Handwritten Digit Classification using Convolutional Neural Networks with Federated Learning Integration

Shaba Altaf Shaon
Electrical and Computer Engineering
The University of Alabama in Huntsville
Huntsville, AL 35899
`ss0670@uah.edu`

Abstract

In machine learning, handwritten digit categorization is a basic task that is used as a standard to compare different algorithms' performance. Convolutional Neural Networks (CNNs) have shown impressive performance on datasets such as MNIST, making them an effective tool for image classification applications. However, due to the requirement to centralize data from several sources, standard centralized training methodologies might not be practical in situations where data privacy is an issue. In this work, we offer a unique method for handwritten digit classification that combines Federated Learning (FL) with CNNs. FL solves privacy and data ownership issues by allowing model training across dispersed devices while maintaining localized data. Our suggested approach makes use of federated learning techniques to train the model cooperatively across numerous edge devices and uses the power of CNNs to extract relevant features from handwritten digit images. Through extensive trials on the MNIST dataset, we show that this methodology is effective in achieving comparable performance when compared to typical centralized training approaches, all while maintaining data confidentiality and privacy. We also go over the efficiency and scalability of our suggested system, emphasizing its potential for practical uses in situations where data privacy is crucial. In summary, our research adds to the expanding corpus of knowledge on handwritten digit classification using CNNs as well as on FL and emphasizes the significance of privacy-preserving machine learning methods in the big data and distributed computing era.

Keywords: convolutional neural network, federated learning, handwritten digit classification

1 Introduction

Machine learning's foundational technique for classifying handwritten numbers has significant applications in a variety of fields, including optical character recognition and digital document processing. The availability of digitized data and the development of sophisticated algorithms have led to a revolutionary evolution of this fundamental problem in recent years. CNNs are one of the most effective tools available today, exhibiting exceptional ability to identify complex patterns and features that are present in unprocessed pixel data [1].

CNNs are the most popular models for classifying handwritten digits because of their inherent ability to automatically derive hierarchical representations from input images. CNNs have uncovered intricate correlations present in handwritten numbers by utilizing hierarchical feature learning, opening the door to previously unheard-of levels of accuracy and generalization. These neural networks detect little features, edges, and shapes in a manner similar to human visual perception. This allows them to identify minute elements that are essential for precise digit classification.

But despite all of this amazing advancement, data security and privacy remain a major worry. For model training, traditional centralized learning paradigms require aggregating sensitive data from several sources into a centralized server. This exposes individual data to potential security flaws and privacy violations. FL is a powerful paradigm that offers a paradigm change towards decentralized model training in this setting, all the while protecting the privacy of individual data providers.

Given these developments, this project proposal aims to combine federated learning with CNNs in order to create a strong handwritten digit classifier that has FL's privacy-preserving properties. Through the utilization of CNNs for feature extraction and classification, together with FL's decentralized training paradigm, the suggested framework aims to provide a balanced balance between privacy and performance. Our multidisciplinary project aims to push the boundaries of handwritten digit classification forward, promoting improved security and privacy in an increasingly digital environment.

2 Materials and Method

In this section, we outline the dataset, techniques, and procedures employed in our project to develop a robust handwritten digit classifier and later integrating federated learning (FL) with it. We describe the architecture of the Convolutional Neural Networks (CNNs) that are used, the datasets that are used for training and evaluation, and how federated learning is implemented to protect the privacy and security of data. We also present the experimental setup and assessment measures that we used to evaluate our suggested framework's performance. We hope to provide a thorough knowledge of the methodology used in this project by methodically presenting our materials and techniques.

2.1 MNIST Dataset

This work utilizes MNIST dataset for handwritten digit recognition. This dataset has been extensively used to validate novel techniques in computer vision, and in recent years, many authors have explored the performance of convolutional neural networks (CNNs) and other deep learning techniques over this dataset [2].

The MNIST dataset, short for Modified National Institute of Standards and Technology database, serves as a vast repository of handwritten digits, extensively utilized for training various image processing systems and machine learning models [7]. Originally, it was assembled by combining samples from NIST's original datasets. The creators felt that since NIST's training dataset was taken from American Census Bureau employees, while the testing dataset was taken from American high school students, it was not well-suited for machine learning experiments. To address this, MNIST's creators opted to remix the samples to ensure a more balanced representation suitable for machine learning experiments.

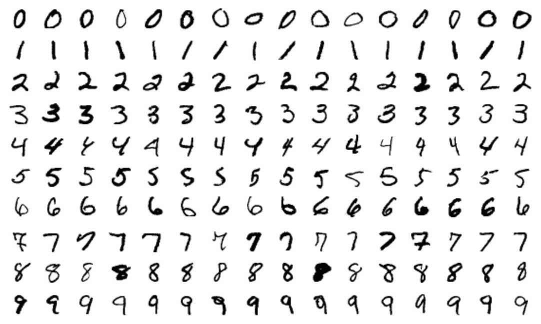


Figure 1: Sample image of MNIST test dataset taken from Wikipedia.

Moreover, the images in the MNIST database underwent standardization processes. Initially grayscale images from NIST were normalized to fit within a 28x28 pixel bounding box and subjected to anti-aliasing, resulting in images with varying grayscale levels. Comprising a total of 60,000 training images and 10,000 testing images, the MNIST dataset draws half of its samples from NIST's training dataset and the other half from its testing dataset. This balanced distribution ensures a fair representation of data across both training and testing phases.

2.2 Convolutional Neural Network

Convolutional Neural Networks (CNNs) represent a specialized form of artificial neural networks, meticulously designed to process and analyze visual data, such as images. CNNs have revolutionized various fields, including computer vision, image recognition, and pattern recognition, owing to their remarkable ability to automatically extract meaningful features from raw pixel data [3].

At the heart of CNNs lie convolutional layers, where convolution operations are applied to input images. These operations involve sliding small filters (also known as kernels) across the input image to detect patterns and features. Each filter detects specific features, such as edges, corners, or textures, through convolutions, which are then combined to form higher-level features. Through successive convolutional layers, CNNs can learn increasingly complex and abstract representations of the input data.

Additionally, CNNs incorporate pooling layers, which serve to downsample feature maps obtained from convolutional layers. Pooling operations, such as max pooling or average pooling, reduce the spatial dimensions of feature maps while retaining the most salient information. This helps in reducing computational complexity and mitigating overfitting by enforcing translation invariance.

Furthermore, CNN architectures often include fully connected layers, where neurons are densely connected to the neurons in the preceding layer. These layers enable the network to learn high-level representations by combining features extracted from earlier layers. Finally, CNNs are typically followed by activation functions, such as ReLU (Rectified Linear Unit), which introduce non-linearities into the network, enabling it to capture complex relationships in the data.

Training CNNs involves the iterative process of forward propagation, where input data is passed through the network to generate predictions, and backward propagation, where the network learns from errors and adjusts its parameters (weights and biases) to minimize the loss function. This process, often facilitated by optimization algorithms like gradient descent, enables CNNs to learn intricate patterns and features from vast amounts of labeled data [4].

In recent years, CNNs have emerged as a powerful tool for image analysis, offering unparalleled performance in tasks such as object detection, image classification, and segmentation. Their hierarchical architecture, coupled with efficient parameter learning, makes them indispensable in the realm of computer vision and beyond.

2.2.1 General Architecture of Convolutional Neural Network

A general CNN architecture typically consists of several layers designed to process and extract features from input data, such as images. Here's a high-level overview of a typical CNN architecture:

Input Layer: This layer receives the raw input data, which could be an image or a sequence of data, depending on the application.

Convolutional Layers: These layers are the core building blocks of CNNs. Each convolutional layer applies a set of learnable filters (also called kernels) to the input data. These filters extract features from the input by performing convolutions across the input spatial dimensions. Typically, multiple convolutional layers are stacked to learn hierarchical representations of the input data, with later layers capturing more abstract features.

Activation Function: After each convolutional operation, an activation function is applied element-wise to the output feature maps. Common choices include Rectified Linear Unit (ReLU), which introduces non-linearity into the network and helps capture complex patterns.

Pooling Layers: Pooling layers are used to downsample the spatial dimensions of the feature maps generated by the convolutional layers, reducing computational complexity and controlling overfitting. Max pooling and average pooling are the most commonly used pooling operations.

Fully Connected Layers: In the final layers of the CNN, fully connected (or dense) layers are used to perform classification or regression tasks. These layers connect every neuron from the previous layer to every neuron in the current layer, allowing the network to learn complex relationships in the data.

Flattening: Before passing the output of the last convolutional layer to the fully connected layers, the feature maps are typically flattened into a one-dimensional vector. This flattening operation converts the spatial information into a format suitable for the fully connected layers.

Output Layer: The output layer produces the final predictions or outputs of the CNN. The number of neurons in this layer depends on the specific task. For example, in image classification tasks, the output layer may have one neuron per class for binary classification or one neuron per class for multi-class classification.

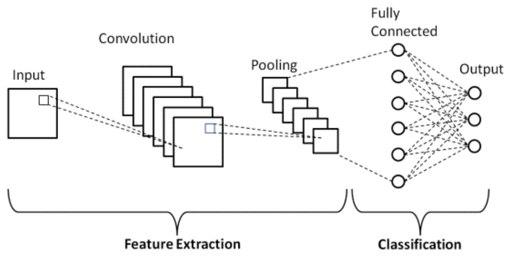


Figure 2: General architecture of a CNN model.

2.3 Data Preprocessing

Data preprocessing is a crucial step in preparing datasets for training machine learning models, including CNNs [8]. It involves transforming raw data into a format that is suitable for input into the model, improving model performance, and facilitating effective learning. Below are some common data preprocessing techniques used in the context of CNNs.

2.3.1 Normalization

Training can be stabilized and sped up by normalizing the pixel values of images to a common scale, usually between 0 and 1. By ensuring that each input characteristic has a comparable range, this keeps the learning process from being dominated by a single feature.

2.3.2 Resizing

Resizing images to a uniform size ensures consistency in input dimensions across the dataset. This is particularly important when working with image data, as CNN architectures typically require fixed-size inputs. Resizing also helps in reducing computational overhead during training.

2.3.3 Augmentation

Data augmentation techniques, such as rotation, flipping, scaling, and cropping, are used to artificially increase the diversity of the training dataset. This helps in improving the robustness and generalization ability of the CNN model by exposing it to variations in the input data.

2.3.4 Noise Reduction

In some cases, images may contain noise or artifacts that could adversely affect model performance. Applying noise reduction techniques, such as Gaussian blurring or median filtering, can help in smoothing out irregularities and enhancing the quality of the input data.

2.3.5 Feature Extraction

Preprocessing may involve extracting relevant features from the raw data to improve model performance. In the case of image data, this could include extracting edges, textures, or other informative features using techniques like edge detection or histogram equalization.

2.3.6 Label Encoding

For classification tasks, converting categorical labels into numerical representations (e.g., one-hot encoding) is essential for training CNN models. This ensures that the model can effectively learn and make predictions based on the provided labels.

2.3.7 Data Splitting

Before training the model, the dataset is typically split into separate training, validation, and test sets. This ensures that the model's performance can be evaluated on unseen data and helps in detecting overfitting during training.

Training Dataset: The training dataset forms the largest portion, comprising the samples used to train the Convolutional Neural Network (CNN). Here, the model learns from the input data and adjusts its parameters to minimize the training loss.

Validation Dataset: The validation dataset serves as a proxy for real-world performance assessment during training. It helps in fine-tuning hyperparameters and monitoring the model's progress, enabling us to make informed decisions regarding model architecture and training strategies.

Test Dataset: the test dataset remains untouched during training and is reserved exclusively for evaluating the model's performance on unseen data.

By applying appropriate preprocessing techniques, we can ensure that the data fed into CNN models is clean, standardized, and conducive to effective learning, ultimately leading to improved model performance and generalization.

In this work, we employ dropout regularization, batch normalization, and data augmentation as the key preprocessing techniques to enhance the effectiveness of our Convolutional Neural Network (CNN) model. As mentioned before, dropout regularization is a widely-used technique in CNNs to prevent overfitting by randomly deactivating neurons during training. It operates by temporarily removing a fraction of neurons, along with their connections, from the network architecture during each training iteration. By normalizing the data, we ensure that all input features have a similar scale, which can greatly improve the convergence speed of the training process and the overall performance of the model. Normalization also helps in mitigating the effects of varying illumination conditions and contrasts across different images in the dataset. This ensures that the CNN model focuses on learning relevant patterns and features in the data, rather than being influenced by irrelevant variations in pixel intensities. By incorporating data normalization as part of our preprocessing pipeline, we aim to create a more stable and robust training environment for our CNN model. This, in turn, is expected to result in improved model performance and generalization ability when applied to unseen data. Data augmentation is a technique commonly used in CNNs to artificially increase the size of the training dataset by applying various transformations to the existing images. These transformations include rotations, translations, scaling, flips, and changes in brightness or contrast.

Moreover, we also incorporate data splitting in this work. We divide the dataset into training, validation, and test datasets. By adhering to the meticulous data splitting strategy, we aim to ensure the reliability and robustness of our CNN model, facilitating meaningful insights and informed conclusions about its effectiveness in real-world applications.

2.4 Performance Metrics

Performance metrics evaluation is a critical step in assessing the effectiveness of a machine learning model. It provides quantitative measures of how well the model generalizes to unseen data and helps in understanding its strengths and weaknesses. Here are some key aspects of performance metrics evaluation:

2.4.1 Accuracy

Accuracy provides an overall measure of the model's correctness. However, it may not be sufficient for imbalanced datasets where one class dominates the others. Therefore, it's essential to consider other metrics as well. It is is calculated as the ratio of correctly predicted instances to the total instances.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

2.4.2 Precision

Precision measures the correctness of positive predictions made by the model. It is calculated as the ratio of true positive predictions to the total positive predictions made by the model.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

2.4.3 Recall

Recall measures the model's ability to correctly identify positive instances. It is calculated as the ratio of true positive predictions to the total actual positive instances in the data.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

2.4.4 F1-score

F1-score is the harmonic mean of precision and recall and provides a balanced measure of a model's performance. It considers both false positives and false negatives and is especially useful when dealing with imbalanced datasets.

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

2.5 Various Extensions to the Baseline CNN Model

Exploring and applying various extensions to the baseline model is crucial for enhancing performance, improving robustness, preventing overfitting, optimizing resources, and staying current with advances in machine learning [6]. It allows for iterative improvement, adaptation to changing requirements, and continuous refinement of the model's effectiveness in real-world applications. Among many other techniques used as extensions to the baseline model, dropout regularization is a frequently used method:

2.5.1 Dropout Regularization

In this work, we apply dropout regularization to the baseline CNN model to further improve the model's performance. Dropout regularization is a widely-used technique in neural network training, particularly in deep learning models. During training, dropout randomly deactivates a fraction of neurons in the network, effectively dropping them out of the forward and backward pass. This process introduces noise and prevents co-adaptation of neurons, forcing the network to learn more robust features.

Moreover-

- By randomly dropping out neurons, dropout regularization helps prevent the model from relying too heavily on specific neurons or features in the training data, thereby reducing the risk of overfitting.
- Dropout encourages the network to learn more generalizable features by forcing it to learn redundant representations. This results in a model that is better able to generalize to unseen data.
- Dropout acts as a form of regularization by adding noise to the network during training. This regularization helps control the model's complexity and prevents it from memorizing the training data.

2.5.2 Batch Normalization

In this work, we integrate batch normalization as a crucial component to improve the performance of our baseline model. Batch normalization is a technique that normalizes the input of each layer by subtracting the batch mean and dividing by the batch standard deviation. By doing so, batch normalization addresses the issue of internal covariate shift, which can hinder the training of deep neural networks. In the context of CNNs, batch normalization helps stabilize and accelerate the training process by ensuring that the inputs to each layer are consistently normalized, regardless of changes in the distribution of inputs due to variations in network parameters.

2.5.3 Data Augmentation

We also incorporate data augmentation in this work to expand our baseline model. Data augmentation involves applying a variety of transformations to the existing training data to create additional synthetic samples. These transformations include but are not limited to rotation, translation, scaling, flipping, and adding noise to the images. By introducing such variations to the training dataset, data augmentation serves to increase the diversity and richness of the training samples available to the model.

In the context of our handwritten digit classification task, data augmentation plays a vital role in mitigating overfitting and improving the generalization capability of our CNN model. By generating augmented samples with realistic variations in writing styles, orientations, and noise levels, we expose the model to a more comprehensive range of potential input variations that it might encounter during inference on real-world data.

2.6 Federated Learning

In our project, we implement Federated Learning (FL) to enable the distributed training of our handwritten digit classifier model across multiple devices or servers. FL allows us to train the model collaboratively while keeping the digitized data localized on individual devices [5]. Thus:

- FL addresses privacy concerns by ensuring that sensitive digitized data remains on local devices, reducing the risk of data exposure or breaches during model training.
- By leveraging distributed devices or servers for training, FL optimizes resource utilization and reduces the burden on any single device or server, leading to improved scalability and efficiency.
- FL enables the scalability of our handwritten digit classifier model across a network of devices, allowing for collaborative training without the need for central data aggregation.

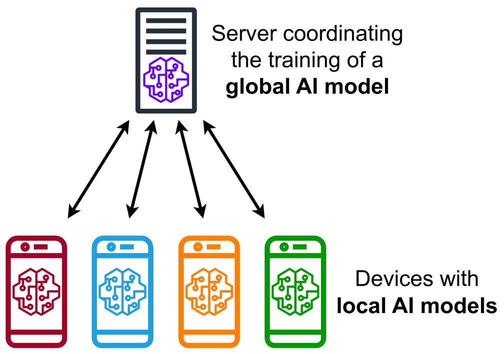


Figure 3: FL protocol taken from Wikipedia.

Federated Learning offers a secure and collaborative approach to training our handwritten digit classifier model, allowing for privacy-preserving and efficient utilization of distributed resources. By implementing FL, we aim to leverage the collective intelligence of distributed devices while ensuring the confidentiality and integrity of digitized data.

2.7 Methodology Flowchart

The methodology of this project is illustrated in the following flowchart:

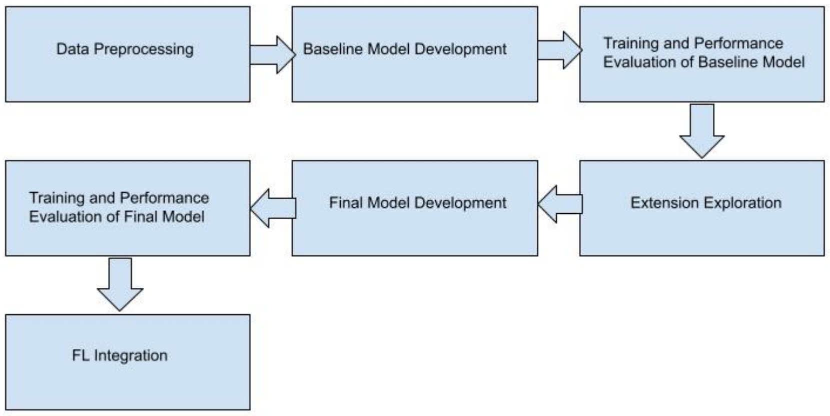

Data Preprocessing
Baseline Model Development
Training and Performance Evaluation of Baseline Model
Extension Exploration
Final Model Development
Training and Performance Evaluation of Final Model
FL Integration

Figure 4: Methodology flowchart of our project.

The steps are discussed below:

Step-1: Data Preprocessing
We normalize the input data to ensure consistent scaling across features and divide the dataset into training, validation, and test datasets for model evaluation.

Step-2: Baseline Model Development
We design and implement a simple CNN model as the baseline model.

Step-3: Training and Performance Evaluation of the Baseline Model
We train the baseline model on the training dataset and evaluate its performance on the validation set. Then we calculate baseline performance metrics such as accuracy, precision, recall, and F1-score on the validation set.

Step-4: Extension Exploration
We explore extension like dropout regularization to improve the baseline model's performance.

Step-5: Final Model Development
Based on findings from the baseline model and extension, we develop a finalized CNN model.

Step-6: Training and Performance Evaluation of the Baseline Model
We train the finalized model on the entire training dataset to maximize learning. Then we evaluate the performance of the finalized model on the test dataset and calculate performance metrics, such as accuracy, precision, recall, and F1-score, on the test set. We incorporate the finalized model to make predictions on new handwritten digit images.

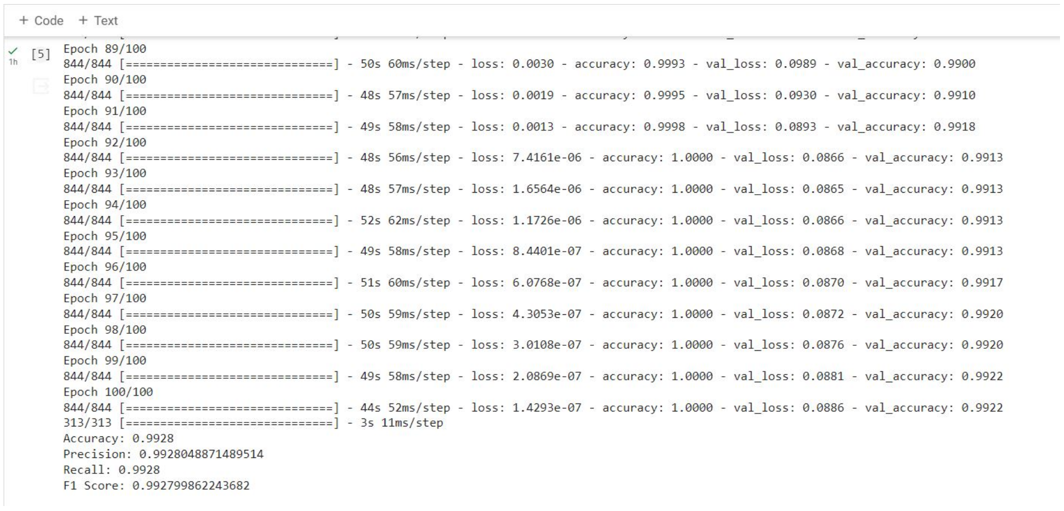
Step-7: FL Integration
We implement Federated Learning (FL) to enable distributed training of the CNN model across multiple devices or servers.

3 Simulation Result

The tool that has been used to do all the simulations in this project is Python. The GitHub repository for this project has been submitted. Here is the link for the GitHub repository:

`https://github.com/ShabaGit/CPE590_Project`

First, the script to develop, train, and evaluate a handwritten digit classifier using CNNs on the MNIST dataset has been run for 100 epochs. Here is a figure showing part of the result after running the simulation:



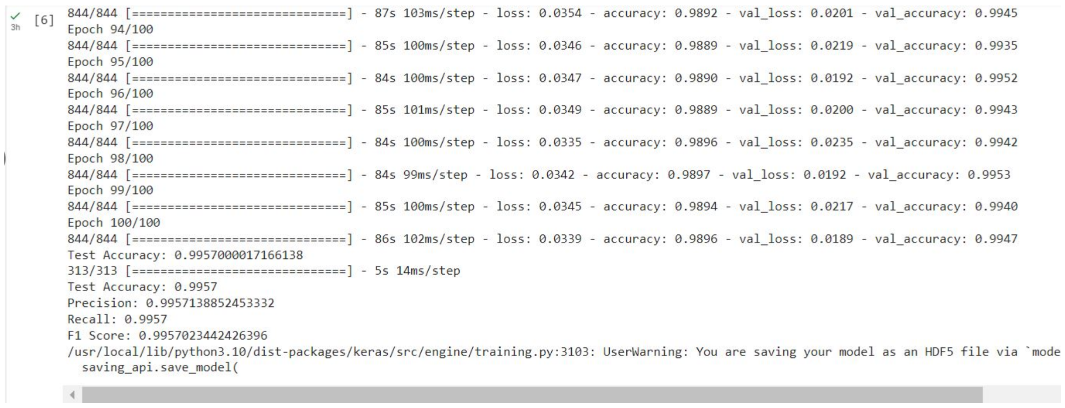
```
Epoch 89/100
844/844 [==============================] - 50s 60ms/step - loss: 0.0030 - accuracy: 0.9993 - val_loss: 0.0989 - val_accuracy: 0.9900
Epoch 90/100
844/844 [==============================] - 48s 57ms/step - loss: 0.0019 - accuracy: 0.9995 - val_loss: 0.0930 - val_accuracy: 0.9910
Epoch 91/100
844/844 [==============================] - 49s 58ms/step - loss: 0.0013 - accuracy: 0.9998 - val_loss: 0.0893 - val_accuracy: 0.9918
Epoch 92/100
844/844 [==============================] - 48s 56ms/step - loss: 7.4161e-06 - accuracy: 1.0000 - val_loss: 0.0866 - val_accuracy: 0.9913
Epoch 93/100
844/844 [==============================] - 48s 57ms/step - loss: 1.6564e-06 - accuracy: 1.0000 - val_loss: 0.0865 - val_accuracy: 0.9913
Epoch 94/100
844/844 [==============================] - 52s 62ms/step - loss: 1.1726e-06 - accuracy: 1.0000 - val_loss: 0.0866 - val_accuracy: 0.9913
Epoch 95/100
844/844 [==============================] - 49s 58ms/step - loss: 8.4401e-07 - accuracy: 1.0000 - val_loss: 0.0868 - val_accuracy: 0.9913
Epoch 96/100
844/844 [==============================] - 51s 60ms/step - loss: 6.0768e-07 - accuracy: 1.0000 - val_loss: 0.0870 - val_accuracy: 0.9917
Epoch 97/100
844/844 [==============================] - 50s 59ms/step - loss: 4.3053e-07 - accuracy: 1.0000 - val_loss: 0.0872 - val_accuracy: 0.9920
Epoch 98/100
844/844 [==============================] - 50s 59ms/step - loss: 3.0108e-07 - accuracy: 1.0000 - val_loss: 0.0876 - val_accuracy: 0.9920
Epoch 99/100
844/844 [==============================] - 49s 58ms/step - loss: 2.0869e-07 - accuracy: 1.0000 - val_loss: 0.0881 - val_accuracy: 0.9922
Epoch 100/100
844/844 [==============================] - 44s 52ms/step - loss: 1.4293e-07 - accuracy: 1.0000 - val_loss: 0.0886 - val_accuracy: 0.9922
313/313 [==============================] - 3s 11ms/step
Accuracy: 0.9928
Precision: 0.9928048871489514
Recall: 0.9928
F1 Score: 0.992799862243682
```

Figure 5: Simulation result snippet of baseline model run.

From the simulation snippet, it can be seen that the results are as follows:
Accuracy: 0.9928
Precision: 0.9928048871489514
Recall: 0.9928
F1 Score: 0.992799862243682

Here, these results suggest that the model has been trained effectively and is performing well on the MNIST dataset.

Then, another code is run to extend the baseline model by incorporating dropout, batch normalization, and data augmentation techniques. It then trains the finalized model on the entire training dataset and evaluates its performance on the test dataset. Then it calculated the performance metrics. This code is run for 100 epochs too. Below is a figure showing part of the result after running the simulation:



```
844/844 [==============================] - 87s 103ms/step - loss: 0.0354 - accuracy: 0.9892 - val_loss: 0.0201 - val_accuracy: 0.9945
Epoch 94/100
844/844 [==============================] - 85s 100ms/step - loss: 0.0346 - accuracy: 0.9889 - val_loss: 0.0219 - val_accuracy: 0.9935
Epoch 95/100
844/844 [==============================] - 84s 100ms/step - loss: 0.0347 - accuracy: 0.9890 - val_loss: 0.0192 - val_accuracy: 0.9952
Epoch 96/100
844/844 [==============================] - 85s 101ms/step - loss: 0.0349 - accuracy: 0.9889 - val_loss: 0.0200 - val_accuracy: 0.9943
Epoch 97/100
844/844 [==============================] - 84s 100ms/step - loss: 0.0335 - accuracy: 0.9896 - val_loss: 0.0235 - val_accuracy: 0.9942
Epoch 98/100
844/844 [==============================] - 84s 99ms/step - loss: 0.0342 - accuracy: 0.9897 - val_loss: 0.0192 - val_accuracy: 0.9953
Epoch 99/100
844/844 [==============================] - 85s 100ms/step - loss: 0.0345 - accuracy: 0.9894 - val_loss: 0.0217 - val_accuracy: 0.9940
Epoch 100/100
844/844 [==============================] - 86s 102ms/step - loss: 0.0339 - accuracy: 0.9896 - val_loss: 0.0189 - val_accuracy: 0.9947
Test Accuracy: 0.9957000017166138
313/313 [==============================] - 5s 14ms/step
Test Accuracy: 0.9957
Precision: 0.9957138852453332
Recall: 0.9957
F1 Score: 0.9957023442426396
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `mode
  saving_api.save_model(
```

Figure 6: Simulation result snippet of finalized model run.

Also, from the simulation snippet, it can be seen that the finalized model yields better result.

The results for performance metrics of the finalized model are given below:
Test Accuracy: 0.9957
Precision: 0.9957138852453332
Recall: 0.9957
F1 Score: 0.9957023442426396

Accuracy measures the overall correctness of the model's predictions. An accuracy of 99.57% indicates that the model is making correct predictions for the vast majority of the dataset. Precision measures the ratio of correctly predicted positive observations to the total predicted positives. A precision of 99.57138852453332% suggests that when the model predicts a digit as a certain class, it's very likely to be correct. Recall measures the ratio of correctly predicted positive observations to all the observations in the actual class. A recall of 99.57% indicates that the model is capturing nearly all instances of each digit class in the dataset. The F1 score is the harmonic mean of precision and recall. It provides a single score that balances both precision and recall. With an F1 score of 99.57023442426396%, the model is achieving high performance across both precision and recall. Taken together, these outcomes indicate that the model has undergone effective training and demonstrates outstanding performance when tasked with classifying the MNIST dataset.

From the results mentioned above, the final accuracy achieved by the baseline model is 99.28% while the final accuracy achieved by the finalized model is 99.57%. If baseline and finalized model are compared with each other, it can be said that the finalized model outperforms the baseline model achieving higher accuracy level.

However, these metrics are based on the dataset used for training and evaluation. It's important to test the model on unseen data to ensure its generalization ability. Hence, we next run simulation to incorporate the finalized model to make predictions on new handwritten digit images.

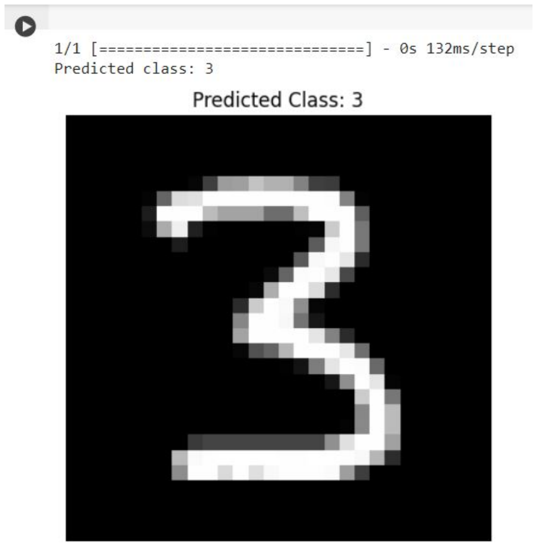

```
1/1 [==============================] - 0s 132ms/step
Predicted class: 3
```

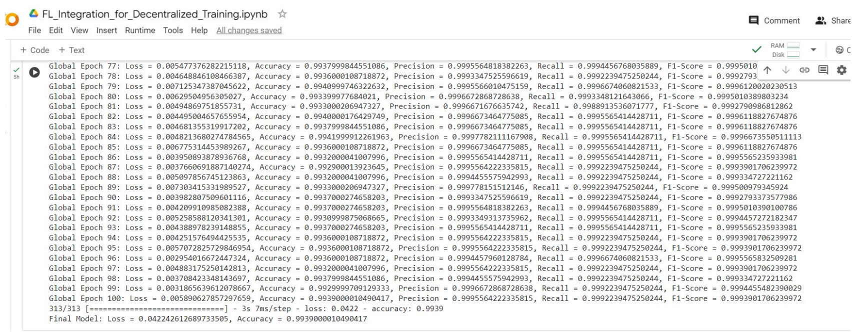
Figure 7: Finalized model predicting on new handwritten digit image.

Finally, we do the simulation to implement FL to enable distributed training of the CNN model across multiple devices or servers for 100 global iterations.

From Figure 8, it can be seen that approximately 99.39% accuracy has been achieved here, which indicates a good accuracy level.

On the next page, there are some figures from the simulation:

After running simulations to implement Federated Learning (FL) for distributed training of the CNN model across multiple devices or servers, the accuracy graph in Figure 9 showcases the collaborative learning process. Initially, individual devices or servers contribute their local model updates, resulting in slight fluctuations in accuracy. However, as the federated averaging algorithm aggregates these updates and refines the global model, the accuracy steadily improves over iterations. The graph


FL_Integration_for_Decentralized_Training.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
5h

```
Global Epoch 77: Loss = 0.005477376282215118, Accuracy = 0.9937999844551086, Precision = 0.9995564818382263, Recall = 0.9994456768035889, F1-Score = 0.9995010
Global Epoch 78: Loss = 0.004648846108466387, Accuracy = 0.9936000108718872, Precision = 0.9993347525596619, Recall = 0.9992239475250244, F1-Score = 0.9992793
Global Epoch 79: Loss = 0.007125347387045622, Accuracy = 0.9940999746322632, Precision = 0.9995566010475159, Recall = 0.9996674060821533, F1-Score = 0.9996120020230513
Global Epoch 80: Loss = 0.00629504956305027, Accuracy = 0.993399977684021, Precision = 0.9996672868728638, Recall = 0.9993348121643066, F1-Score = 0.9995010389803234
Global Epoch 81: Loss = 0.00494869751855731, Accuracy = 0.9933000206947327, Precision = 0.9996671676635742, Recall = 0.9988913536071777, F1-Score = 0.9992790986812862
Global Epoch 82: Loss = 0.004495004657655954, Accuracy = 0.9940000176429749, Precision = 0.9996673464775085, Recall = 0.9995565414428711, F1-Score = 0.9996118827674876
Global Epoch 83: Loss = 0.004681355319917202, Accuracy = 0.9937999844551086, Precision = 0.9996673464775085, Recall = 0.9995565414428711, F1-Score = 0.9996118827674876
Global Epoch 84: Loss = 0.0048213680274784565, Accuracy = 0.9941999912261963, Precision = 0.9997782111167908, Recall = 0.9995565414428711, F1-Score = 0.9996673550511113
Global Epoch 85: Loss = 0.006775314453989267, Accuracy = 0.9936000108718872, Precision = 0.9996673464775085, Recall = 0.9995565414428711, F1-Score = 0.9996118827674876
Global Epoch 86: Loss = 0.003950893878936768, Accuracy = 0.9932000041007996, Precision = 0.9995565414428711, Recall = 0.9995565414428711, F1-Score = 0.9995565235933981
Global Epoch 87: Loss = 0.0037660691887140274, Accuracy = 0.992900013923645, Precision = 0.9995564222335815, Recall = 0.9992239475250244, F1-Score = 0.9993901706239972
Global Epoch 88: Loss = 0.005097856745123863, Accuracy = 0.9932000041007996, Precision = 0.9994455575942993, Recall = 0.9992239475250244, F1-Score = 0.99933472722116２
Global Epoch 89: Loss = 0.007303415331989527, Accuracy = 0.9933000206947327, Precision = 0.999778151512146, Recall = 0.9992239475250244, F1-Score = 0.999500979345924
Global Epoch 90: Loss = 0.003982807509601116, Accuracy = 0.9937000274658203, Precision = 0.9993347525596619, Recall = 0.9992239475250244, F1-Score = 0.9992793373577986
Global Epoch 91: Loss = 0.004209910985082388, Accuracy = 0.9937000274658203, Precision = 0.9995564818382263, Recall = 0.9994456768035889, F1-Score = 0.9995010390100786
Global Epoch 92: Loss = 0.005258588120341301, Accuracy = 0.9930999875068665, Precision = 0.9993349313735962, Recall = 0.9995565414428711, F1-Score = 0.9994457272182347
Global Epoch 93: Loss = 0.004388978239148855, Accuracy = 0.9937000274658203, Precision = 0.9995565414428711, Recall = 0.9995565414428711, F1-Score = 0.9995565235933981
Global Epoch 94: Loss = 0.004251576494425535, Accuracy = 0.9936000108718872, Precision = 0.9995564222335815, Recall = 0.9992239475250244, F1-Score = 0.9993901706239972
Global Epoch 95: Loss = 0.0057072825729846954, Accuracy = 0.9936000108718872, Precision = 0.9995564222335815, Recall = 0.9992239475250244, F1-Score = 0.9993901706239972
Global Epoch 96: Loss = 0.002954016672447324, Accuracy = 0.9936000108718872, Precision = 0.9994457960128784, Recall = 0.9996674060821533, F1-Score = 0.9995565832509281
Global Epoch 97: Loss = 0.004883175250142813, Accuracy = 0.9932000041007996, Precision = 0.9995564222335815, Recall = 0.9992239475250244, F1-Score = 0.9993901706239972
Global Epoch 98: Loss = 0.003708423348143697, Accuracy = 0.9937999844551086, Precision = 0.9994455575942993, Recall = 0.9992239475250244, F1-Score = 0.999334727221162
Global Epoch 99: Loss = 0.0031865639612078667, Accuracy = 0.9929999709129333, Precision = 0.9996672868728638, Recall = 0.9992239475250244, F1-Score = 0.9994455482390029
Global Epoch 100: Loss = 0.005890627857297659, Accuracy = 0.9939000010490417, Precision = 0.9995564222335815, Recall = 0.9992239475250244, F1-Score = 0.9993901706239972
313/313 [==============================] - 3s 7ms/step - loss: 0.0422 - accuracy: 0.9939
Final Model: Loss = 0.04224261268973350５, Accuracy = 0.9939000010490417
```

Figure 8: FL integration for distributed model training.

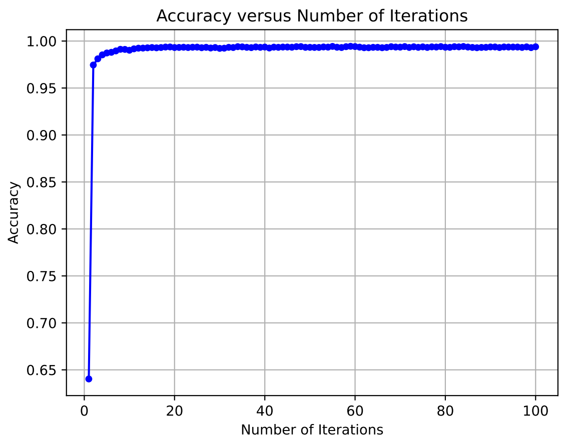


Figure 9: Accuracy versus number of iterations after running FL.

illustrates the effectiveness of FL in harnessing collective intelligence while preserving data privacy, enabling the CNN model to achieve impressive performance across the distributed network.

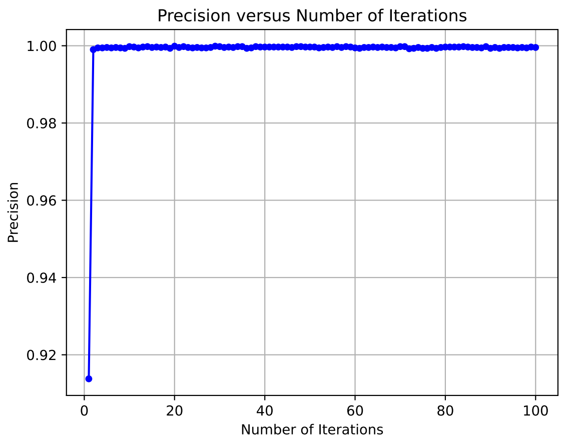


Figure 10: Precision versus number of iterations after running FL.

The precision graph in Figure 10 vividly illustrates the collaborative learning dynamics. As the federated averaging algorithm aggregates model updates and fosters collaboration, precision tends to stabilize and improve over iterations.

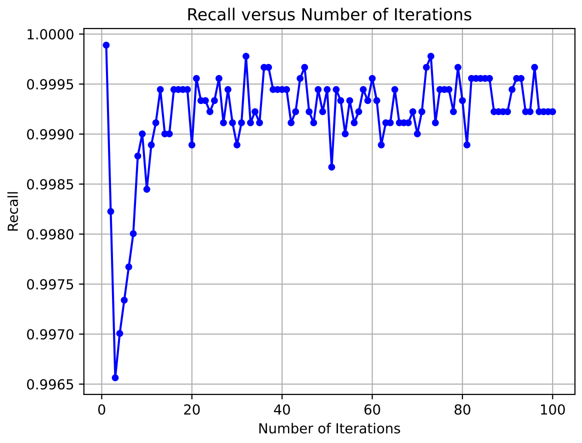


Figure 11: Recall versus number of iterations after running FL.

The recall graph in Figure 11 shows fluctuations in recall values for the first few iterations. However, as the number of iterations increases, these values also tend to stabilize and improve.

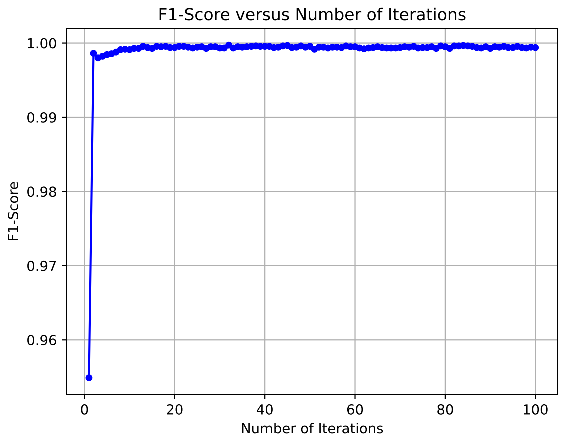


Figure 12: F1-Score versus number of iterations after running FL.

Similar to the precision graph, the F1-score graph in Figure 12 also demonstrates that as the federated averaging algorithm aggregates model updates, f1-score values also stabilize and improve over iterations.

4 Discussion and Conclusions

This project outlines a systematic methodology for developing a high-performing handwritten digit classifier using CNNs and integrating FL for distributed training. By meticulously designing and evaluating CNN architectures, implementing data preprocessing techniques, we create a robust model capable of accurately recognizing handwritten digits. We incorporate FL to enable distributed training of CNN model while safeguarding user data and security. Through this approach, we aspire to contribute to the advancement of machine learning techniques for real-world applications, balancing model effectiveness with ethical considerations surrounding data privacy and security. Finally, the effectiveness of this project has been demonstrated by simulation results.

References

[1] Shao, Haijian Ma, Edwin Zhu, Ming Deng, Xing Zhai, Shengjie. (2023). MNIST Handwritten Digit Classification Based on Convolutional Neural Network with Hyperparameter Optimization. Intelligent Automation Soft Computing. 36. 3595. 10.32604/iasc.2023.036323.

[2] Baldominos, A.; Saez, Y.; Isasi, P. A Survey of Handwritten Character Recognition with MNIST and EMNIST. Appl. Sci. 2019, 9, 3169. https://doi.org/10.3390/app9153169.

[3] Alzubaidi, L., Zhang, J., Humaidi, A.J. et al. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. J Big Data 8, 53 (2021). https://doi.org/10.1186/s40537-021-00444-8.

[4] Hasselmo, M.E., Schnell, E. & Barkai, E. (1995) Dynamics of learning and recall at excitatory recurrent synapses and cholinergic modulation in rat hippocampal region CA3. *Journal of Neuroscience* **15**(7):5249-5262.

[5] H. N. C. Neto, J. Hribar, I. Dusparic, D. M. F. Mattos and N. C. Fernandes, "A Survey on Securing Federated Learning: Analysis of Applications, Attacks, Challenges, and Trends," in IEEE Access, vol. 11, pp. 41928-41953, 2023, doi: 10.1109/ACCESS.2023.3269980.

[6] D. Beohar and A. Rasool, "Handwritten Digit Recognition of MNIST dataset using Deep Learning state-of-the-art Artificial Neural Network (ANN) and Convolutional Neural Network (CNN)," 2021 International Conference on Emerging Smart Computing and Informatics (ESCI), Pune, India, 2021, pp. 542-548, doi: 10.1109/ESCI50559.2021.9396870.

[7] L. Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]," in IEEE Signal Processing Magazine, vol. 29, no. 6, pp. 141-142, Nov. 2012, doi: 10.1109/MSP.2012.2211477.

[8] Kun Zhou, Sung-Kwun Oh, Witold Pedrycz, Jianlong Qiu, Data preprocessing strategy in constructing convolutional neural network classifier based on constrained particle swarm optimization with fuzzy penalty function, Engineering Applications of Artificial Intelligence, Volume 117, Part B, 2023, 105580, ISSN 0952-1976, https://doi.org/10.1016/j.engappai.2022.105580.