the softmax layer is of size $M_{softmax} \times N_{softmax}$. These scaling factors were determined empirically through initial testing of the baseline architecture.

4.4.2 C++ Implementation

The layers used in the baseline CNN were all created as classes in C++, with layer parameters stored as class variables and the various forward and backpropagation functions implemented as member functions. Organizing the layers in this manner facilitated linking them together to form a network in a more concise manner, and it allowed for each layer to be tested individually during development. These layer functions were linked to create the baseline architecture by routing the output of one function as the input to the subsequent layer function. For example, the output of the forward propagation function of the convolutional layer was used as input to the average pooling forward propagation function. In backpropagation the direction that each layer occurs is reversed so the output, or gradient, from the pooling layer is then used as input for the convolutional backpropagation function.

The two datasets used in testing the baseline architecture are comprised of either grayscale or RGB images. The data represented as a single pixel in these images is stored as either a single 8-bit value in the case of grayscale images or three 8-bit values corresponding

to the red, green, and blue channels of the pixel. As such, the data for a given pixel is in the range $[0, 255]$, which is well outside the range for useful computations within the network. Typically neural networks deal with floating point data within the range $(-1, 1)$ as consecutive arithmetic operations in this range cannot result in values growing to infinity. To address this issue, each input image was normalized to the range $[-0.5, 0.5]$ before being fed into the network. These normalized pixel values were calculated as:

$$p_{i \text{ normalized}} = \frac{p_{i \text{ input}}}{255} - 0.5 \tag{4.6}$$

Distributed learning was incorporated into the baseline architecture through federated averaging. This was accomplished through the use of an averaging function that is called after each node in the distributed learning model has finished training on its batch of images. This function averages each node's local weight values for each layer into a global set of weights that is then sent to each node. These new global weights are subsequently updated individually by each node until a second batch is completed. This process of federated averaging continues until all of the images in the training set have been exhausted. For the scope of this research, the nodes participated in the federated averaging scheme run in series on a single device instead of in parallel across multiple devices.

Both the number of images contained in a single batch as well as the total number of nodes performing training were treated as hyperparameters in this research, and the effects of modifying these values can be found in Chapter 5. The same set of initial weights are distributed to each node participating in training; however, it is possible that individual nodes would achieve better results if they are given different sets of weights at the beginning of training. Exploration of this modification and other possible improvements to this distributed learning technique can be found in Section 6.2.

4.5 Secure Architecture

The goal of the secure architecture developed here is to perform training in an encrypted domain where no plain-text data is needed to properly train the model. This was accomplished by integrating the security model described in Section 4.3 into the backpropagation phase of training where the network's weights are updated. The security model could also be integrated into the forward propagation phase of training in theory if additional functions between secure values are implemented. This would allow for the entirety of training as well as testing to be performed on encrypted data, but this added complexity was outside the scope of this research. Future improvements and incorporation of the security model are described in Section 6.2.

The secure 4-tuple scheme was integrated into the baseline architecture by first creating modified backpropagation functions for each of the layer types. This was accomplished by utilizing a class structure to store the 4-tuple floating point values and using operator overloading to modify how arithmetic functions operate between two variables. Utilizing C++ classes in this way made implementing secure versions of the backpropagation functions easier as only the data types used in a function had to be modified, not the core algorithms of each function.

Creating a network from these modified layer functions requires converting the plaintext floating point data into secure 4-tuple values at the beginning of the backpropagation phase. This encryption occurs after the initial loss gradient has been calculated as in Equation (2.9), the encrypted version of which is used as input for the secure softmax backpropagation function. All of the intermediate values required to perform backpropagation have to be converted as the aim of the secure architecture is to prevent any form of data leakage during training. This includes not only the inputs and outputs for each layer function, but also the layer weights as well as values calculated during the forward propagation phase of training such as the last input to a layer. Other than the added overhead required for encrypting and decrypting each value used in training, the secure network is implemented

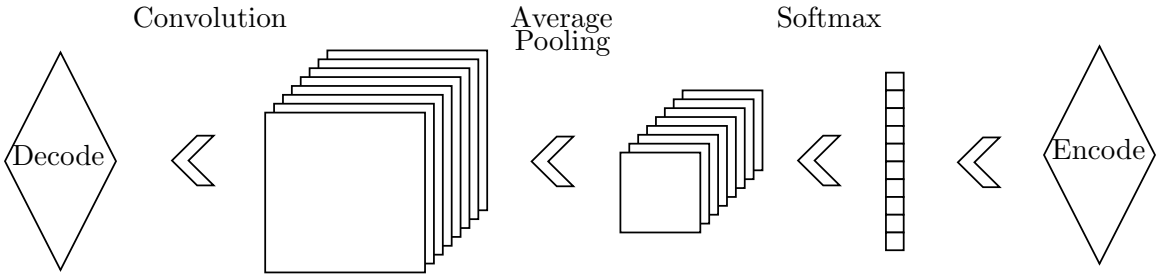


Figure 4.8: Secure Backpropagation

similarly to the baseline model as a set of concurrent layer functions.

Once the backpropagation for an image has finished, the updated encrypted weights and biases have to be decrypted in order to use them again during the forward propagation of the next training image. As the forward section of training is assumed to occur on a trusted device, these updated values no longer have to remain encrypted. This general functionality of the secure architecture is shown in Figure 4.8.

The federated averaging scheme described in Section 4.4.2 was incorporated in the secure architecture through the use of a function that computes the averages of each node's secure weights and biases. This federated averaging function was implemented to take the plain-text weights and biases used in the forward propagation of training as input and compute the averages of these values on encrypted versions of them. For a fully secure distributed learning model, it is assumed that each node participating in training is performing computations on a non-secure device and as such would require the entirety of its learning algorithms to run in the secure domain. This would require a slight modification to the secure averaging scheme as the inputs and outputs would remain in the secure domain during training, reducing the overhead incurred from encrypting and decrypting each value during averaging. This added network complexity is explored in Section 6.2.

4.6 Measurements

The primary metrics used for validation of the work completed in this research are the classification accuracy found when testing a network configuration, the execution

times of the various layer functions, and the overhead incurred from the use of the use of the security model as well as the federated learning techniques employed. The validation accuracy for a given network configuration is calculated as the percentage of images correctly classified during testing compared to the total number of images in the validation set. Training accuracy was an additional metric measured in this research; however, it does not fully describe the performance of a classifier as training can result in over-fitting of weights to the features only present in the training set. This can cause the training accuracy to be several percentage points higher than the testing accuracy, and is not indicative of better network performance. These results for the baseline and secure architectures are presented in Chapter 5.

Chapter 5

Results

This chapter documents the results gathered from testing different network configurations and hyperparameters used during training. Section 5.1 covers the results gathered from testing on the MNIST dataset and details the baseline network results in 5.1.1 and the secure network results in 5.1.2. Section 5.2 covers the results gathered from testing on the CIFAR-10 dataset with sections 5.2.1 and 5.2.2 covering the baseline and secure architecture results respectively. Section 5.3 details the communication overhead required by the security model in order to perform training. All results detailed in this section were obtained by running the networks on 2.33GHz processors with 12GB of RAM, and detailed timing results for each test are included in appendix A.

5.1 MNIST

The MNIST dataset was used exclusively during the development of both the baseline and secure architectures as its simplicity allowed for a sparse network to be developed while still achieving relatively high validation accuracy results. The simplicity of the dataset also allowed for faster development iterations as both training and testing on this dataset are less computationally expensive than larger and more complex datasets. This faster development cycle facilitated rapid modification and testing of individual layer functions for

improving the accuracy and optimization of the developed networks.

5.1.1 Baseline Architecture

This section covers various hyperparmeter and network configurations tested for the baseline architecture training on the MNIST dataset. Table 5.1 shows the baseline hyperparameters that were selected as the control group for measuring the impact of varying the different hyperparameters. Explanations for these values are detailed later in the section when discussing the impact of varying each parameter independently.

Table 5.1: MNIST Baseline Hyperparameters

Training Epochs	Learning Rate	Filter Size	Number of Filters
8	0.005	3×3	8

The baseline architecture originally contained a max pooling layer instead of an average pooling layer as discussed in Section 4.4.1. This original network configuration was found to have the highest validation accuracy of all of the architectures trained on the MNIST dataset. These results, shown in Figure 5.1, validate the theory that a max pooling layer is better able to propagate the most important features extracted by the convolutional layer than an average pooling layer. However, average pooling layers were used in all subsequent versions of the baseline and secure architectures because, as stated previously, average pooling layers the proposed security model does not support the comparisons between encrypted 4-tuple values that would be required to implement the backpropagation of weights through a secure max pooling layer.

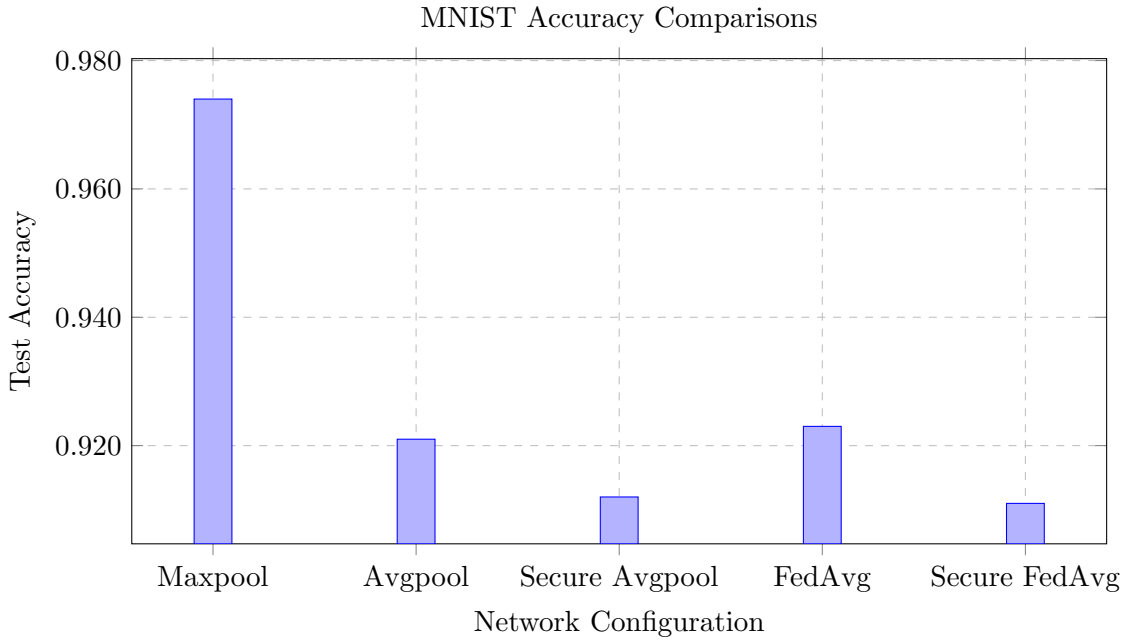
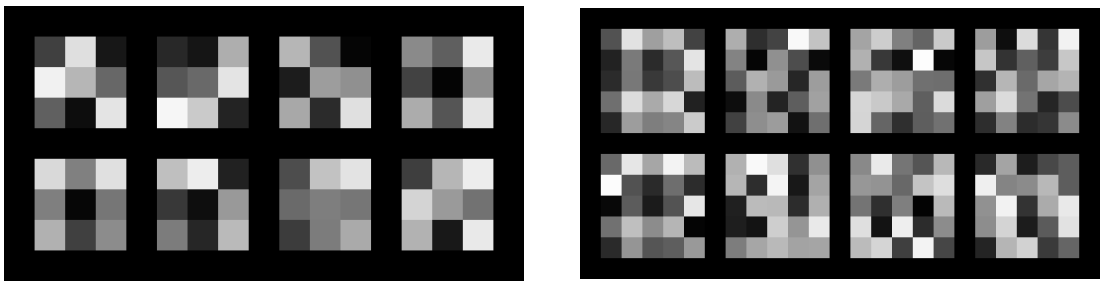


Figure 5.1: The hyperparameters used in this test were: 0.005 learning rate, eight 3×3 convolutional filters, 8 training epochs, and 4 nodes / 100 images per batch for the federated averaging schemes



(a) (b)

Figure 5.2: Convolutional layer filters for the MNIST dataset colorized for (a) 3×3 filter size (b) 5×5 filter size

The training of a CNN can be visualized by viewing the weights used in the convolutional layer. These filter weights are used to extract features from the input image, and visualizing them can give insights on what types of features a network was able to learn. A visualization of the convolutional layer filters for the baseline 3-layer network can be seen in Figure 5.2 for both 3×3 and 5×5 filter sizes. In both cases only eight filters were used

to train the network, and they were colorized in the figure by making the maximum value of each filter brighter in the image and the minimum values darker. Neither of these visualized filters display features that correspond to typical human-tuned filters commonly used in image processing, such as filters that determine edges or shapes in an image, showing that the network was able to learn features from the dataset from a random initialization.

One of the most important hyperparameters that determines how well a classification network learns is the number of epochs that it trains before performing validation testing. A training epoch in this research consists of performing forward and backpropagation of each image through the network in order to update the weights of each layer. Performing numerous training epochs in a row allows the network to train on each image multiple times and generally leads to an increase in validation accuracy as the network is able to better learn the features present in the training set. This isn't always the case, however, as overfitting may occur in which the network weights reflect only the data in the training set and not the entire dataset as a whole. Other examples that can cause a decrease in validation accuracy to occur as more training epochs are performed are explored in Section 5.1.2.

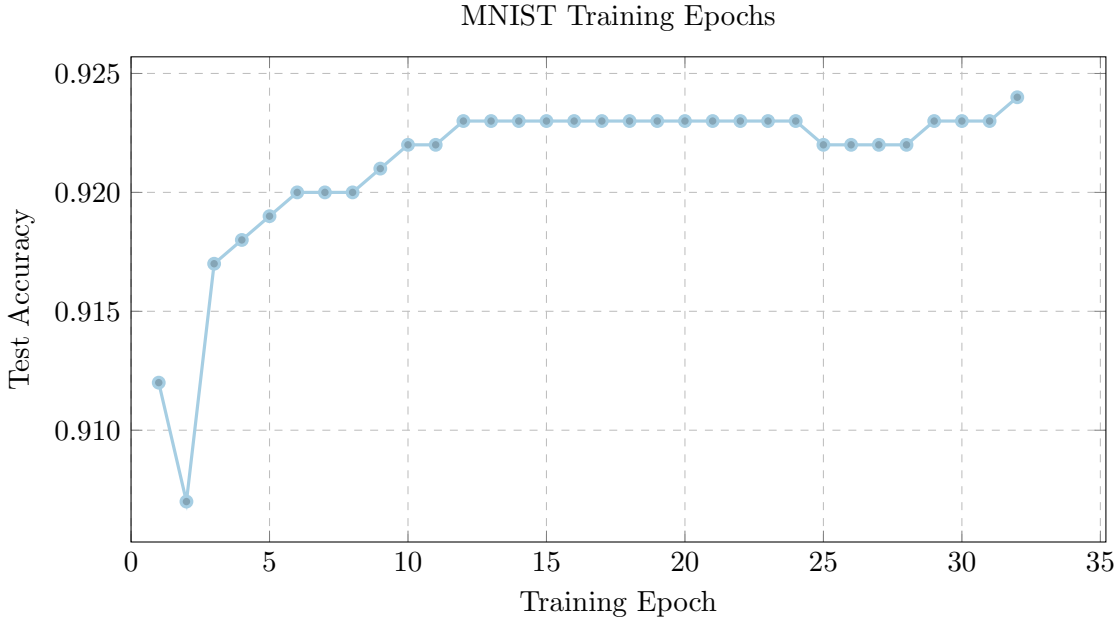


Figure 5.3: The hyperparameters used in this test were: 0.005 learning rate, and eight 3 × 3 convolutional filters

The effect of performing multiple training epochs was investigated by testing on the validation set after each epoch had completed. For typical experiments, all training epochs finish before a single round of validation testing occurs; however, conducting testing after each epoch allows for viewing the validation accuracy of a network over the course of training. Conducting testing in this way has no impact on the learned values of the network as testing of an image only performs the forward propagation functions and does not update any layer weights or biases. Figure 5.3 shows how the validation accuracy changes over time for the 3-layer baseline architecture, showing a reduction in learning occurs after twelve training epochs.

While the maximum validation accuracy of 92.4% is achieved after training for 32 epochs, the plateauing of improvement that occurs earlier makes the use of extra training epochs essentially redundant for our testing purposes as the extra 0.1% improvement in accuracy comes at the cost of tripling training time. Due to this large computational cost for little accuracy improvement, a baseline of 8 training epochs was used throughout this research as typically the validation accuracy of the network is still improving at this point in training, and using a smaller number of training epochs allows for training to occur over a more reasonable time-frame.

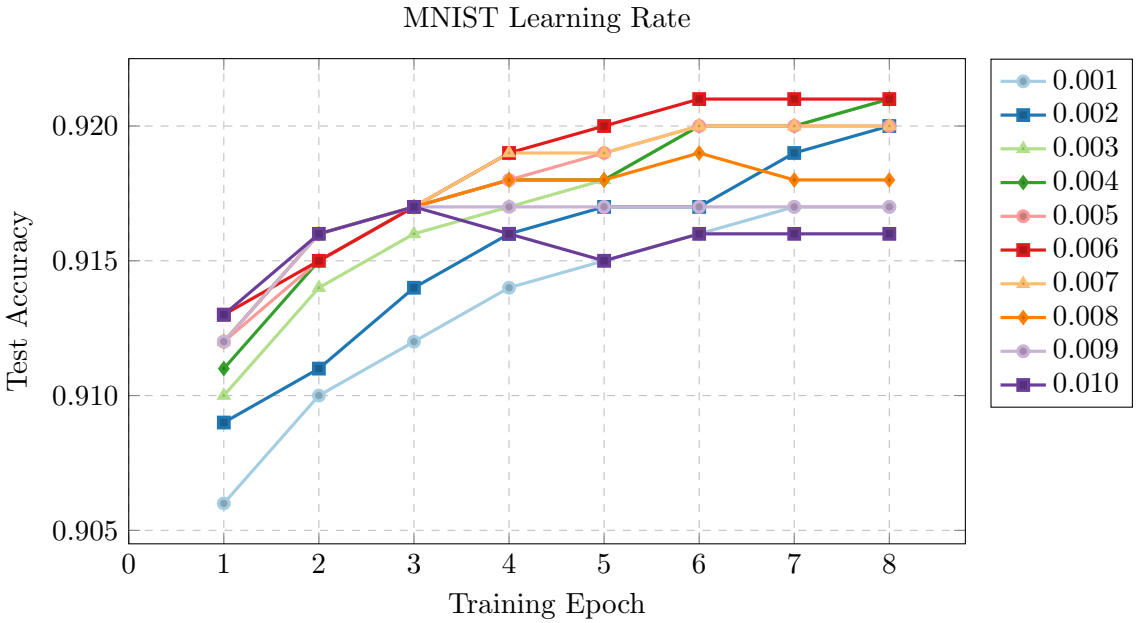


Figure 5.4: The hyperparameters used in this test were: eight 3×3 convolutional filters, and 8 training epochs

The impact of the learning rate used while training was measured similar to the number of training epochs, meaning the validation accuracy of the network was tested after each epoch. The learning rate of a network determines the magnitude of updates to the weights and biases of each layer due to the loss gradient during backpropagation as represented by Equation (2.3). Larger learning rates allow for these values to change more drastically over a shorter number of training cycles, whereas smaller learning rates lead to a network learning in a more gradual manner. In this research learning rates ranging between 0.001 and 0.01 were tested, the results of which can be seen in Figure 5.4.

After eight training epochs, it was noticed that both larger and smaller learning rates achieved a lower validation accuracy than learning rates in the middle of the tested range. Due to this trend, a learning rate of 0.005 was used as the baseline when testing the impact of other hyperparameters on the accuracy of the network.

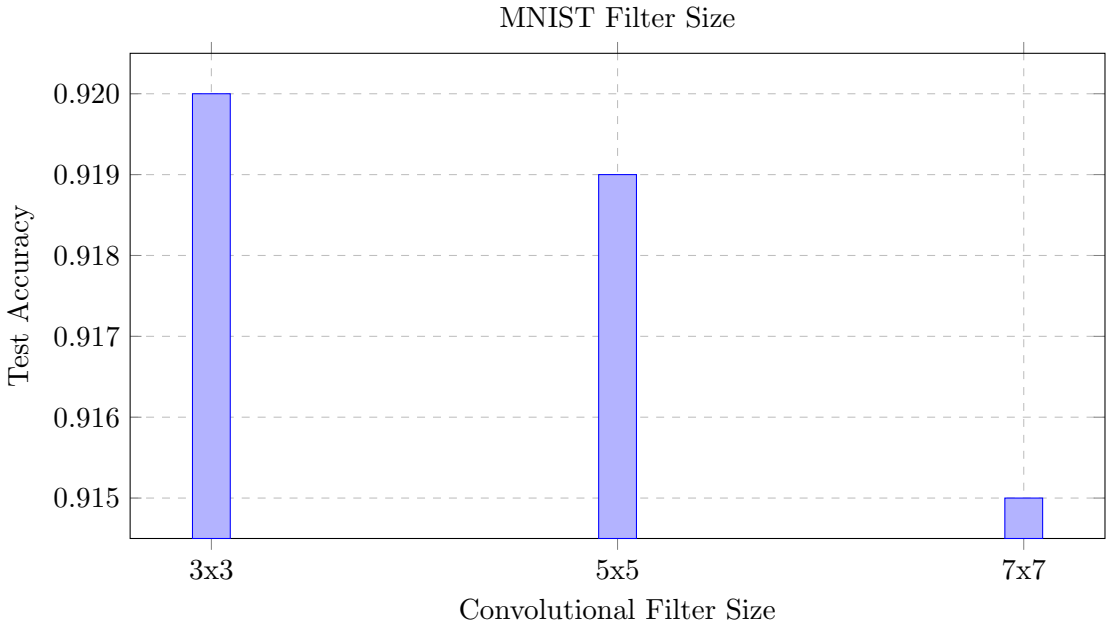


Figure 5.5: The hyperparameters used in this test were: 0.005 learning rate, eight convolutional filters, and 8 training epochs

The size of the filters used in the convolutional layer dictates the size of features that can be extracted from an input image. As padding was not used in this study, this filter size also dictates the size of each layer placed after it, and as such the number of learnable weights and biases of those other layers. Figure 5.5 compares the three filter sizes that were tested, with the 3×3 filters displaying the best performance. For datasets with larger images, a convolutional filter this small typically would not be able to properly detect meaningful features, but the small size of the images in the MNIST dataset prove to contain more learnable features at this smaller size for this sparse network configuration. Utilizing a smaller filter size also directly impacts both the training and testing times for the network as fewer computations are required compared to larger filter sizes. This reduction in computation time can be seen in Table 1.

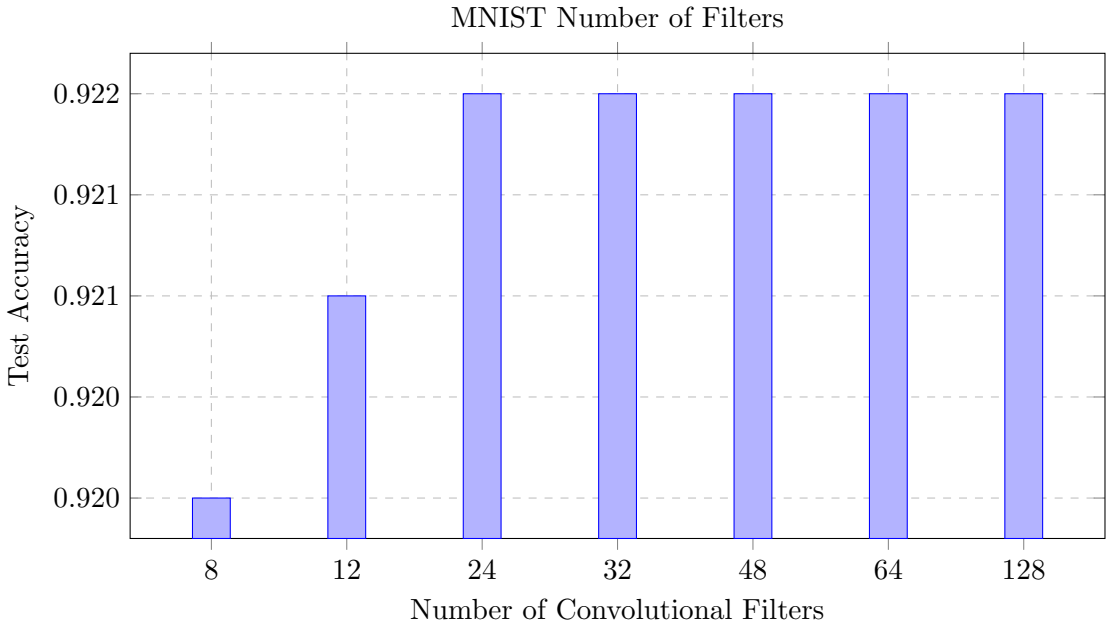


Figure 5.6: The hyperparameters used in this test were: 0.005 learning rate, 3×3 convolutional filters, and 8 training epochs

The number of filters used in a convolutional layer determines the number of unique features that a network can learn from a dataset. As such, using a small number of filters often prohibits a network from learning all available features, thus negatively impacting the validation accuracy of the network. In the case of the MNIST dataset, however, there is a small range of features present across the dataset as each image is simple in nature. As a result, a much lower number of convolutional filters are required to properly train a network, suggesting smaller networks may be used to achieve similar validation accuracy as larger networks training on more complex datasets.

The effect of the number of filters used on validation accuracy was tested for a range of 8-128 as shown in Figure 5.6. Increasing the number of convolutional filters beyond 24 had no apparent effect on the test accuracy of the network after eight epochs, and the effect of using more than 8 filters was negligible, with a total difference in test accuracy of only 0.2%. Similar to the size of the convolutional filters, the number of filters used in the convolutional layer directly impacts the size of each subsequent layer in the network.

Because of this, the total computation time required to train or test the network is directly proportional to the number of convolutional filters used as shown in Table 2.

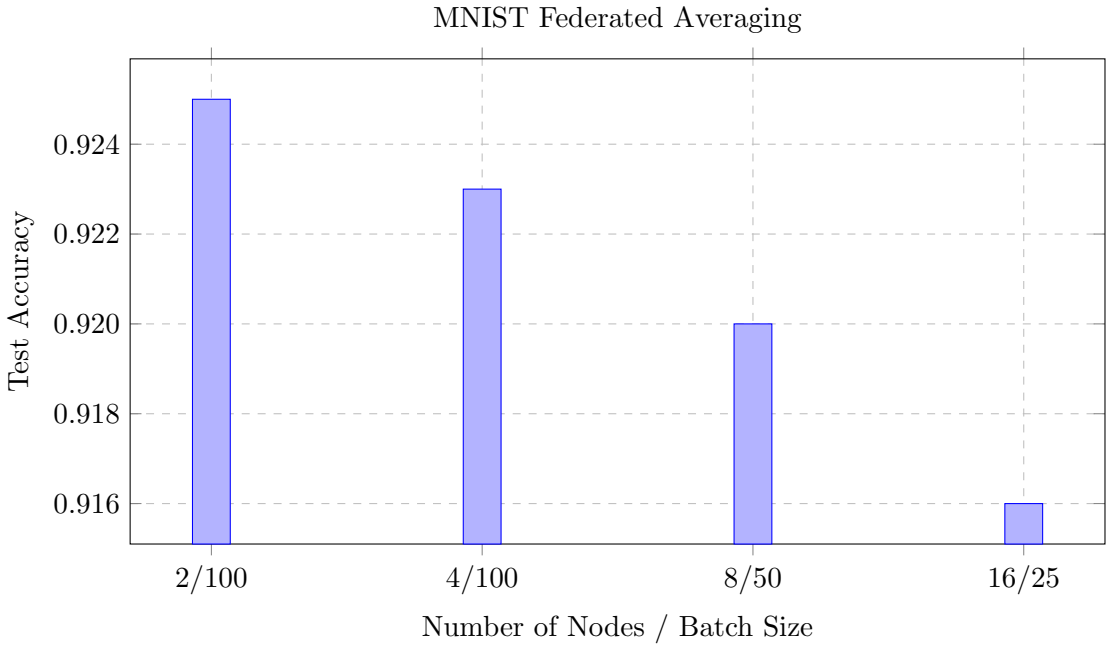


Figure 5.7: The hyperparameters used in this test were: 0.005 learning rate, eight 3×3 convolutional filters, and 8 training epochs

The inclusion of federated averaging in the baseline architecture was tested using a range of simulated training nodes as well as varying batch sizes. The number of nodes tested varied from 2-16, and the batch sizes were selected for these configurations such that the network never trained on more than 400 images per averaging cycle. This value was selected as the total training set size of 50,000 is divisible by it and utilizing a smaller batch size increases the rate at which the global model is able to learn. The accuracy results from this testing displayed in Figure 5.7 show that as the number of training nodes increases, the validation accuracy of the network decreases. This trend is due to the lack of image variety that each node has access to during a single averaging round, leading to less performant training of local weights.

The trade-off for this increased accuracy is that the portion of the training set distributed to each node is larger for smaller node counts, limiting the parallelization benefits of

training with federated averaging as the training time for each node is directly proportional to the number of nodes as seen in Table 3. As the number of nodes increases, the overhead incurred by the federated averaging algorithm also increases; however, this overhead is small in proportion to training times for all but 16 nodes.

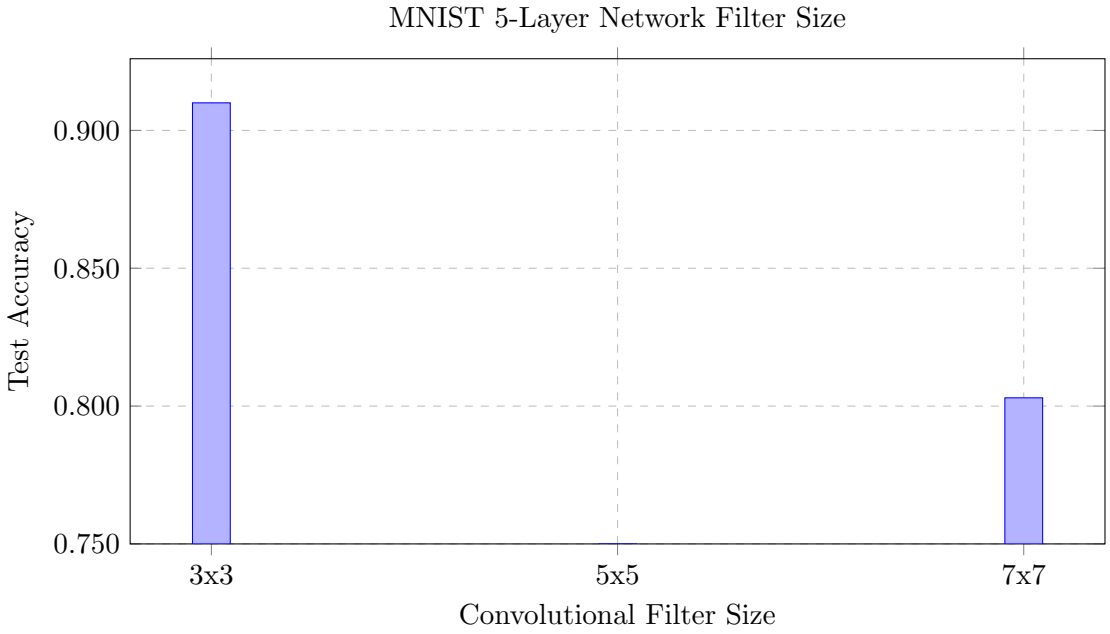


Figure 5.8: The hyperparameters used in this test were: 0.005 learning rate, eight convolutional filters, and 8 training epochs

Although a 5-layer version of the baseline architecture was created initially in an attempt to increase the validation accuracy achieved on the CIFAR-10 dataset, it was tested on the MNIST dataset as well. This network configuration uses two convolutional layers, each followed by an average pooling layer with a softmax layer supplying the network's final output. The results from the testing showed that this more complex network was unable to achieve a higher validation accuracy than the basic 3-layer network for all tested hyperparameter configurations, with certain configurations even causing the network to fail training entirely.

The first hyperparameter tested with this 5-layer network was the size of the convolutional filters shown in Figure 5.8. Both the 3×3 and 7×7 configurations resulted in

a decrease in validation accuracy, and using 5 × 5 filters caused the network to fail training before reaching 8 epochs. This training failure occurred due to the layer weights inflating to infinity as no activation functions were used to prevent this from occurring. Possible future solutions to this issue are provided in section 6.2. Timing results for this test are provided in Table 5, with the convolutional and pooling layer times combined for the two instances of each layer.

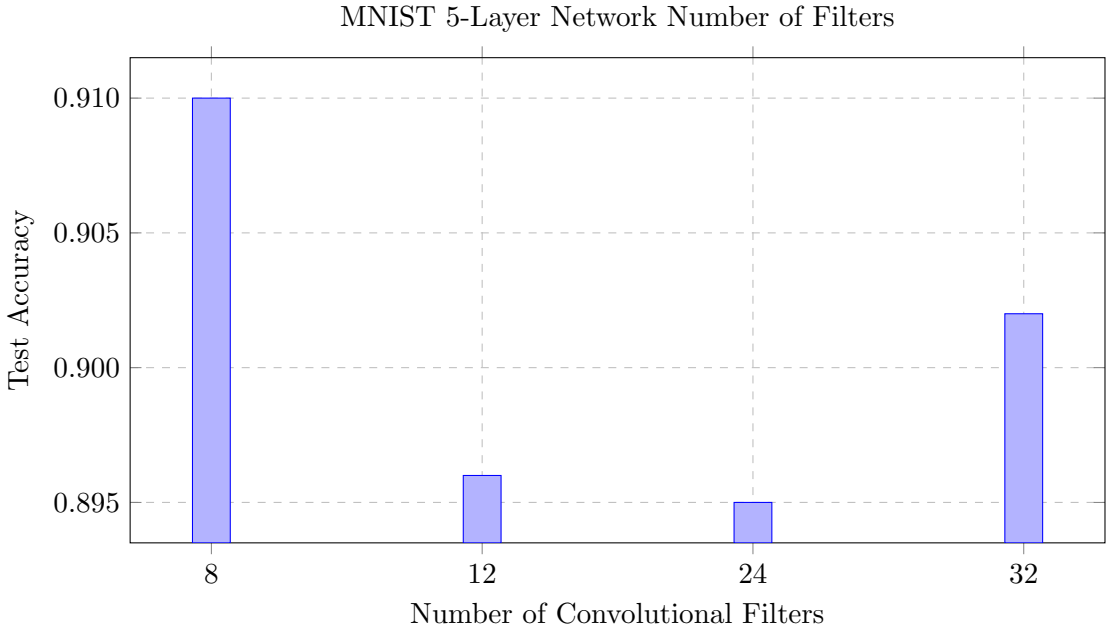


Figure 5.9: The hyperparameters used in this test were: 0.005 learning rate, 3 × 3 convolutional filters, and 8 training epochs

The second hyperparameter tested with the 5-layer network configuration was the number of convolutional filters, the results being shown in Figure 5.9. Similar to the testing of the size of the filters, validation accuracy was lower for this network in comparison to the 3-layer network using the same number of filters. Timing results for this test are included in Table 5 in Appendix A.

The decrease in accuracy of the 5-layer baseline architecture compared to the 3-layer architecture is likely due to a number of factors. As stated previously, neither configuration included activation functions to help prevent over-fitting and weight inflation, and it is

possible that this addition to the network would increase the validation accuracy achieved with the larger network. The testing also found that the performance of the 5-layer network was highly dependent on the random initialization of each layer's weights, and it is possible that different initializations of these weights may provide additional accuracy. Due to this poor performance, all of the secure architectures used in this research were based on the 3-layer architecture instead of integrating the security model into both network configurations.

5.1.2 Secure Architecture

The secure 3-layer architecture was tested using the same baseline hyperparameters as the non-secure architecture as listed in Table 5.1 with the exception of the number of training epochs being 4 for most cases instead of 8. This modification was made as training would fail for certain test cases after 5 to 7 epochs as shown in Figure 5.10. This training failure is due to the rounding error incurred from performing numerous additions and multiplications between two 4-tuple secure values, which can cause a weight inflation similar to that observed with the 5-layer network. Possible solutions to this error introduced by the security model are explored in Section 6.2.

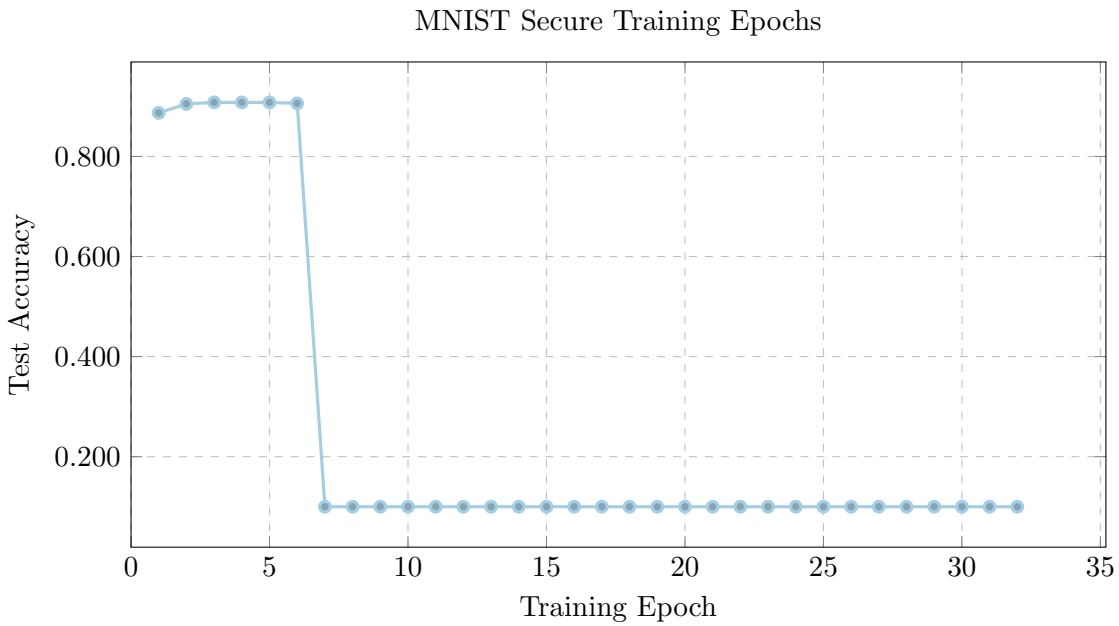


Figure 5.10: The hyperparameters used in this test were: 0.005 learning rate, and eight 3 × 3 convolutional filters

The impact of learning rate on validation accuracy was tested for the secure architecture similar to the baseline architecture as shown in Figure 5.11. As shown in this figure, the different learning rates tested had a large impact on the training performance of the network. Larger learning rates led to earlier training failure as the weights used in the network inflated to values that caused the final output loss to grow to infinity. This is apparent in Figure 5.11 as the validation accuracy drops off the graph after a certain number of training epochs. While several learning rates such as 0.001 displayed better accuracy over time than the baseline 0.005 learning rate, a learning rate of 0.005 was used for subsequent testing of the secure architecture to provide more consistent comparisons to the baseline architecture.

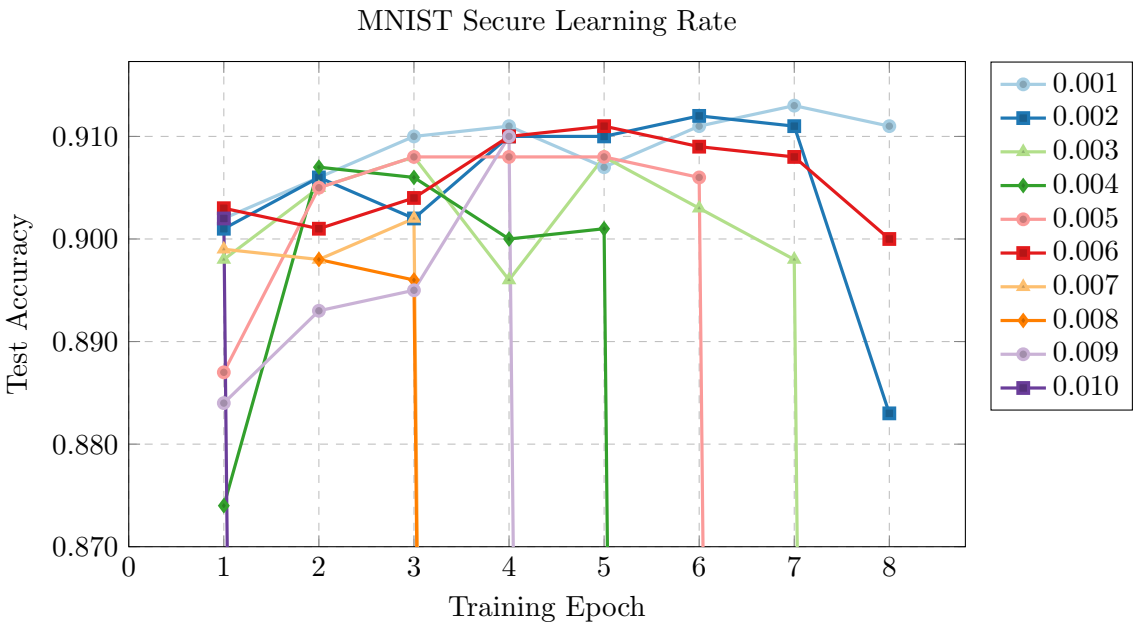


Figure 5.11: The hyperparameters used in this test were: eight 3 × 3 convolutional filters, and 8 training epochs

The impact of the size of the convolutional layer filters on validation accuracy could not be measured for the secure architecture due to the 5 × 5 and 7 × 7 filter tests failing during the first training epoch. This result is again due to the inflation of layer weights caused by the calculation error incurred from performing numerous computations in the secure domain. Performing one convolution with a 5 × 5 filter versus a 3 × 3 filter requires almost three times as many additions and multiplications between 4-tuple values to occur, expediting the rate at which these errors accumulate. As the 3 × 3 filter achieved the best validation accuracy on the baseline architecture, this issue is not as critical when comparing the overall performance of the baseline and secure architectures.

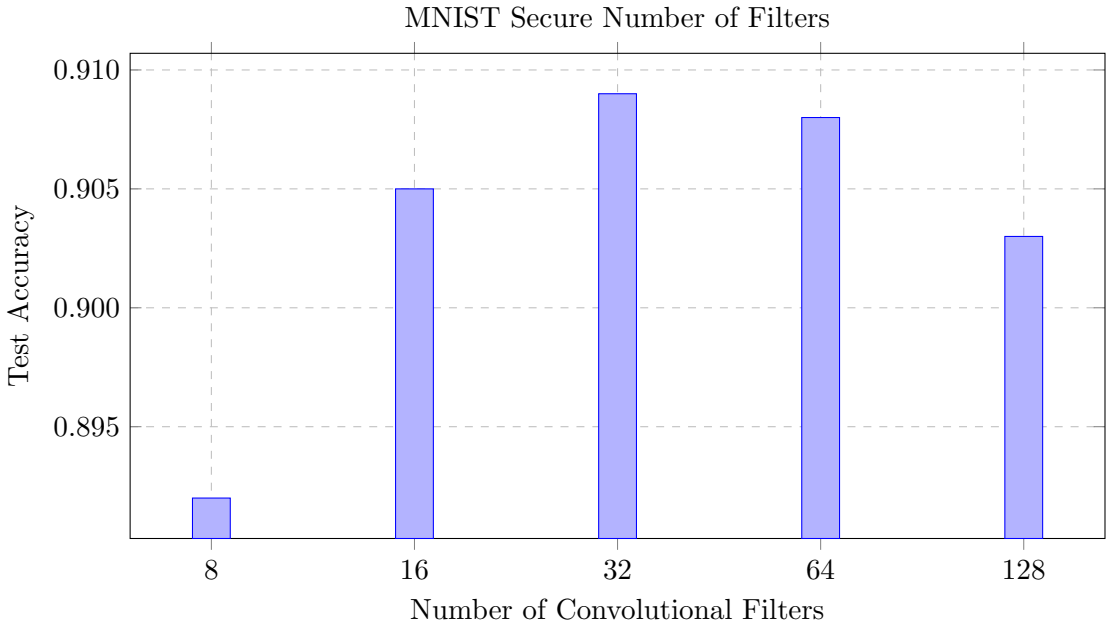


Figure 5.12: The hyperparameters used in this test were: 0.005 learning rate, 3 × 3 convolutional filters, and 4 training epochs

The number of filters used in the convolutional layer was tested for the secure architecture, with the results shown in Figure 5.12. The results were found to be similar to those for the baseline architecture, where utilizing more filters in the network leads to a general higher validation accuracy, albeit to a greater extent for the secure implementation. A possible inclusion of these larger quantity of filters with a lower learning rate could further prove to increase the validation accuracy for the secure architecture and reduce the accuracy variation between the baseline and secure networks.

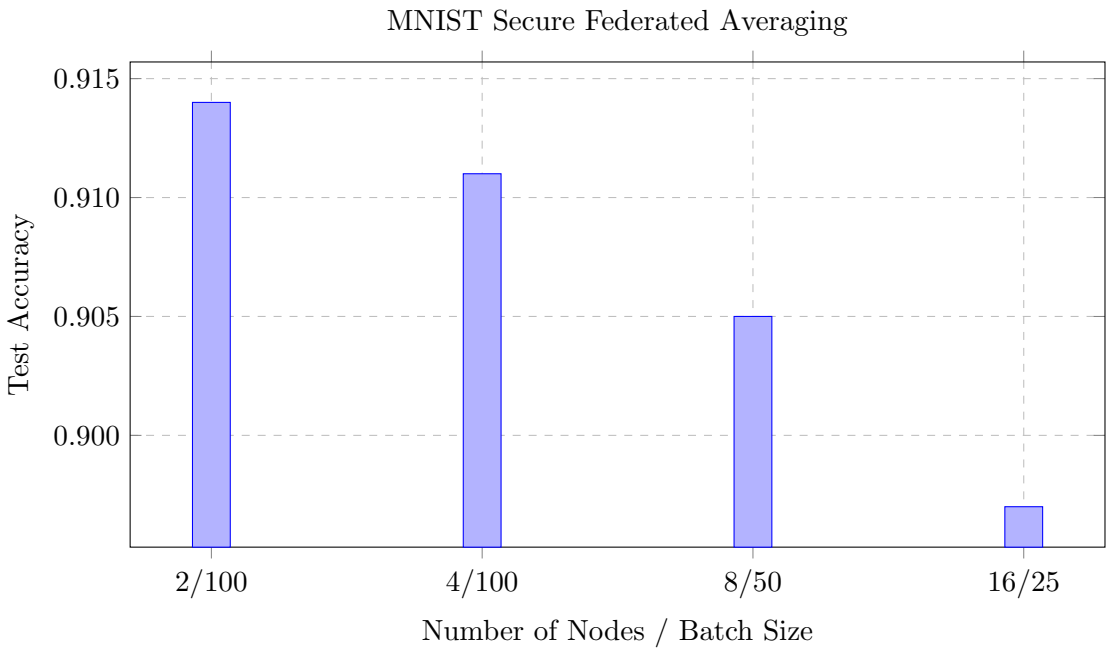


Figure 5.13: The hyperparameters used in this test were: 0.005 learning rate, eight 3×3 convolutional filters, and 8 training epochs

The same node counts and batch sizes were used to test the secure federated averaging scheme as the baseline federated averaging scheme. A similar effect on the validation accuracy was measured, with the accuracy decreasing as the number of nodes participating in training increased as can be seen in Figure 5.13. This decrease was slightly more substantial for the secure federated averaging scheme, likely due to the additional rounding errors incurred by performing the averaging scheme in the secure domain as well as the training. One result of note is that when performing this set of tests, the secure federated averaging network was able to train for a full 8 epochs without failure, perhaps due to a variation in the order of which each image is fed through the network as the initial weights of each layer were identical to the baseline implementation.

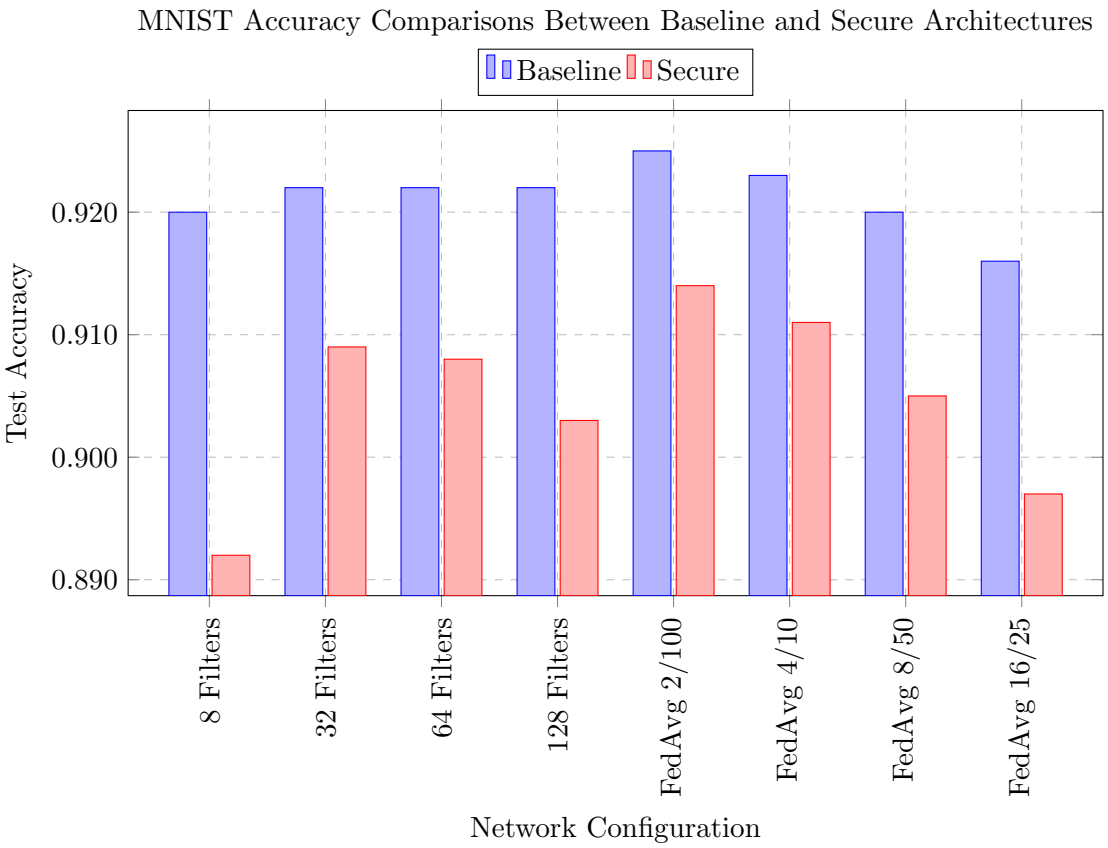


Figure 5.14: The hyperparameters used in this test were: 0.005 learning rate, 3×3 convolutional filters, and 8 training epochs

Figure 5.14 compares the accuracy between the baseline and secure architectures for different hyperparameter configurations. The difference in validation accuracy between these two architectures ranges from 1.1-2.8%, with the notable exception that the tests varying the convolutional layer filter sizes trained for only 4 epochs in the secure implementation instead of the 8 epochs that the baseline model trained for. Further improvements to the security model that could reduce this accuracy difference across all network configurations are detailed in section 6.2.

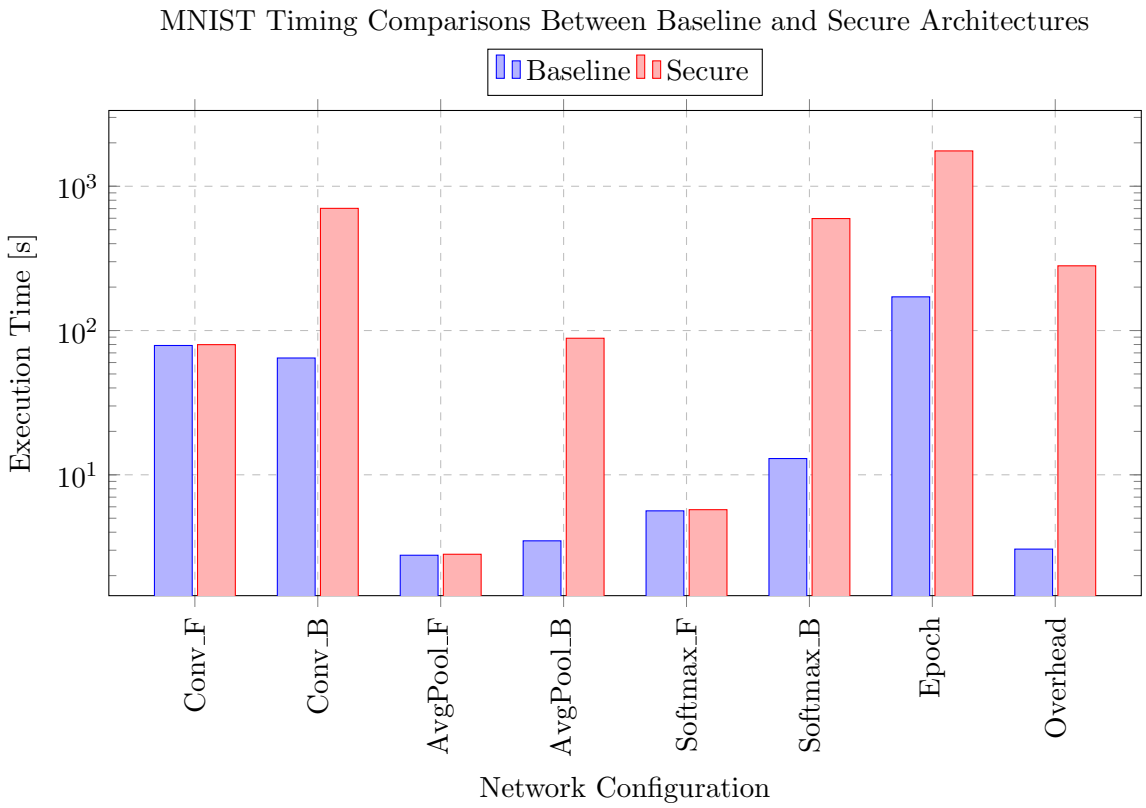


Figure 5.15: The hyperparameters used in this test were: 0.005 learning rate, eight 3×3 convolutional filters, and 8 training epochs

Figure 5.15 shows the differences in execution times for each layer function in the baseline and secure networks for a single training epoch. The inclusion of the security model in training did not impact the forward propagation functions of any layer, and as a result, the execution time of these functions remains unchanged between the baseline and secure implementations. The backpropagation functions, on the other hand, required between 11-46× as long to complete in the secure domain. These timing results do not take into account any communication time that would be required for a real-world implementation of the secure architecture and are only influenced by the added computational complexity of performing arithmetic functions between two secure 4-tuple values. Simulated communication overhead is detailed in Section 5.3.

5.2 CIFAR-10

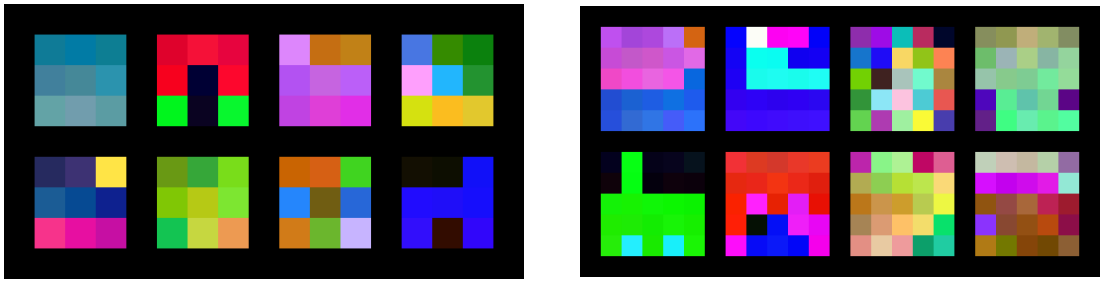
The CIFAR-10 dataset was used for training the developed networks to test the validity of the secure architecture on more complex data. As the images in this dataset are RGB instead of greyscale like MNIST, the computational complexity of the convolutional layer functions are three times greater, testing the ability of the secure architecture to learn without weight inflation. The images in this dataset also contain noisy backgrounds as opposed to the black backgrounds in the MNIST dataset, and as such the extraction and learning of relevant features are much more challenging. Because of these challenges, achieving a validation accuracy similar to the MNIST implementation with this dataset is beyond the scope of this research as more complex CNNs are typically required to achieve high validation accuracy.

5.2.1 Baseline Architecture

This section covers the various hyperparmeter and network configurations tested for the baseline architecture training on the CIFAR-10 dataset. Table 5.2 shows the baseline hyperparameters selected for the baseline network for this dataset. Similar to the development in section 5.1.1, the reasoning behind these baseline values will be explained later in this section.

Table 5.2: CIFAR-10 Baseline Hyperparameters

Training Epochs	Learning Rate	Filter Size	Number of Filters
8	0.001	3×3	8



(a) (b)

Figure 5.16: Convolutional layer filters for the CIFAR-10 dataset colorized for (a) 3 × 3 filter size (b) 5 × 5 filter size

Figure 5.16 visualizes the convolutional layer filters after training for one epoch on the CIFAR-10 dataset. As each filter has a red, green and blue channel, this visualization was created by normalizing the filter weights to values between 0-255 by assigning the maximum value among the three channels to 255 and the minimum to 0. Similar to the visualizations of Figure 5.2, none of the filters necessarily corresponds to human-readable values; however, they show that the network is indeed learning features from the dataset.

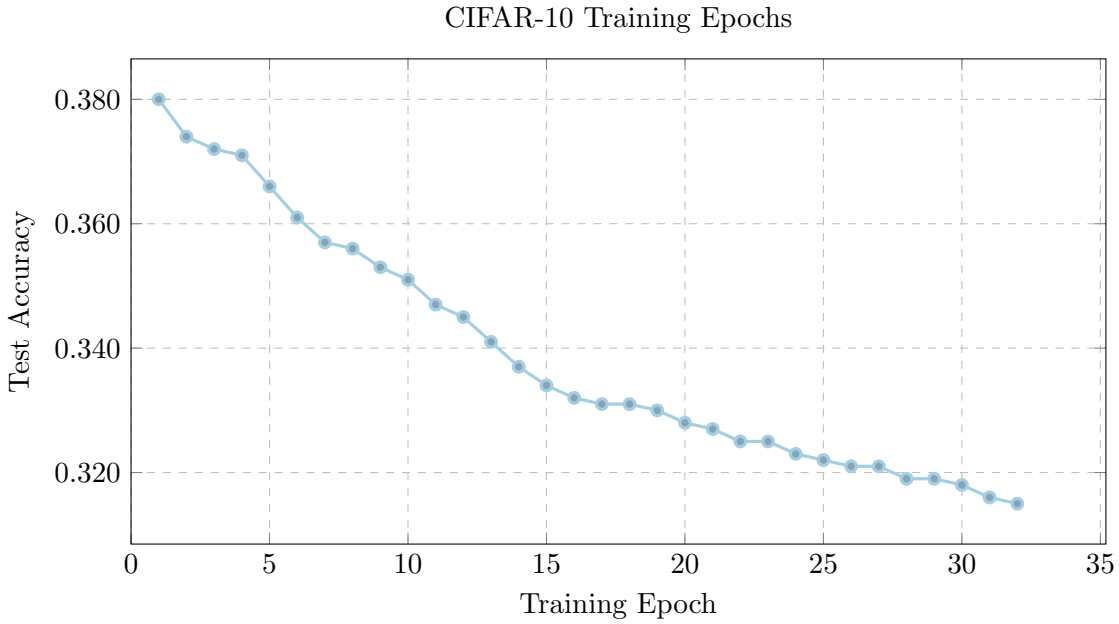


Figure 5.17: The hyperparameters used in this test were: 0.005 learning rate, and eight 3 × 3 convolutional filters

Figure 5.17 shows the validation accuracy of the baseline CNN for the CIFAR-10 dataset after each training epoch utilizing a learning rate of 0.005. However, unlike Figure 5.3 which shows an increase in validation accuracy as the network performs more training epochs, the validation accuracy of CIFAR-10 with these network parameters decreases for each subsequent epoch. This is an example of over-fitting the training data, an issue that occurs as the network updates layer weights and biases to learn features present in the training data, but not present in the entire dataset. This highlights the fact that network parameters often have to be hand-tuned to a particular dataset for training to occur effectively.

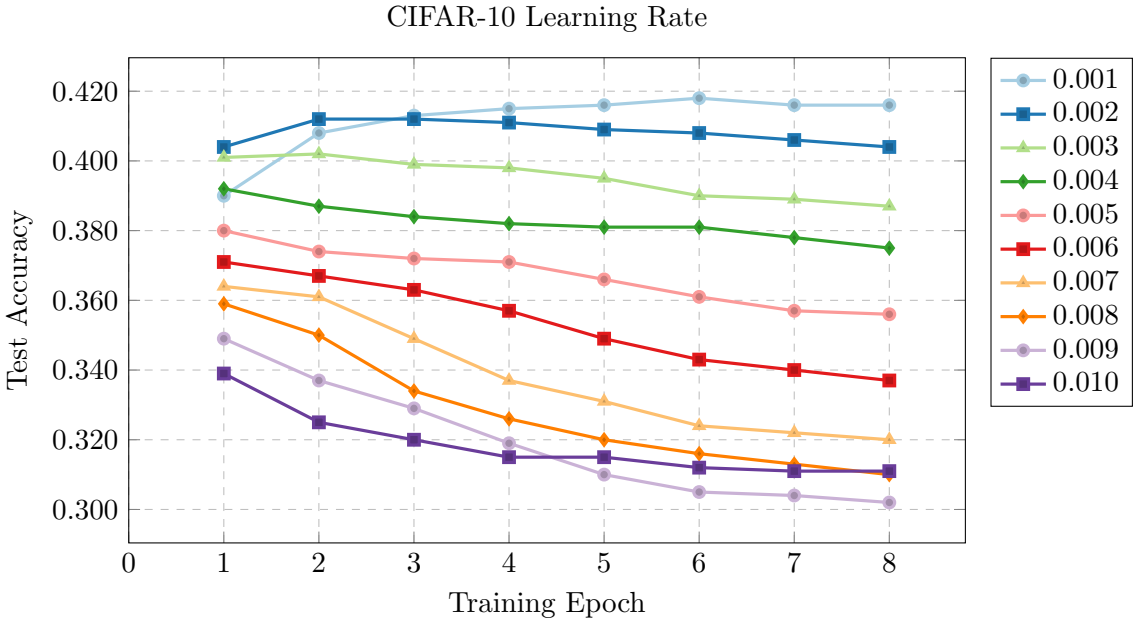


Figure 5.18: The hyperparameters used in this test were: eight 3×3 convolutional filters, and 8 training epochs

The learning rate used in training had a much greater effect on the overall accuracy of the network for the CIFAR-10 dataset than for MNIST. Figure 5.18 shows an almost 12% difference in validation accuracy between the range of tested learning rates after 8 training epochs, with all other hyperparameters remaining constant among tests. This difference also highlights that hyperparameter selection can have a large impact on network performance

as all but two of the learning rates tested over-fit the training data and did not exhibit an increase in accuracy over time. For this reason, a learning rate of 0.001 was used as a baseline for testing other hyperparameters as this value led to the best validation accuracy for the dataset.

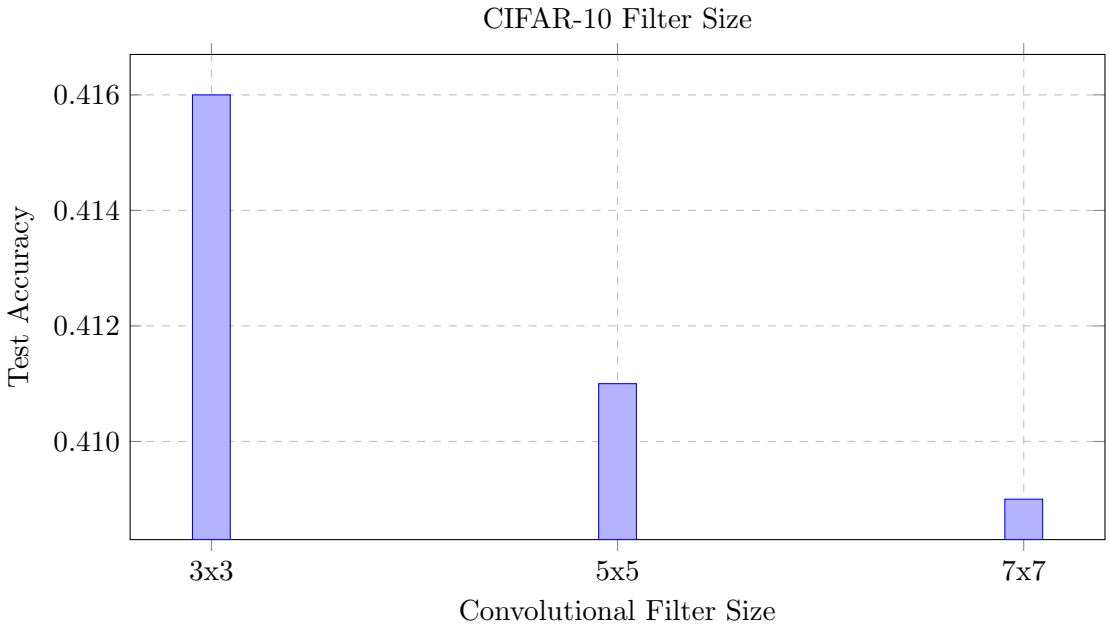


Figure 5.19: The hyperparameters used in this test were: 0.001 learning rate, eight convolutional filters, and 8 training epochs

The same filter sizes used in the convolutional layer were tested on the CIFAR-10 dataset as with MNIST to determine whether an increase in validation accuracy resulted from utilizing various filter sizes. Likewise, the number of convolutional filters was also tested as it was hypothesized that more filters would be needed for this dataset due to the larger variety of features present in the data. After testing both of these hyperparameters, however, it was determined that modifying either the size or number of convolutional layer filters had little impact on the validation accuracy of the baseline architecture as shown in Figures 5.19 and 5.20.

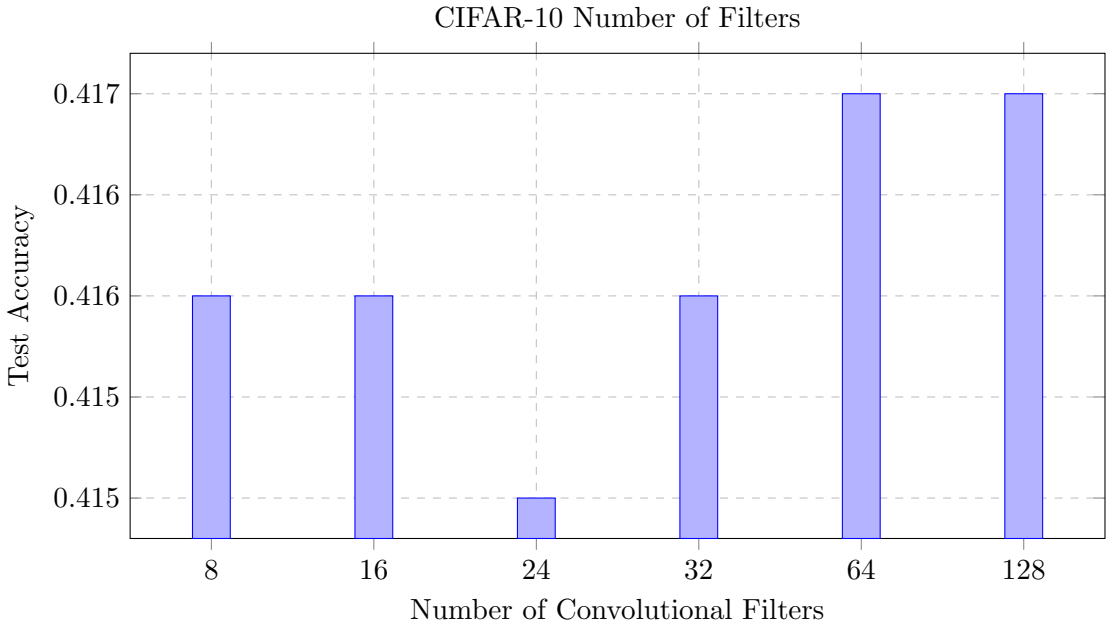


Figure 5.20: The hyperparameters used in this test were: 0.001 learning rate, 3 × 3 convolutional filters, and 8 training epochs

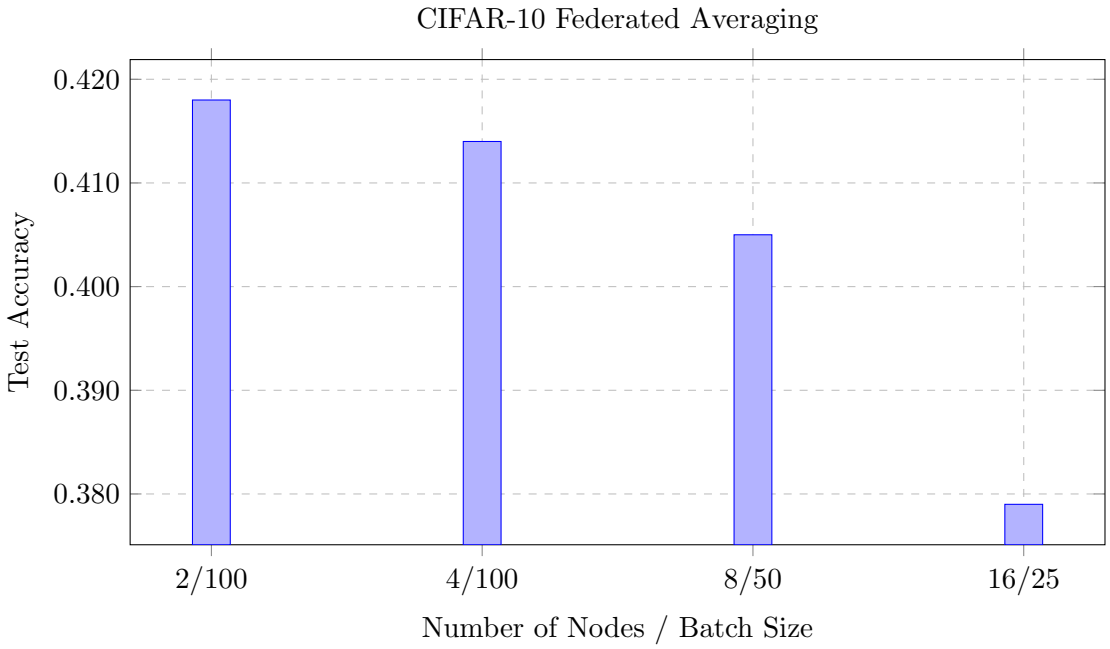


Figure 5.21: The hyperparameters used in this test were: 0.001 learning rate, eight 3 × 3 convolutional filters, and 8 training epochs

The federated averaging scheme described in section 4.4.2 was tested on the CIFAR-10 dataset similar to the MNIST dataset, with the same number of training nodes and batch sizes tested as seen in Figure 5.21. The discrepancy in validation accuracy for the various number of training nodes used was slightly higher than for the MNIST implementation; however, the decrease in accuracy follows the same trend.

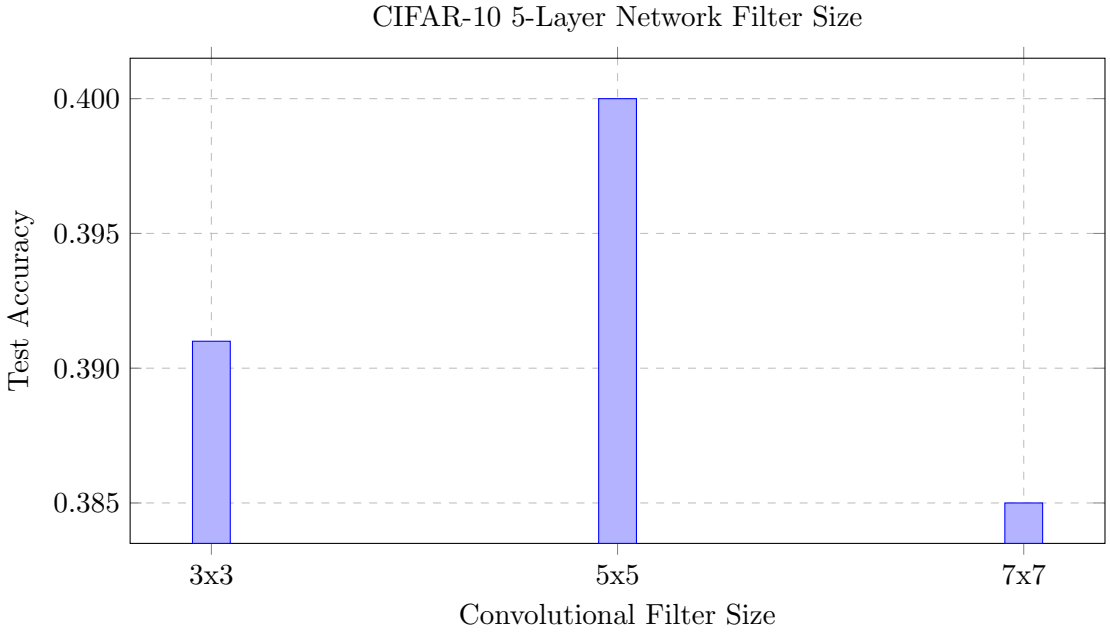


Figure 5.22: The hyperparameters used in this test were: 0.001 learning rate, eight convolutional filters, and 8 training epochs

As stated previously, the 5-Layer baseline architecture was originally developed with the goal of improving the validation accuracy achieved when training on the CIFAR-10 dataset. However, a decrease in validation accuracy of 1.1-2.6% was found when increasing the complexity of the baseline architecture. This decrease in accuracy held for all hyperparameters tested as evident in Figures 5.22 and 5.23.

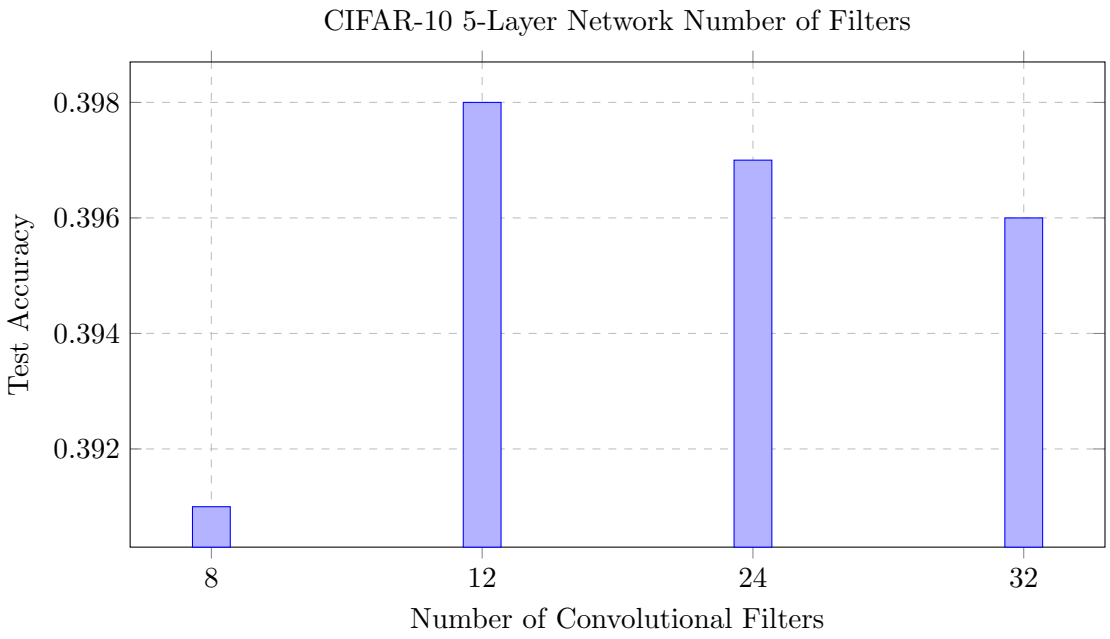


Figure 5.23: The hyperparameters used in this test were: 0.001 learning rate, 3 × 3 convolutional filters, and 8 training epochs

5.2.2 Secure Architecture

The secure 3-layer architecture was implemented using the hyperparameters seen in Table 5.2; although similar to the MNIST implementation, only 4 training epochs were used instead of 8. This modification was again made because certain test cases failed to train for more than 4 epochs due to inflating weights caused by performing numerous consecutive arithmetic calculations in the secure domain. A larger discrepancy between the validation accuracy achieved by the baseline and the secure architectures was observed while testing on CIFAR-10 than on MNIST, with further details given at the end of this section.

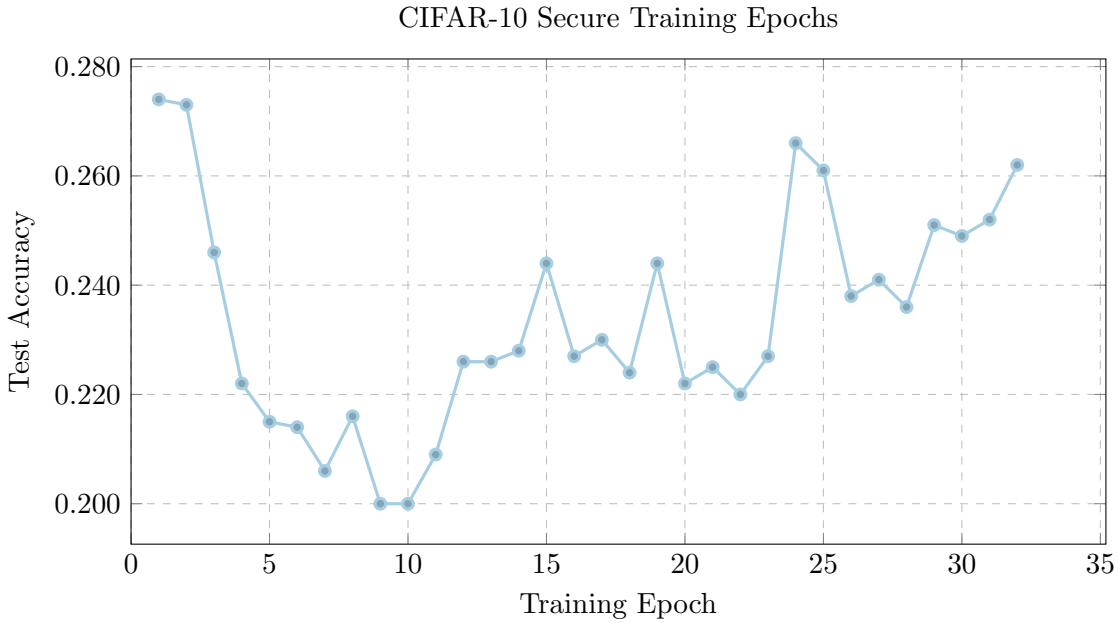


Figure 5.24: The hyperparameters used in this test were: 0.001 learning rate, and eight 3 × 3 convolutional filters

The impact of performing multiple training epochs on validation accuracy was recorded for the secure CIFAR-10 implementation using a learning rate of 0.001. This learning rate was selected as it proved to increase the network's ability to learn over multiple epochs better than the other learning rates tested on the baseline architecture. Figure 5.24 shows how this validation accuracy is affected over time; notably the results for the same learning rate are worse and much more sporadic than for the baseline tests shown in Figure 5.18.

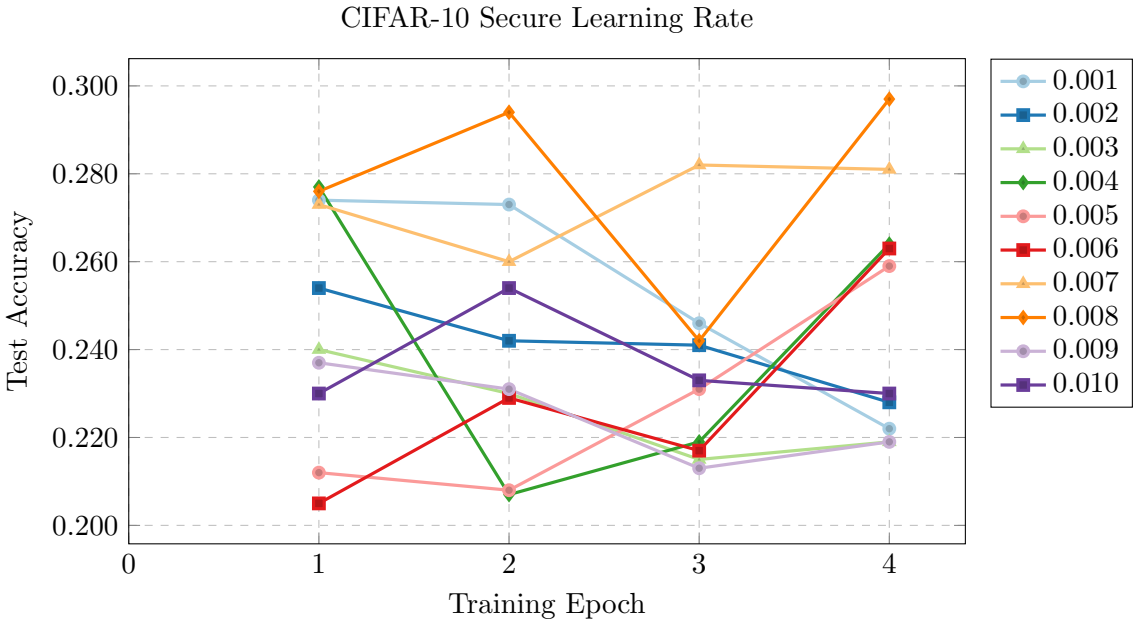


Figure 5.25: The hyperparameters used in this test were: eight 3×3 convolutional filters, and 4 training epochs

Learning rates in the range of 0.001-0.01 were tested to measure their impact on the ability of the secure architecture to learn over multiple epochs for the CIFAR-10 dataset. The results from these tests, shown in Figure 5.25, differ substantially from the learning rate tests performed on the baseline architecture. Most notably, higher learning rates proved more effective for training in the secure domain, whereas lower learning rates achieved better validation accuracy for the baseline architecture. The other notable difference between the two architectures is how the accuracy of different learning rates varied in a non-linear fashion. Although a learning rate of 0.001 performed quite poorly for the secure architecture, it was selected as the baseline parameter to facilitate more fair comparisons between the two implementations.

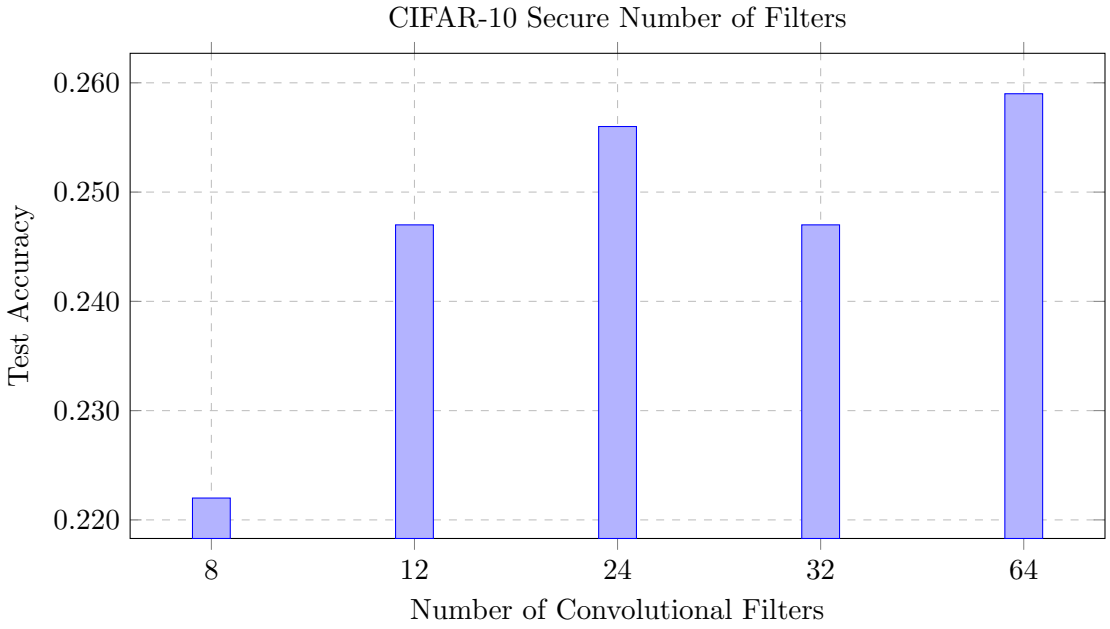


Figure 5.26: The hyperparameters used in this test were: 0.001 learning rate, 3 × 3 convolutional filters, and 4 training epochs

The impact of the number of convolutional filters on the validation accuracy of the secure architecture followed a trend similar to the baseline architecture; however, the accuracy differences between different filter counts varied to a greater extent. Similar to the MNIST implementation, varying the size of the convolutional filters led to a weight inflation failure during the first training epoch, so those results are not included here.

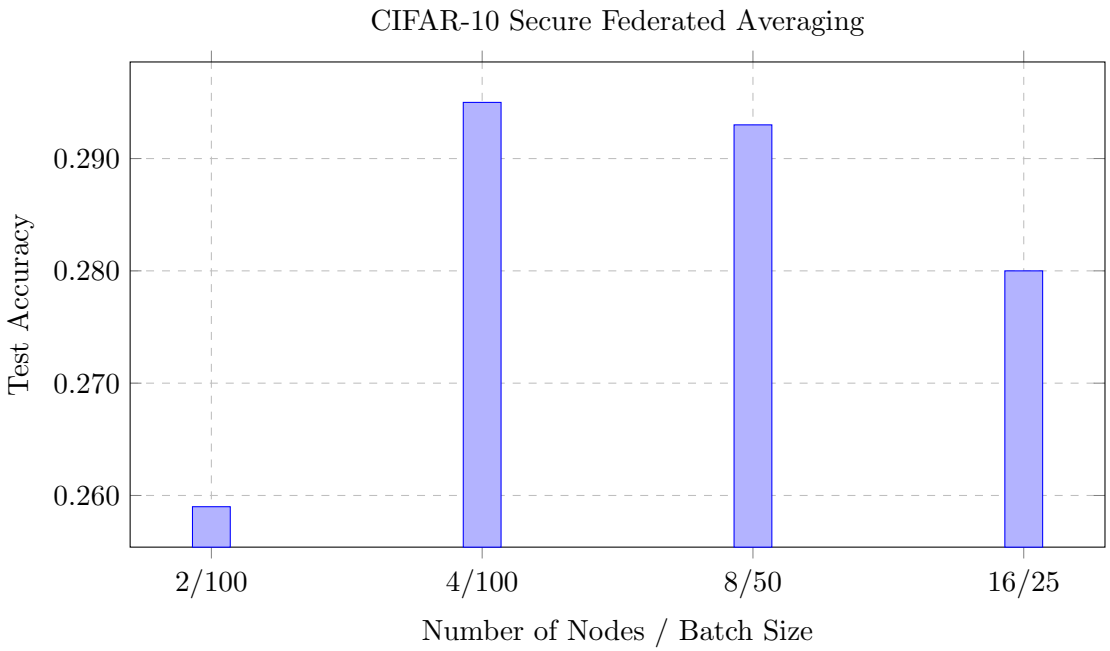


Figure 5.27: The hyperparameters used in this test were: 0.001 learning rate, eight 3×3 convolutional filters, and 4 training epochs

The same node and batch parameters were used to test the secure federated averaging scheme as the baseline federated averaging scheme. The most significant difference in the results for the secure implementation is the reduction in accuracy found using 2 training nodes and a batch size of 100, a configuration which resulted in the highest validation accuracy for the baseline federated averaging strategy. This decrease in accuracy may be due to how the training data is distributed to the nodes as the images in the training set of the CIFAR-10 dataset are randomly arranged. This could lead to a case where each of the two training nodes are receiving a non-uniformly distributed subset of the training samples, meaning certain classes are more heavily distributed to one node than the other.

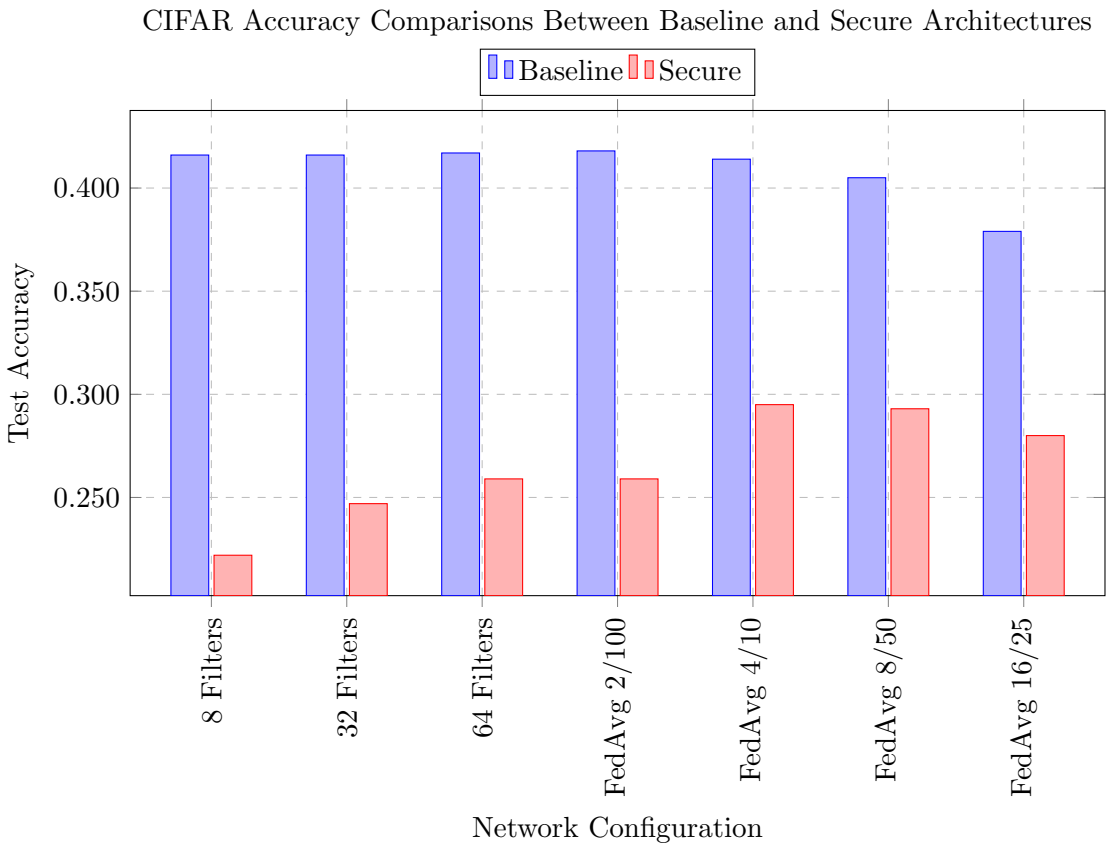


Figure 5.28: The hyperparameters used in this test were: 0.001 learning rate, eight 3×3 convolutional filters, and 8 training epochs

Figure 5.28 compares the validation accuracy achieved on the CIFAR-10 dataset between the baseline and secure architectures for different hyperparameter and network configurations. The large discrepancy in the accuracy of the baseline and secure architectures in the range of 9.9-19.4% compared to the range of 1.1-2.8% for MNIST results can be attributed to a couple of key factors. Perhaps the most important factor is the difference between the two datasets, with the images in CIFAR-10 contain three channels of data as opposed to only one. This causes the convolutional layer to perform three times as many arithmetic computations in the secure domain, which, as explored earlier, causes a decrease in the validation accuracy of a network. Another key factor for this large variation in accuracy is that the ideal learning rate for the baseline network performs poorly for the secure network. Possible solutions to decrease the difference in validation accuracy between the

baseline and secure architectures are explored in Section 6.2.

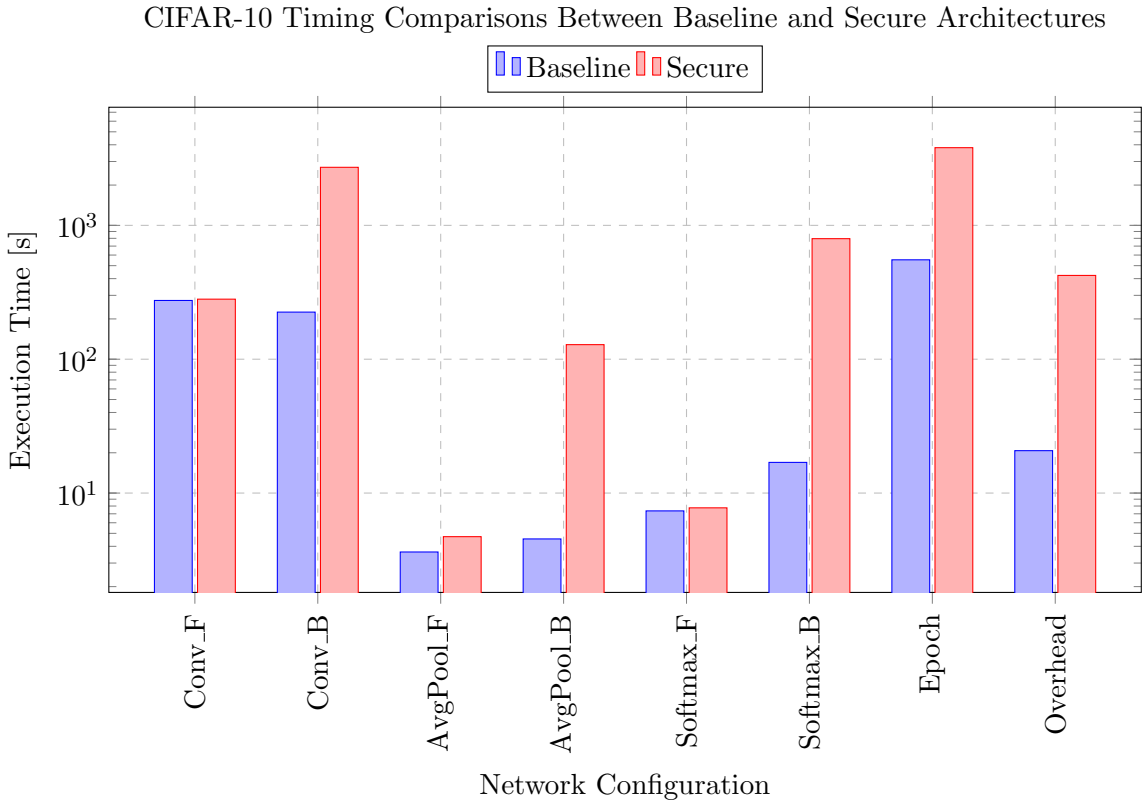


Figure 5.29: The hyperparameters used in this test were: 0.001 learning rate, eight 3×3 convolutional filters, and 8 training epochs

Figure 5.29 shows the differences in execution times for each layer function for the baseline and secure networks measured for a single training epoch. The backpropagation functions took between 12-47× as long to execute for the secure architecture as they did for the baseline architecture. These functions as well as the added overhead incurred from encrypting and decrypting 4-tuple values led to an overall increase in training time for a single epoch by a factor of 6.88× when using eight 3×3 convolutional filters. As with the MNIST results, these times do not include the additional communication overhead that is incurred through using the secure model. Simulations of this communication overhead are found in section 5.3.

5.3 Communication Simulation

The number of communication rounds needed by the security model to perform training on a single image is calculated as the total number of arithmetic calculations between two 4-tuple values, a result of the simplified version of the scheme proposed in [1] requiring only a single 16-byte communication between two parties in order to perform a given calculation. Although the more thorough 4-tuple scheme proposed by Aliasgari et. al requires more rounds of communication to occur to perform any arithmetic operation, it is much more robust at preventing data-leakage through these communications. Tables 5.3 and 5.4 detail the communication rounds required by the simplified scheme for training on a single image from the MNIST and CIFAR-10 datasets, respectively. These communication rounds do not take into account the communication required if the integer values that make up a secure 4-tuple are further encrypted using an MPC encryption scheme such as SPDZ. This further encryption would introduce additional overhead potentially magnitudes of order greater than described here.

Table 5.3: MNIST Communication Rounds per Training Image

	Convolution	Average Pooling	Softmax	Federated Averaging	Total
8 Filters	146232	10816	67660		224708
32 Filters	584928	43264	270460		898652
64 Filters	1169856	86528	540860		1797244
FedAvg 2/100	146232	10816	67660	81612	306320
FedAvg 4/100	146232	10816	67660	163224	387932
FedAvg 8/50	146232	10816	67660	326448	551156
FedAvg 16/25	146232	10816	67660	652896	877604

Assuming a 1Gb/s communication speed between each participating device in training, the communication overhead incurred from training on the MNIST dataset in the secure domain ranges from 0.027 - 0.214 seconds per image for the different network configurations tested, assuming ideal conditions. This would, therefore, require an additional 1,350

- 10,700 seconds of training time per epoch in addition to the 1,758 - 14,480 seconds required for the secure computations. This additional communication overhead consequently increases training times to almost double of that if they were run on a single secure device.

Using the same calculations on the CIFAR-10 dataset provides similar results, with communication overhead for a single image ranging from 0.082 - 0.656 seconds, dependant on network configuration. This leads to a total communication overhead ranging from 4,100 - 32,800 seconds for a single training epoch compared to the 3,805 - 41,415 seconds required for computations. These timing calculations again assume ideal communication conditions, and real-world results may prove that the added communication overhead required by the security model results in more time than performing all of the computations necessary for training.

Table 5.4: CIFAR-10 Communication Rounds per Training Image

	Convolution	Average Pooling	Softmax	Federated Averaging	Total
8 Filters	583848	14400	90060		688308
32 Filters	2335392	57600	360060		2753052
64 Filters	4670784	115200	720060		5506044
FedAvg 2/100	583848	14400	90060	109356	797664
FedAvg 4/100	583848	14400	90060	218712	907020
FedAvg 8/50	583848	14400	90060	437424	1125732
FedAvg 16/25	583848	14400	90060	874848	1563156

Chapter 6

Conclusions and Future Work

This chapter summarizes the contributions and accomplishments of this research as well as details the observations made while working towards the goal of creating a privacy-preserving image classification model. This chapter also includes several directions for future work for achieving more optimal results.

6.1 Conclusions

The field of machine learning based classifiers continues to grow, and as a result, many new domains are being introduced that require some level of privacy. While many techniques for implementing privacy-preserving classifiers exist, several have yet to address the problem of performing training in a secure domain without utilizing a plain-text model. The goal of this research was to create such a privacy-preserving architecture that utilizes secure data for all inputs and outputs to the classifier as well as every value used within the classifier during training, ultimately exploring the impact of integrating the necessary security models into a neural network-based classifier.

The results from this work demonstrate that an encrypted floating point scheme can be incorporated into the backpropagation phase of training such that its impact on validation accuracy remains relatively low. The largest complication that this proposed method

introduces is the increase in training times due to the increased computational complexity and communication overhead required to train in the secure domain. This additional training time was found to range from between 8 - 21× longer for the privacy-preserving network in comparison to a non-secure baseline architecture. This is within the range in which federated averaging can be utilized to achieve similar if not faster training times in the secure domain than on a single device at the cost of lower validation accuracy. Implementation of this research in a real-world setting would, thus, require some modifications to the implemented security model to reduce this computational and communication overhead in order to truly gain the benefits of training on non-secure devices as well as to address the possible data-leakage concerns generated by the simplified security model.

The contributions of this research can be summarized as follows:

1. A secure floating point scheme was developed that allows for basic arithmetic functions to be computed between two encrypted values without revealing the underlying data.

2. A secure CNN was developed using the proposed security model and was tested against a baseline network using both the MNIST and CIFAR-10 datasets.

3. Distributed learning was integrated into the secure architecture through the use of federated averaging and was employed to reduce the impact of the computational overhead caused by the security model by performing training in parallel across multiple devices.

4. The secure architecture achieved a validation accuracy 1.1-2.8% lower than that of the baseline architecture for the MNIST dataset and 9.9-19.4% lower for the CIFAR-10 dataset depending on the network configuration.

5. The computational and communication overhead associated with the proposed security model was found to outweigh the potential benefits from performing training on a distributed cloud-based computing platform instead of on a secure local device unless a distributed learning scheme was also utilized.

6.2 Future Work

The scope of this research limited the inclusion of multiple machine learning techniques that could improve the results found in this study. Thus, future work could extend this research through the suggestions listed below:

1. The security model developed in this research can be integrated with pre-existing machine learning frameworks to facilitate the creation of more complex CNNs with closer to state-of-the-art performance.

2. Optimization techniques such as GPU parallelization may be explored for reducing the training and testing time of the proposed secure architecture.

3. Increasing the validation of the secure architecture could be achieved by applying standard machine learning techniques such as preprocessing of the dataset before training and the inclusion of activation functions between intermediate layers in the network.

4. Integration of the security model into the forward propagation phase of training could decrease the overhead caused by encoding and decoding the the hidden values of the network for each training image. This would also allow for federated averaging to occur without the same encoding and decoding of all local weights.

5. Further encryption of the integer values that make up the secure floating point 4-tuples through the use of an MPC encryption scheme such as SPDZ could address the data leaking concerns present in the basic security model. Further SPDZ optimization techniques such as pre-computing of triples could also be applied to reduce the added communication overhead incurred with this second form of encryption.

6. More complex federated learning techniques could be applied to increase the validation accuracy associated with the distributed learning networks proposed in this research.

7. Modification of the arithmetic functions used by the security model could allow for larger convolutional filters to be used in the secure network, and could potentially lessen the decreased validation accuracy reported in this research caused by weight inflation and rounding errors. The weight inflation problem could also potentially be solved with the inclusion of activation functions placed between intermediate layers in the network.

Appendices

Appendix A Hyperparameter Effects on Computation Time

Timing results measured in seconds for different hyperparameters recorded for a single training epoch.

Table 1: MNIST Filter Size

	Conv_F	Conv_B	AvgPool_F	AvgPool_B	Softmax_F	Softmax_B
3×3	78.780	64.512	2.770	3.486	5.625	12.965
5×5	196.270	161.106	2.580	3.234	5.244	12.079
7×7	320.302	272.358	2.160	2.723	4.401	10.151

Table 2: MNIST Number of Filters

	Conv_F	Conv_B	AvgPool_F	AvgPool_B	Softmax_F	Softmax_B
8	78.780	64.512	2.770	3.486	5.625	12.965
16	158.592	129.370	5.500	6.934	11.250	25.906
24	234.274	193.593	8.196	10.371	16.828	38.885
32	313.625	257.910	10.965	13.908	22.631	52.164
48	474.320	387.251	16.454	20.861	34.230	78.180
64	643.398	524.008	25.095	30.121	47.399	107.379
128	1261.462	1030.691	45.153	55.583	90.766	207.897

Table 3: MNIST Federated Averaging

	Conv_F	Conv_B	AvgPool_F	AvgPool_B	Softmax_F	Softmax_B	FedAvg
2/100	41.075	33.591	1.454	1.820	3.091	6.830	0.211
4/100	20.472	16.781	0.723	0.909	1.545	3.413	0.377
8/50	10.249	8.391	0.362	0.456	0.776	1.708	1.425
16/25	5.204	4.245	0.183	0.228	0.393	0.860	5.585

Table 4: MNIST 5-Layer Filter Size

	Conv_F	Conv_B	AvgPool_F	AvgPool_B	Softmax_F	Softmax_B
3×3	269.044	375.704	4.547	6.913	1.191	2.971
5×5	496.516	643.023	3.755	5.699	0.756	1.915
7×7	616.425	759.366	2.923	4.411	0.214	0.510

Table 5: MNIST 5-Layer Number of Filters

	Conv_F	Conv_B	AvgPool_F	AvgPool_B	Softmax_F	Softmax_B
8	269.044	375.704	4.547	6.913	1.191	2.971
12	528.841	763.050	6.697	10.200	1.689	4.455
24	1712.762	2721.586	13.318	20.292	3.355	8.904
32	2900.599	4718.094	17.877	27.216	4.501	11.869

Table 6: MNIST Secure Number of Filters

	Conv_F	Conv_B	AvgPool_F	AvgPool_B	Softmax_F	Softmax_B	Overhead
8	79.843	702.978	2.815	88.424	5.731	597.283	280.918
16	157.231	1376.227	5.480	173.021	11.179	1172.283	541.437
32	327.468	2900.056	11.379	363.468	23.419	2441.245	1121.413
64	657.781	5877.508	22.832	738.453	47.179	4890.194	2245.269
128	1260.934	11209.354	44.282	1435.373	89.694	9386.402	4275.258

Table 7: MNIST Secure Federated Averaging

	Conv_F	Conv_B	AvgPool_F	AvgPool_B	Softmax_F	Softmax_B	FedAvg	Overhead
2/100	42.142	368.399	1.485	46.785	3.144	309.449	8.835	296.422
4/100	20.850	181.613	0.746	23.463	1.592	152.016	15.664	297.468
8/50	10.486	88.629	0.372	11.668	0.786	73.186	58.102	293.376
16/25	5.232	43.720	0.185	5.872	0.392	35.521	224.887	296.212

Table 8: CIFAR-10 Filter Size

	Conv_F	Conv_B	AvgPool_F	AvgPool_B	Softmax_F	Softmax_B
3×3	274.7475	224.8857	3.629186	4.542112	7.359882	16.95134
5×5	669.806	537.164	3.163	4.004	6.459	14.807
7×7	1186.617	953.281	2.949	3.626	5.986	13.541

Table 9: CIFAR-10 Number of Filters

	Conv_F	Conv_B	AvgPool_F	AvgPool_B	Softmax_F	Softmax_B
8	274.748	224.886	3.629	4.542	7.360	16.951
16	589.157	482.307	7.745	9.562	15.688	35.950
24	879.520	720.397	11.622	14.463	23.495	54.026
32	1191.369	976.695	15.688	19.362	34.703	72.313
64	2258.296	1846.959	31.890	37.745	61.590	139.918
128	4664.687	3816.148	67.628	80.908	126.881	290.140

Table 10: CIFAR-10 Federated Averaging

	Conv_F	Conv_B	AvgPool_F	AvgPool_B	Softmax_F	Softmax_B	FedAvg
2/100	140.218	115.037	2.354	2.771	4.543	8.555	0.258
4/100	70.204	57.374	1.180	1.398	1.929	4.289	0.460
8/50	35.108	28.745	0.589	0.696	0.965	2.144	1.731
16/25	17.515	14.396	0.294	0.347	0.483	1.073	6.659

Table 11: CIFAR-10 5-Layer Filter Size

	Conv_F	Conv_B	AvgPool_F	AvgPool_B	Softmax_F	Softmax_B
3×3	438.885	524.416	4.525	5.601	1.324	2.944
5×5	917.096	1035.290	3.686	4.612	0.865	1.993
7×7	1398.457	1480.893	3.139	3.858	0.404	0.803

Table 12: CIFAR-10 5-Layer Number of Filters

	Conv_F	Conv_B	AvgPool_F	AvgPool_B	Softmax_F	Softmax_B
8	438.885	524.416	4.525	5.601	1.324	2.944
12	741.470	1058.686	6.436	8.111	1.909	4.210
24	2147.803	3195.818	13.136	16.500	3.725	8.568
32	4001.004	6134.509	22.771	35.178	5.853	14.935

Table 13: CIFAR-10 Secure Number of Filters

	Conv_F	Conv_B	AvgPool_F	AvgPool_B	Softmax_F	Softmax_B	Overhead
8	280.979	2715.316	4.723	128.477	7.758	795.493	422.693
12	448.826	4293.894	7.361	204.867	12.279	1273.012	664.745
24	897.417	8579.298	14.750	422.178	24.554	2593.897	1312.983
32	1196.431	11971.068	19.647	554.824	35.184	3412.559	1749.679
64	2393.105	27417.859	39.439	1125.303	65.807	6882.540	3491.723

Table 14: CIFAR-10 Secure Federated Averaging

	Conv_F	Conv_B	AvgPool_F	AvgPool_B	Softmax_F	Softmax_B	FedAvg	Overhead
2/100	137.268	1153.477	1.812	56.968	3.683	339.021	10.744	367.172
4/100	69.185	573.969	0.907	28.463	1.844	166.672	18.832	367.141
8/50	34.561	284.747	0.453	14.225	0.925	82.067	70.886	367.377
16/25	17.242	141.554	0.227	7.114	0.464	40.557	276.464	364.305

Bibliography

[1] Mehrdad Aliasgari, Marina Blanton, Yihua Zhang, and Aaron Steele. Secure computation on floating point numbers. In *NDSS*, 2013.

[2] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 503–513, 1990.

[3] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

[4] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for federated learning on user-held data. *arXiv preprint arXiv:1611.04482*, 2016.

[5] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.

[6] Valerie Chen, Valerio Pastro, and Mariana Raykova. Secure computation for machine learning with spdz. *arXiv preprint arXiv:1901.00329*, 2019.

[7] Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, and Jurgen Schmidhuber. Convolutional neural network committees for handwritten character classification. In *2011 International Conference on Document Analysis and Recognition*, pages 1135–1139. IEEE, 2011.

[8] Dan Claudiu Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *Twenty-second international joint conference on artificial intelligence*, 2011.

[9] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P Smart. Practical covertly secure mpc for dishonest majority–or: breaking

the spdz limits. In *European Symposium on Research in Computer Security*, pages 1–18. Springer, 2013.

[10] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Annual Cryptology Conference*, pages 643–662. Springer, 2012.

[11] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[12] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, pages 201–210. PMLR, 2016.

[13] Oded Goldreich. Secure multi-party computation. *Manuscript. Preliminary version*, 78, 1998.

[14] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, 2018.

[15] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. Cryptodl: Deep neural networks over encrypted data. *arXiv preprint arXiv:1711.05189*, 2017.

[16] Andrew G Howard. Some improvements on deep convolutional neural network based image classification. *arXiv preprint arXiv:1312.5402*, 2013.

[17] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[18] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[19] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[21] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[22] Mariano Lemus, Mariana F. Ramos, Preeti Yadav, Nuno A. Silva, Nelson J. Muga, André Souto, Nikola Paunković, Paulo Mateus, and Armando N. Pinto. Generation and distribution of quantum oblivious keys for secure multiparty computation. *Applied Sciences*, 10(12), 2020.

[23] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020.

[24] Viktor M Lidkea, Radu Muresan, and Arafat Al-Dweik. Convolutional neural network framework for encrypted image classification in cloud-based its. *IEEE Open Journal of Intelligent Transportation Systems*, 1:35–50, 2020.

[25] Qian Lou and Lei Jiang. She: A fast and accurate deep neural network for encrypted data. *arXiv preprint arXiv:1906.00148*, 2019.

[26] Dengsheng Lu and Qihao Weng. A survey of image classification methods and techniques for improving classification performance. *International journal of Remote sensing*, 28(5):823–870, 2007.

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

[28] Vaikkunth Mugunthan, Antigoni Polychroniadou, David Byrd, and Tucker Hybinette Balch. Smpai: Secure multi-party computation for federated learning. In *33rd Conference on Neural Information Processing Systems*, 2019.

[29] Siddhartha Sankar Nath, Girish Mishra, Jajnyaseni Kar, Sayan Chakraborty, and Nilanjan Dey. A survey of image classification methods and techniques. In *2014 International conference on control, instrumentation, communication and computational technologies (ICCICCT)*, pages 554–557. IEEE, 2014.

[30] Seong Joon Oh, Max Augustin, Bernt Schiele, and Mario Fritz. Towards reverse-engineering black-box neural networks, 2018.

[31] Soo Beom Park, Jae Won Lee, and Sang Kyoon Kim. Content-based image classification using a neural network. *Pattern Recognition Letters*, 25(3):287–300, 2004.

[32] Ruben Seggers and Koen van der Veen. Privately training cnns using two-party spdz. 2018.

[33] Priyanshi Sharma. Everything about pooling layers and different types of pooling, 2018.

[34] Patrice Y Simard, David Steinkraus, John C Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *Icdar*, volume 3. Citeseer, 2003.

[35] Sameer Wagh, Divya Gupta, and Nishanth Chandran. Securenn: 3-party secure computation for neural network training. *Proceedings on Privacy Enhancing Technologies*, 2019(3):26–49, 2019.

[36] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H Yang, Farhad Farokhi, Shi Jin, Tony QS Quek, and H Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Transactions on Information Forensics and Security*, 15:3454–3469, 2020.

[37] Runhua Xu, Nathalie Baracaldo, Yi Zhou, Ali Anwar, and Heiko Ludwig. Hybridalpha: An efficient approach for privacy-preserving federated learning. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, pages 13–23, 2019.

[38] Runhua Xu, James BD Joshi, and Chao Li. Cryptonn: Training neural networks over encrypted data. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 1199–1209. IEEE, 2019.

[39] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

[40] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167. IEEE, 1986.