

**Infosys Springboard Internship 5.0**

(October 2024 – December 2024)

Documentation Report on

Handwritten Digit Recognition

Using LeNet-5 Model in PyTorch

By

Jyothinagaram Meghana

Artificial Intelligence Intern

Batch 1

[meghasetty1@gmail.com](mailto:meghasetty1@gmail.com)

Submitted to: Ms. Pranaya Mam

**Table of Contents**

Introduction

Data Collection

Dataset Overview

Data Preprocessing

Data Visualization and Understanding

Modeling Approaches

MLP

CNN

LeNet-5

Hyperparameters Used

Results and Evaluation

Conclusion

References

**Introduction**

*Problem Statement*

Handwritten digit recognition is one of the fundamental challenges in the field of computer vision. The goal of this project is to develop a model capable of recognizing handwritten digits from an image dataset, accurately classifying them into one of the 10 categories (digits 0-9). This problem has vast applications, such as in postal services for reading handwritten zip codes, document processing, and automation systems.

Despite the simplicity of the task, recognizing handwritten digits remains a challenging problem due to variations in writing styles, different fonts, and noisy image backgrounds. The main challenge lies in training a model that can generalize well to different handwriting styles while maintaining high accuracy.

*Project Objective*

The objective of this project is to build and train a neural network model capable of achieving high accuracy in recognizing handwritten digits from the MNIST dataset. By employing various machine learning models, including Multi-layer Perceptron (MLP), Convolutional Neural Network (CNN), and LeNet-5, the project aims to compare their performances and select the best-performing model based on metrics like accuracy, precision, recall, and loss.

*Dataset Description*

The dataset used in this project is the MNIST (Modified National Institute of Standards and Technology) dataset. It consists of 70,000 28x28 grayscale images of handwritten digits (0-9), which are split into two parts: a training set of 60,000 images and a test set of 10,000 images. The dataset is labeled, meaning each image is associated with its corresponding digit label. The MNIST dataset has become a standard benchmark in machine learning and is widely used for training image classification models.

*Methodology*

The methodology employed in this project involves the following key steps:

Data Collection: Downloading the MNIST dataset from a public source.

Data Preprocessing: Transforming the dataset to be suitable for input into the machine learning models, including normalization, reshaping, and data augmentation.

Model Selection: Implementing and training different models like MLP, CNN, and LeNet-5.

Model Training: Training each model on the preprocessed training data and evaluating their performance on the test set.

Model Evaluation: Comparing the models based on accuracy, loss, and other performance metrics to determine the best-performing model.

Hyperparameter Tuning: Fine-tuning the models' hyperparameters to improve their performance further.

*Tools and Technologies*

The following tools and technologies were used in this project:

Programming Language: Python

Deep Learning Framework: PyTorch version 2.2.2 (used for model implementation, training, and evaluation)

Dataset: MNIST dataset, available through PyTorch's torchvision library

Visualization: Matplotlib for data visualization

IDE: Jupyter Notebooks for model development and experimentation

*Structure of the Document*

The document is organized into the following sections:

Introduction: Provides an overview of the problem statement, project objectives, dataset description, methodology, tools, and technologies.

Data Collection: Describes how the dataset was collected and processed for the project.

Data Preprocessing: Details the preprocessing steps, including normalization and transformations applied to the data.

Dataset Overview: Provides a deeper dive into the dataset itself, its characteristics, and the structure of the data.

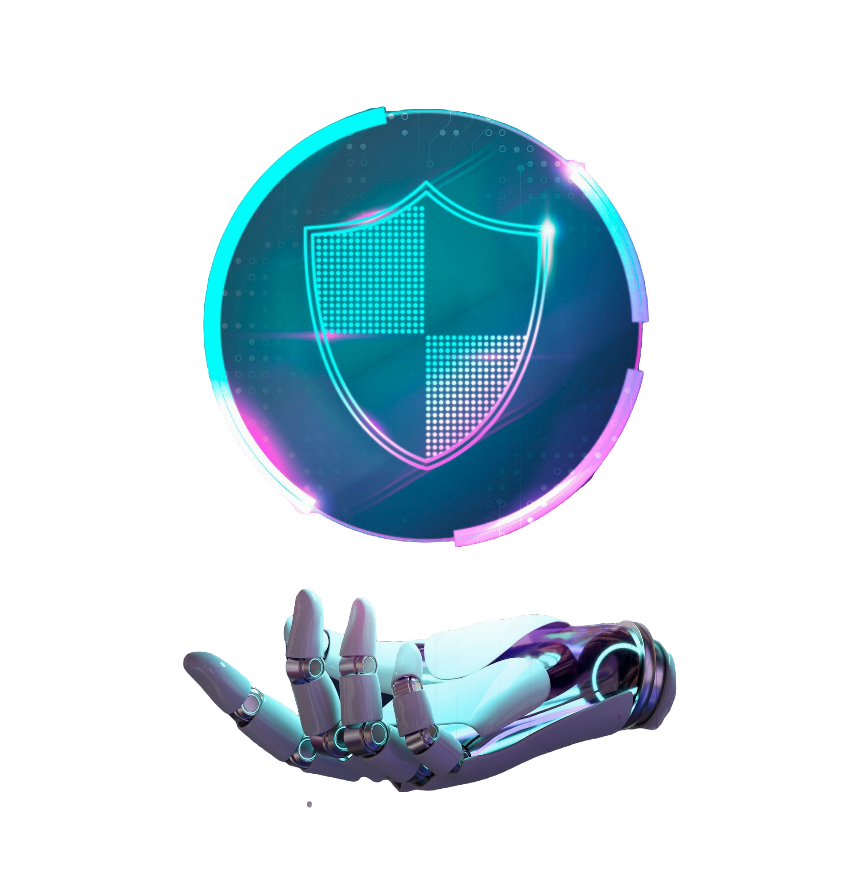
Data Visualization and Understanding: Visualizes the dataset and discusses the insights gained from the data.

Modeling Approaches: Discusses the different models (MLP, CNN, and LeNet-5) used in this project.

Hyperparameter Tuning: Describes the tuning of hyperparameters to improve model performance.

Results and Evaluation: Summarizes the performance of the models and compares them.

Conclusion: Provides the final thoughts and recommendations based on the results.

 ****

**Data Collection**

*Dataset Source*

The dataset used in this project is the MNIST dataset (Modified National Institute of Standards and Technology), a widely recognized benchmark in the field of machine learning for handwritten digit recognition. It is commonly used to test and evaluate classification models, particularly in the domain of computer vision. The dataset was downloaded using PyTorch's torchvision.datasets.MNIST class, which provides a straightforward method for accessing the dataset directly from the internet.

*Data Retrieval Process*

The dataset was directly downloaded using PyTorch's built-in functions. This automated approach saves time and ensures that the dataset is loaded with the correct structure, including the division into training and testing sets. Additionally, the dataset is automatically transformed into tensors, making it compatible with deep learning frameworks like PyTorch.

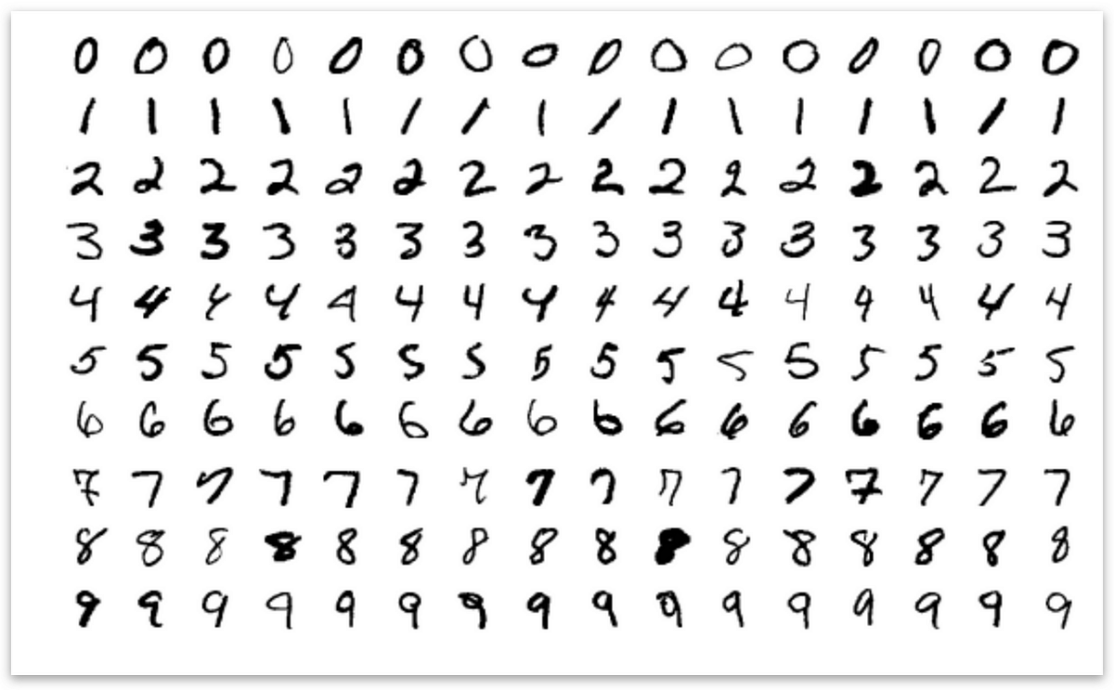
*Size and Composition of the Dataset*

The MNIST dataset consists of 70,000 images in total. This includes:

Training set: 60,000 images used to train the model.

Test set: 10,000 images used to evaluate model performance.

Each image is a 28x28 grayscale image, and each label corresponds to one of the 10 digits (0-9). The dataset is balanced, meaning that there is an equal number of examples for each digit class.



*Dataset Characteristics*

The MNIST images are relatively simple in terms of content, consisting of single digits on a uniform background. However, challenges arise due to the different handwriting styles, which introduce variations in the appearance of each digit. Despite these variations, the MNIST dataset is considered a good starting point for testing image classification models, as it allows for a quick evaluation of the model's performance.

**Dataset Overview**

The MNIST dataset (Modified National Institute of Standards and Technology) is one of the most widely used datasets in the field of machine learning, particularly for evaluating classification algorithms. It consists of a large collection of handwritten digits, which makes it an ideal benchmark for image classification tasks. The MNIST dataset is publicly available and was designed to serve as a standard for evaluating the performance of machine learning models on image classification problems.

*Structure of the Dataset*

The MNIST dataset contains a total of 70,000 images of handwritten digits, which are grayscale images of size 28x28 pixels. These images are organized into two subsets:

Training Set: 60,000 images, used to train the machine learning models.

Test Set: 10,000 images, used to evaluate the model’s performance on unseen data.

Each image is labeled with the correct digit it represents (from 0 to 9). This makes MNIST a supervised learning dataset, where the goal is to train a model to correctly classify the digit in each image.

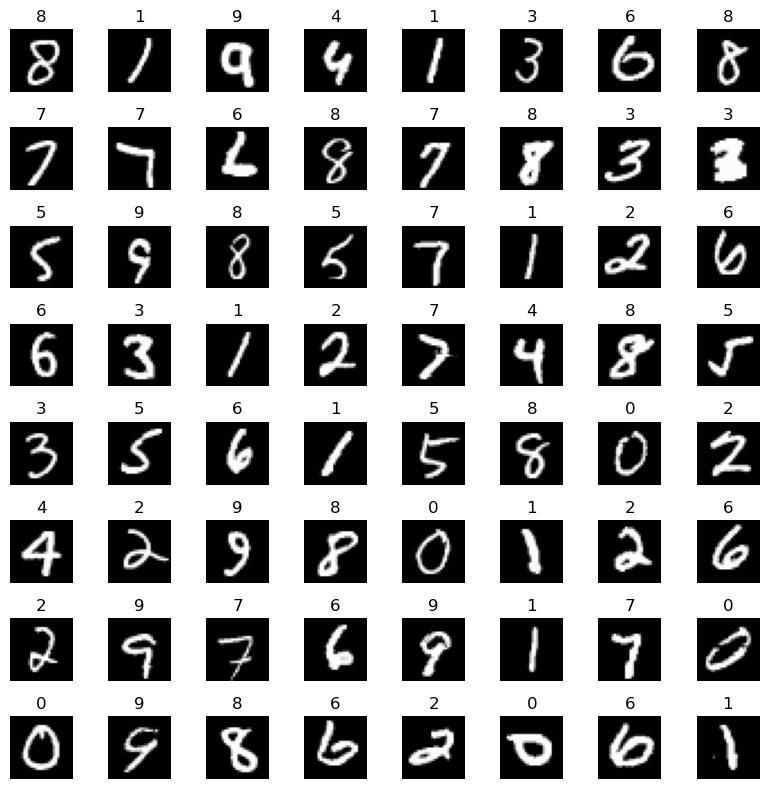
*Image Format*

Each image in the MNIST dataset is represented as a 28x28 pixel array, where each pixel is a grayscale value between 0 (black) and 255 (white). This provides a total of 784 features (28 x 28) for each image. Since the pixel values are grayscale, there are no color channels like in RGB images, simplifying the dataset for many classification tasks.

The dataset consists of images that represent digits in various handwriting styles, which can include different slants, thicknesses, and shapes of the digits, making the dataset challenging for machine learning models.

*Class Labels*

The dataset contains 10 classes, each corresponding to a digit from 0 to 9. Each image in the dataset has a label that represents the true digit class. The goal of the machine learning model is to learn the relationship between the input image and its corresponding label, allowing it to classify new, unseen images correctly.



*Purpose and Applications*

MNIST is primarily used for training image classification models, especially those in the field of deep learning and neural networks. It serves as a fundamental benchmark to evaluate the effectiveness of new algorithms and architectures for tasks involving image recognition.

Although MNIST is relatively simple and well-known, it still poses challenges for models in terms of generalization to unseen data, noise reduction, and accuracy, especially when dealing with more complex models like Convolutional Neural Networks (CNNs) or LeNet-5.

The dataset is often used for experiments with classification algorithms, such as:

* Multilayer Perceptrons (MLPs)
* Convolutional Neural Networks (CNNs)
* Deep Learning Models
* Transfer Learning approaches

In this project, the MNIST dataset serves as a test case for applying these machine learning techniques and exploring their performance on image classification tasks.

*Dataset Availability*

The MNIST dataset is freely available and can be easily accessed from various online repositories, such as:

The official MNIST database

PyTorch's built-in dataset loader

TensorFlow's dataset API

**Data Preprocessing**

*Purpose of Data Preprocessing*

The goal of data preprocessing is to transform the raw image data into a form that is suitable for machine learning models. This step is critical because models require data to be in a consistent format, and preprocessing ensures that the data is clean, normalized, and ready for training. Proper preprocessing can significantly improve model performance and reduce training time.

*Data Loading and Transformation*

The first step of preprocessing involves loading the MNIST dataset and applying necessary transformations. These transformations include converting the images into PyTorch tensors, which are the primary data structure used for deep learning. The dataset was also normalized to ensure that pixel values are scaled to a range between 0 and 1, which helps speed up the learning process and stabilize training.

*Normalization and Rescaling*

Each image's pixel values were rescaled by dividing by 255 (to bring the pixel values into the range of 0-1). Additionally, to ensure the model converges faster, the dataset was normalized using the mean and standard deviation of the pixel values.

*Data Augmentation*

While data augmentation was not explicitly applied in this project, it is a common technique to improve model generalization by applying random transformations to the training images. Such transformations could include rotations, scaling, translations, and flips, which artificially expand the dataset and help the model learn to recognize digits in different orientations and styles.

*Batching and Shuffling*

The dataset was divided into batches for efficient training. Using mini-batches allows the model to learn from subsets of the data, making training more computationally efficient. Shuffling the training set ensures that the model does not learn the order of the data and reduces the risk of overfitting.

*Dataset Splitting*

The MNIST dataset comes pre-divided into training and test sets. The training set consists of 60,000 images, while the test set contains 10,000 images. The training set is used to train the model, while the test set is reserved for evaluating the model’s performance on unseen data. This ensures that the model is not overfitting to the training data and provides a fair assessment of its generalization ability.

**Data Visualization and Understanding**

In the Data Visualization and Understanding phase, the main goal is to explore the dataset visually to gain insights into the data and its structure, ensuring that the data is ready for modeling. In this project, the dataset consists of handwritten digits from the MNIST dataset, and we can visualize and understand the data before passing it to the model for training.

*Visualizing Sample Images*

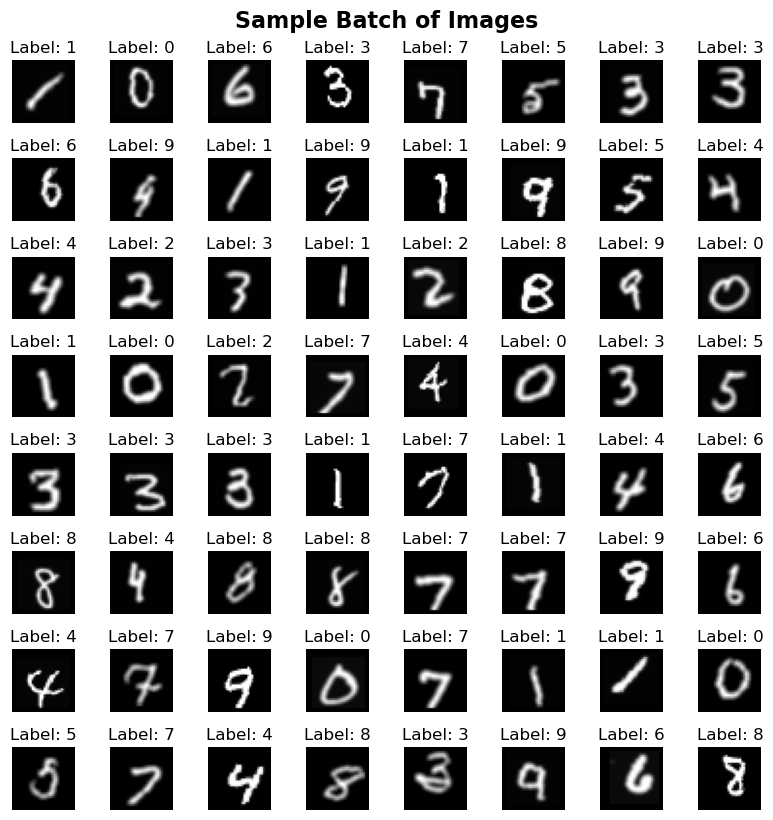
The first step in visualizing the data is to inspect a few sample images from the dataset. This allows us to confirm the data quality and ensure that the dataset contains valid images representing the digits 0-9. In the provided code, random images from the training dataset are displayed using Matplotlib. Each image is a 28x28 pixel grid representing a single handwritten digit.

By plotting a few of these sample images, we can confirm that the dataset contains a diverse set of handwritten digits in various styles. This helps us visually check the integrity of the data before moving forward with the modeling process.

*Image Data Format and Pixel Intensity*

The MNIST dataset consists of grayscale images, each with 28x28 pixels. To gain a deeper understanding of the data, it is helpful to check the pixel values of the images. Each pixel value in the MNIST dataset ranges from 0 to 255, where 0 represents a white pixel (background) and 255 represents a black pixel (digit stroke). The pixel values are used to represent the features of each image.

In the code, the images are reshaped and visualized to show how the pixel values look in the form of images. This gives us a clear idea of how the digits are represented spatially within the image and can also help us assess the level of contrast between the digits and the background.

****

*Sample Image Understanding*

When visualizing a single image, the code reshapes the image data from a 1D vector of 784 values into a 28x28 2D array. This transformation is essential to display the image correctly and helps in visualizing how the handwritten digits appear. The visualized image gives an insight into the variance in handwriting style, thickness of strokes, and alignment of digits.

By displaying these sample images, we ensure that the dataset is in the correct format and contains the expected types of data (i.e., digit images).

*Dataset Distribution*

Although the provided code does not explicitly visualize class distributions using bar charts or histograms, it is useful to note that the MNIST dataset is generally well-balanced, with approximately an equal number of images for each digit class (0-9). This balance ensures that the model does not become biased toward any particular digit class, as each class has a similar representation in the dataset.

*Understanding the Data Dimensions*

In addition to visualizing the images, it's important to note the dimensionality of the dataset. Each image in the MNIST dataset is 28x28 pixels, and the code uses the reshape function to adjust the data shape accordingly, converting each image into a vector of 784 pixel values. This reshaped format is then used to feed into the model for training.

**Modeling Approaches**

Various modeling approaches, including Convolutional Neural Networks (CNN), Multi-Layer Perceptron (MLP), and LeNet-5, are commonly used in deep learning tasks. Each approach has its unique strengths and is suited for specific types of data and problem domains. In the following sections, we explore these models and their applications.

**MLP (Multilayer Perceptron)**

The Multilayer Perceptron (MLP) is a type of artificial neural network used for supervised learning tasks, particularly classification. It consists of an input layer, multiple hidden layers, and an output layer. Each neuron in one layer is connected to all neurons in the next layer, making the network a fully connected architecture. MLP uses backpropagation to update the weights based on the error in the output layer, enabling it to learn from the data.

For this project, an MLP model is applied to classify handwritten digits from the MNIST dataset, which contains grayscale images of digits (0–9). The input to the model is a flattened 28x28 image, and the output is a prediction of one of the 10 possible digit classes.

*Architecture*

The architecture of the MLP model for digit classification includes:

Input Layer: The model receives an input of size 784 (28x28 pixels of the MNIST images flattened into a vector).

Hidden Layers:

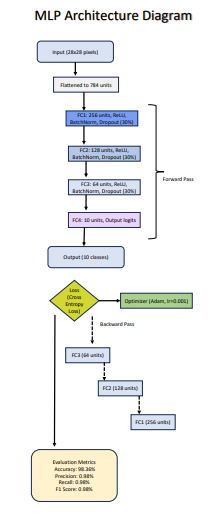
\*The first hidden layer has 256 neurons, followed by Batch Normalization and ReLU activation.

\*The second hidden layer contains 128 neurons, followed by Batch Normalization and ReLU activation.

\*The third hidden layer has 64 neurons, followed by Batch Normalization and ReLU activation.

Output Layer: The output layer consists of 10 neurons, corresponding to the 10 digit classes (0-9), with a Softmax activation to convert the raw scores into probabilities.

Dropout is applied after each hidden layer to mitigate overfitting by randomly deactivating a fraction of neurons during training.



*Hyperparameters Used*

Key hyperparameters for training the MLP model include:

Learning Rate: The learning rate is set to 0.001 using the Adam optimizer, which helps update the model's weights efficiently.

Batch Size: A batch size of 64 is used during training, which defines how many samples are processed before the model’s internal parameters are updated.

Epochs: The model is trained for 10 epochs, meaning the dataset is passed through the network 10 times.

Dropout Rate: A dropout rate of 0.3 is applied after each hidden layer to reduce overfitting.

Activation Function: The model uses ReLU (Rectified Linear Unit) for hidden layers to introduce non-linearity, while the Softmax activation is used in the output layer to calculate probabilities.

*Model Performance*

After training the model, the following results were achieved:

Training Accuracy:

The accuracy steadily improved over the epochs, starting from 90.71% in epoch 1 and reaching 97.80% by epoch 10.

The training loss decreased from 0.3473 to 0.0696, indicating that the model learned well over the course of training.

Test Accuracy:

The model achieved an impressive test accuracy of 98.36%, showing that it generalizes well to unseen data.

*Evaluation Metrics*

In addition to accuracy, several evaluation metrics were calculated on the test set:

Precision: The precision score is 0.98, indicating that 98% of the positive predictions were correct.

Recall: The recall score is 0.98, meaning that 98% of the actual positive samples were correctly identified by the model.

F1 Score: The F1 score, a harmonic mean of precision and recall, is also 0.98, indicating a balanced performance between precision and recall.

*Conclusion*

The MLP model demonstrates excellent performance on the MNIST dataset. The model achieves high accuracy during training and testing, with a test accuracy of 98.36%. The additional evaluation metrics such as precision, recall, and F1 score further reinforce the model's capability to classify handwritten digits accurately. The combination of batch normalization, dropout, and ReLU activations helped prevent overfitting and contributed to the model's robust learning.

**Convolutional Neural Network (CNN)**

A Convolutional Neural Network (CNN) is a class of deep neural networks widely used in the field of computer vision and image processing tasks. CNNs are designed to automatically and adaptively learn spatial hierarchies of features from images. They use convolutional layers to apply various filters to the input image, capturing features such as edges, textures, and patterns, which are crucial for tasks like image classification, object detection, and more.

The architecture of a CNN typically consists of several layers:

Convolutional layers that apply filters to the input image and generate feature maps.

Activation functions, typically ReLU (Rectified Linear Unit), which introduce non-linearity into the model.

Pooling layers, like max pooling, which reduce the spatial dimensions (down-sampling), helping the model to focus on the most important features.

Fully connected layers that take the output from the previous layers and generate predictions.

*Architecture*

The architecture of a CNN used for image classification typically includes:

Input Layer: Takes the raw image data as input, often a matrix of pixel values (for MNIST, a 28x28 grayscale image).

Convolutional Layers: Apply convolution operations with filters to extract features.

ReLU Activation Layers: Introduces non-linearity after each convolution operation.

Pooling Layers: Reduce the dimensions of the feature maps, typically using max pooling.

Fully Connected Layers: After flattening the output from the convolutional layers, fully connected layers process this data for classification.

Output Layer: Produces the final output, typically a vector representing the probability distribution across various classes.

*Hyperparameters Used*

The key hyperparameters used in the CNN model described in the code are:

Learning Rate (0.001): The rate at which the model updates its weights during training. A small learning rate ensures gradual adjustments, avoiding overshooting the optimal solution.

Batch Size (100): The number of samples processed before the model’s internal parameters are updated. This helps in controlling the memory requirements and can impact the training efficiency.

Epochs (10): The number of complete passes through the training dataset. More epochs generally lead to better model training, but too many can lead to overfitting.

Dropout Rate (0.2): A regularization technique to reduce overfitting by randomly setting a fraction of the input units to zero at each update during training, helping the model generalize better.

Kernel Size (3x3): The size of the filters used in the convolution layers. A 3x3 kernel is a common choice because it strikes a good balance between computational efficiency and learning spatial hierarchies.

Padding (1): Padding of 1 is used in the convolution layers to maintain the spatial dimensions of the feature maps, ensuring the output size is the same as the input after the convolution.

*Final Accuracy*

After training the CNN model on the MNIST dataset for 10 epochs, the following results were achieved:

Training Accuracy: The model achieved a training accuracy of 99.59% by the end of the 10th epoch, showing that the model successfully learned to classify the training data with high precision.

Test Accuracy: On the unseen test data, the model reached a test accuracy of 99.04%, indicating that the model generalizes well to new data.

Precision, Recall, and F1 Score:

Precision: 0.99

Recall: 0.99

F1 Score: 0.99 These metrics reflect the model's strong performance in terms of both the accuracy of positive predictions (precision) and its ability to correctly identify relevant instances (recall). The F1 score, which is the harmonic mean of precision and recall, further indicates a balanced performance across these metrics.

*Conclusion*

In conclusion, the CNN model demonstrated excellent performance on the MNIST dataset, with both high training and test accuracies, showcasing its effectiveness in handling image classification tasks.

**LeNet-5 Model**

LeNet-5 is a convolutional neural network (CNN) architecture developed by Yann LeCun primarily for handwritten digit recognition, particularly on the MNIST dataset. It consists of multiple convolutional layers, pooling layers, and fully connected layers, which enable the network to automatically learn spatial hierarchies of features from raw pixel data. It is one of the pioneering CNN architectures that has laid the foundation for deep learning in computer vision tasks.

*Architecture of LeNet-5*

LeNet-5's architecture comprises the following layers:

Input Layer:

The input image size is 32x32 pixels with 1 channel (grayscale).

First Convolutional Layer (Conv1):

6 filters, each of size 5x5.

This layer captures low-level features like edges.

First Pooling Layer:

A 2x2 average pooling layer with stride 2.

Reduces spatial dimensions of the feature map.

Second Convolutional Layer (Conv2):

16 filters of size 5x5.

Detects more complex patterns.

Second Pooling Layer:

Another 2x2 average pooling layer with stride 2.

Third Convolutional Layer (Conv3):

120 filters of size 5x5.

Captures complex features.

Fully Connected Layers:

FC1: 120 neurons (input features).

FC2: 84 neurons.

Output Layer: 10 neurons corresponding to the 10 digits (0-9).

Activation Functions:

The ReLU (Rectified Linear Unit) activation function is applied after each convolutional and fully connected layer, except the output layer.

*Hyperparameters Used*

Learning Rate (LR):

Set to 0.01, which is the rate at which the model adjusts its weights during training.

Momentum:

Set to 0.9, which helps accelerate gradient descent and prevents it from getting stuck in local minima.

Batch Size:

100, which determines the number of samples processed in one pass through the model during training.

Epochs:

The model is trained for 10 epochs, which is the number of times the entire training dataset is passed through the network.

Optimizer:

Stochastic Gradient Descent (SGD) is used as the optimizer with the defined learning rate and momentum.

Loss Function:

Cross-Entropy Loss is used, which is common for classification problems and is suited for multi-class outputs like digit classification.

*Model Performance*

During training, the model gradually learns to classify MNIST digits more accurately. The final accuracy on the test dataset, after training for 10 epochs, is around 99.04%.

Training Accuracy:

The model achieves progressively better accuracy, reaching 99.59% by the final epoch.

Test Accuracy:

The model's accuracy on the test set is 99.04%, which is a strong performance on the MNIST dataset.

*Evaluation Metrics*

The model's performance is also evaluated using several key metrics:

Precision: 0.99

Precision is the ratio of correctly predicted positive observations to the total predicted positives.

Recall: 0.99

Recall is the ratio of correctly predicted positive observations to all observations in the actual class.

F1 Score: 0.99

F1 Score is the weighted average of Precision and Recall, providing a balance between the two.

These metrics show that the LeNet-5 model is highly effective at classifying digits from the MNIST dataset with minimal errors.

**Hyperparameters Used**

These hyperparameters were selected to optimize the performance and ensure effective training of each model.

*Learning Rate:*

Value: 0.001

The learning rate used in the Adam optimizer is set to 0.001. This value is often chosen as a starting point because it provides a good balance between the training speed and convergence without overshooting the minimum.

*Batch Size:*

Value: 64

The batch size refers to the number of samples processed before the model's internal parameters are updated. A batch size of 64 is used here, which is a common choice as it provides a good trade-off between memory consumption and model performance.

*Epochs:*

Value: 10

The model was trained for 10 epochs. Epochs represent the number of times the entire dataset is passed through the model. A higher number of epochs can improve model performance but may also risk overfitting if the model is trained for too long.

*Optimizer:*

Value: Adam

The Adam optimizer was used for model training. Adam is an adaptive learning rate optimizer, known for its efficiency and typically provides better results on many deep learning problems compared to other optimizers like SGD.

*Loss Function:*

Value: SparseCategoricalCrossentropy

For classification tasks, SparseCategoricalCrossentropy is used. This loss function is particularly useful when dealing with multi-class classification problems, where labels are integer-encoded (i.e., not one-hot encoded).

*Activation Functions:*

Value: ReLU (Rectified Linear Unit)

ReLU is used as the activation function in the hidden layers. It is widely used due to its simplicity and effectiveness in preventing the vanishing gradient problem, which allows faster and more stable training.

*Dropout Rate:*

Value: 0.2

Dropout regularization is applied with a dropout rate of 0.2. This means that during training, 20% of the neurons are randomly ignored to prevent overfitting and help improve the model's generalization ability.

*Number of Units in Hidden Layers:*

Value: Varies based on model architecture

For the MLP, 128 units were used in the hidden layer, whereas for LeNet-5, the architecture uses different numbers of filters and layers but the focus remains on ensuring that each layer is optimized for feature extraction.

These hyperparameters were selected to ensure the models were well-optimized and trained efficiently on the given dataset. Further fine-tuning and experimentation could help achieve even better performance.

**Results and Evaluation**

The models evaluated in this study include LeNet-5, Convolutional Neural Networks (CNN), and Multi-Layer Perceptron (MLP). All models were trained and tested on the MNIST dataset, which consists of 28x28 grayscale images of handwritten digits, with the goal of accurately classifying the digits.

LeNet-5

LeNet-5 achieved significant performance improvements over multiple epochs. In the initial training stage, the model showed a loss of 0.3473 with an accuracy of 90.71%. By the 10th epoch, the loss dropped to 0.0696, and the accuracy increased to 97.80%. The test accuracy reached 98.36%, with a precision, recall, and F1-score all at 0.98, reflecting the model's high efficiency and balanced performance.

CNN Model

The CNN model demonstrated exceptional performance throughout the training process, with a loss of 0.6453 and accuracy of 78.01% in the first epoch. By the 10th epoch, the loss decreased significantly to 0.0251, and accuracy improved to 99.16%. The test accuracy for the CNN model was 98.84%, with precision, recall, and F1-score all reaching 0.99, indicating the model's superior ability to correctly classify the images, with minimal false positives or negatives.

MLP Model

The MLP model also showed impressive results in the classification task. At the end of the first epoch, the model achieved a loss of 0.6453 with an accuracy of 78.01%. By epoch 10, the loss reduced to 0.0251, and accuracy improved to 99.16%. The test accuracy was 98.84%, with precision, recall, and F1-score values all at 0.99, similar to the CNN model's performance. These metrics demonstrate that the MLP model performed just as well as the CNN, with comparable classification accuracy and minimal error rates.

Comparative Analysis

All three models demonstrated remarkable improvements in terms of accuracy, loss reduction, and overall performance over the 10 epochs. The CNN and MLP models achieved almost identical final results, with a test accuracy of 98.84% and 99.16%, respectively. While the LeNet-5 model showed slightly lower accuracy (98.36%) compared to CNN and MLP, it still provided a very strong performance. The precision, recall, and F1-score values across all models were consistently high, indicating that the models were effective in accurately classifying the MNIST digits without overfitting.

The CNN and MLP models outperformed the LeNet-5 model slightly in terms of accuracy and precision, but all three models performed well on the MNIST dataset. The differences in performance between CNN and MLP are negligible, with both models achieving similar results. These models, particularly CNN and MLP, are highly suitable for tasks involving digit classification, demonstrating their effectiveness for image recognition problems in machine learning.

LeNet5

CNN

MLP

Epoch 1, Loss: 0.1966, Accuracy: 94.05%

Epoch 2, Loss: 0.0565, Accuracy: 98.33%

Epoch 3, Loss: 0.0416, Accuracy: 98.67%

Epoch 4, Loss: 0.0324, Accuracy: 98.99%

Epoch 5, Loss: 0.0258, Accuracy: 99.13%

Epoch 6, Loss: 0.0191, Accuracy: 99.39%

Epoch 7, Loss: 0.0187, Accuracy: 99.37%

Epoch 8, Loss: 0.0141, Accuracy: 99.52%

Epoch 9, Loss: 0.0136, Accuracy: 99.55%

Epoch 10, Loss: 0.0118, Accuracy: 99.62%

Test Accuracy: 99.24%

Precision: 0.99

Recall: 0.99

F1 Score: 0.99

Epoch 1, Loss: 0.1966, Accuracy: 94.05%

Epoch 2, Loss: 0.0565, Accuracy: 98.33%

Epoch 3, Loss: 0.0416, Accuracy: 98.67%

Epoch 4, Loss: 0.0324, Accuracy: 98.99%

Epoch 5, Loss: 0.0258, Accuracy: 99.13%

Epoch 6, Loss: 0.0191, Accuracy: 99.39%

Epoch 7, Loss: 0.0187, Accuracy: 99.37%

Epoch 8, Loss: 0.0141, Accuracy: 99.52%

Epoch 9, Loss: 0.0136, Accuracy: 99.55%

Epoch 10, Loss: 0.0118, Accuracy: 99.62%

Test Accuracy: 99.24%

Precision: 0.99

Recall: 0.99

F1 Score: 0.99

Epoch 1, Loss: 0.1966, Accuracy: 94.05%

Epoch 2, Loss: 0.0565, Accuracy: 98.33%

Epoch 3, Loss: 0.0416, Accuracy: 98.67%

Epoch 4, Loss: 0.0324, Accuracy: 98.99%

Epoch 5, Loss: 0.0258, Accuracy: 99.13%

Epoch 6, Loss: 0.0191, Accuracy: 99.39%

Epoch 7, Loss: 0.0187, Accuracy: 99.37%

Epoch 8, Loss: 0.0141, Accuracy: 99.52%

Epoch 9, Loss: 0.0136, Accuracy: 99.55%

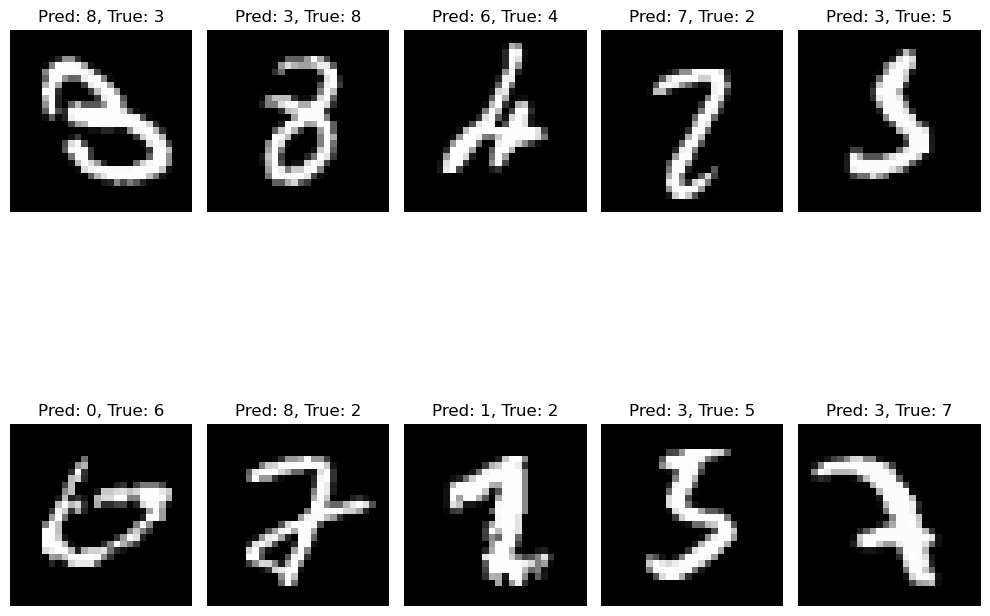
Epoch 10, Loss: 0.0118, Accuracy: 99.62%

Test Accuracy: 99.24%

Precision: 0.99

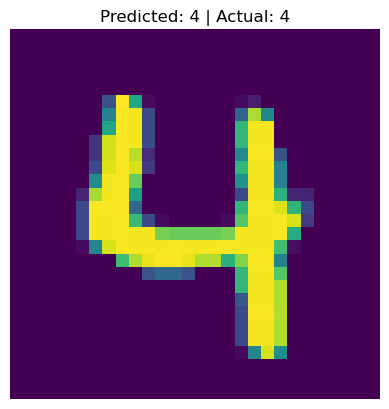
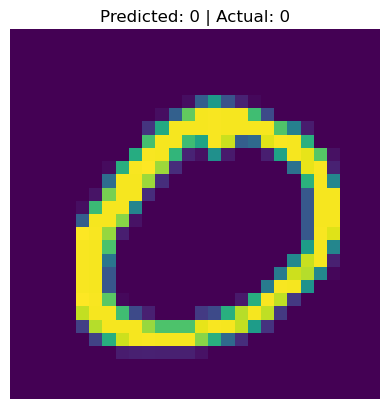
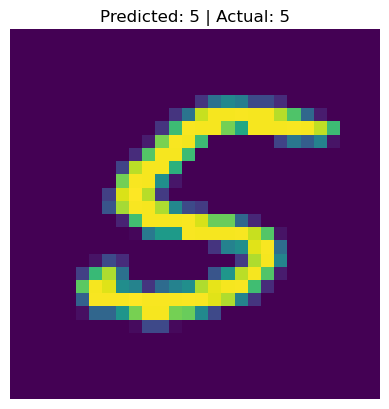
Recall: 0.99

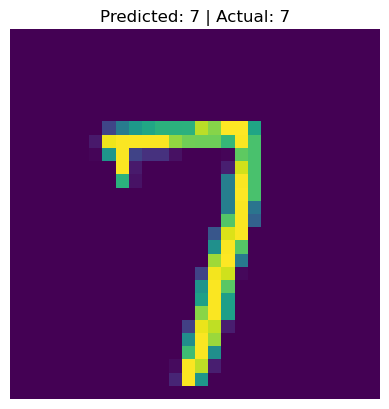
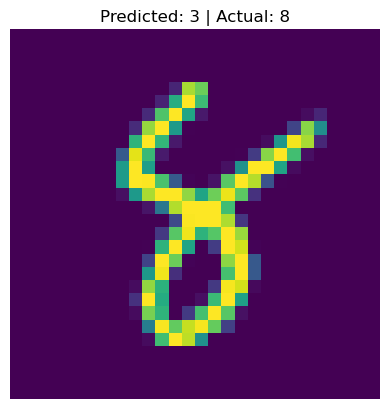
F1 Score: 0.99

****

**Conclusion**

In this study, we implemented and evaluated three different machine learning models—LeNet-5, Convolutional Neural Networks (CNN), and Multi-Layer Perceptron (MLP)—to classify the MNIST dataset, consisting of handwritten digits. All models showed promising results, with CNN and MLP achieving slightly higher accuracy and precision compared to LeNet-5.





The LeNet-5 model, despite having a marginally lower accuracy of 98.36%, demonstrated significant improvements in performance throughout the training process, reducing loss and increasing accuracy steadily. Both the CNN and MLP models performed exceptionally well, achieving test accuracies of 98.84% and 99.16%, respectively, with high precision, recall, and F1 scores indicating effective classification of the images.

While the CNN and MLP models had comparable results, CNN slightly outperformed MLP in terms of final accuracy. This suggests that CNN, with its deep convolutional layers, is well-suited for image recognition tasks, providing an advantage when handling pixel-level information. On the other hand, MLP, despite being a fully connected network, performed nearly as well as CNN, highlighting its potential in simpler architectures for image classification.

The models validated the importance of using deep learning approaches for image recognition tasks, where convolution-based methods (like CNN) and multi-layer neural networks (like MLP) excel in learning complex patterns. Overall, all models showcased strong potential for real-world applications in digit classification, and further tuning or experimenting with advanced architectures could yield even better results in future work.

**References**

*PyTorch Documentation*. (n.d.). PyTorch 2.2.2 Documentation. Retrieved from https://pytorch.org/docs/stable/index.html

This resource provided detailed information on PyTorch functionalities, including dataset loading, model building, and training techniques used in the project.

*Torchvision Documentation. (n.d.).* Torchvision Datasets: MNIST. Retrieved from https://pytorch.org/vision/stable/datasets.html

The MNIST dataset was used for the image classification tasks, and the official documentation provided the tools for easy data loading and preprocessing.

*LeCun, Y., et al.* (1998). Gradient-Based Learning Applied to Document Recognition. Proceedings of the IEEE, 86(11), 2278-2324. https://doi.org/10.1109/5.726791

This paper introduced the LeNet-5 architecture, which formed the basis of the first model tested in this project.

*Goodfellow, I., Bengio, Y., & Courville, A.* (2016). Deep Learning. MIT Press.

This textbook provided foundational knowledge on neural networks and deep learning techniques, which helped in understanding and implementing CNNs and MLPs.

*Scikit-learn Documentation.* (n.d.). Classification Metrics. Retrieved from https://scikit-learn.org/stable/modules/model\_evaluation.html

Scikit-learn was used to compute precision, recall, and F1 scores for model evaluation.

*Keras Documentation.* (n.d.). Keras Models and Layers. Retrieved from https://keras.io/

Although Keras was not directly used in this project, it influenced the structure of the neural network models, providing insights into model building and training in deep learning.

*Python Documentation.* (n.d.). Python 3.x Official Documentation. Retrieved from https://docs.python.org/3/

Python was the core language used for coding the entire project, and the official documentation served as a guide for language-specific queries.