LearnerSpace: Handwritten Digit Recognition Using Convolutional Neural Networks (CNN)

Submitted by: Arjun Singh (23B1272)

July 3, 2025

Contents

1 Introduction **2**

2 Dataset Overview **2**

3 Model Architecture **2**

4 Training and Validation **2**

4.1 Setup 2

4.2 Training Results 3

5 Test Set Evaluation **3**

6 Inference on Custom Image **3**

7 Confusion Matrix **4**

8 Performance Analysis **4**

8.1 Observations 4

8.2 Expected vs Actual 5

9 Challenges and Improvements **5**

9.1 Challenges 5

9.2 Proposed Improvements 5

10 Conclusion **5**

1 Introduction

Handwritten digit recognition is a classical machine learning problem and a common benchmark for evaluating computer vision techniques. This project focuses on implementing a Convolutional Neural Network (CNN) using PyTorch to classify digits from the MNIST dataset. The project workflow includes model creation, training, evaluation, and inference on external images.

2 Dataset Overview

The MNIST dataset contains 70,000 grayscale images of handwritten digits (60,000 for training and 10,000 for testing), each sized 28×28 pixels. Each image is labeled with a digit from 0 to 9. Data is preprocessed using 'ToTensor()' transformation which normalizes the pixel values to the [0, 1] range.

3 Model Architecture

The CNN model used consists of the following layers:

- **Conv2D Layer 1:** 1 input channel, 64 output channels, kernel size 3
- **MaxPooling Layer 1:** kernel size 2
- **Conv2D Layer 2:** 64 input and output channels, kernel size 3
- **MaxPooling Layer 2:** kernel size 2
- **Conv2D Layer 3:** 64 input and output channels, kernel size 3
- **MaxPooling Layer 3:** kernel size 2
- **Flatten**
- **Fully Connected Layers:** [64 * 1 * 1 → 64 → 32 → 10]

The final layer outputs a vector of size 10 corresponding to the class scores for digits 0–9.

4 Training and Validation

4.1 Setup

- Optimizer: Adam
- Loss Function: CrossEntropyLoss
- Epochs: 5
- Batch Size: 64
- Train/Validation Split: 70/30

4.2 Training Results

Epoch	Loss	Train Accuracy	Validation Accuracy
1	0.718	76.04%	92.80%
2	0.255	94.00%	96.00%
3	0.191	96.30%	97.30%
4	0.159	97.14%	97.60%
5	0.130	97.90%	98.10%

Table 1: Training and Validation Accuracy per Epoch

5 Test Set Evaluation

The trained model was evaluated on the 10,000-sample MNIST test set.

- **Test Accuracy:** 98.35%

6 Inference on Custom Image

A custom handwritten digit image (‘4.png‘) was loaded using OpenCV, converted to grayscale, resized to 28x28, normalized, and passed to the trained model.

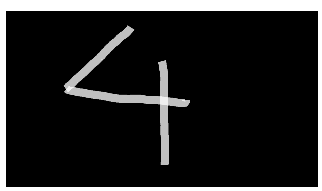

Figure 1: Custom Digit Image (Displayed in RGB)

Prediction: 4

7 Confusion Matrix

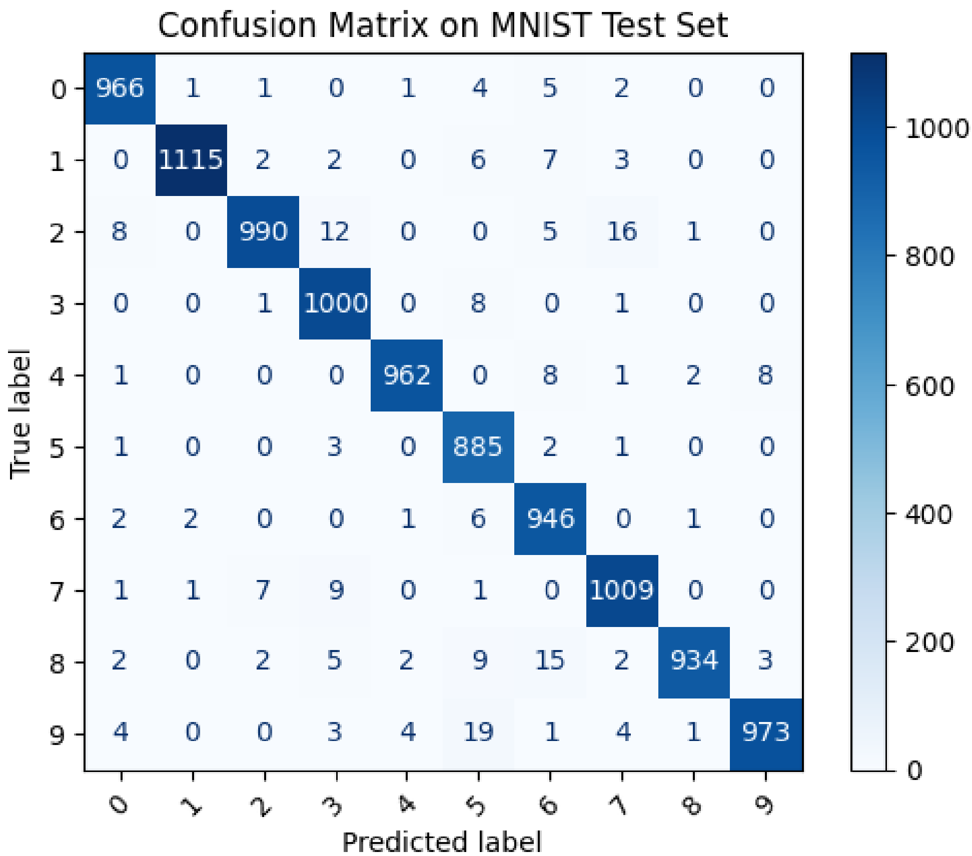

Figure 2: Confusion Matrix on MNIST Test Set

The model performed well across all digit classes, with most misclassifications occurring between visually similar digits such as 4 and 9, and 5 and 3.

8 Performance Analysis

8.1 Observations

- The model achieved high accuracy on both validation and test sets, suggesting it generalizes well.
- Confusion matrix reveals most confusion between digits that are structurally similar.
- Performance is consistent and robust with minimal overfitting.

8.2 Expected vs Actual

The expected accuracy was around 97%, and the model surpassed it with a final accuracy of 98.35%. This aligns well with performance seen in standard CNN-based MNIST classifiers.

9 Challenges and Improvements

9.1 Challenges

- Ensuring the input dimensions match after each convolution and pooling layer.
- Handling external images which often differ in thickness, position, or background noise.

9.2 Proposed Improvements

1. **Data Augmentation:** Introduce transformations like rotation, zoom, and shift to make the model robust to variations in handwriting.
2. **Batch Normalization and Dropout:** Adding batch normalization layers can speed up training and stabilize the learning process, while dropout can prevent overfitting.

10 Conclusion

This project demonstrates the application of Convolutional Neural Networks in recognizing handwritten digits. The implemented model achieves over 98% accuracy on the MNIST dataset and performs well on external images. With further improvements such as data augmentation and regularization techniques, the model can become even more robust and production-ready.

References

- PyTorch Documentation
- MNIST Dataset
- Scikit-learn Confusion Matrix