Final Project Report

Handwritten Digit Classification using CNN

Learners Space'25

1. Project Objective

The goal of this project is to build and train a Convolutional Neural Network (CNN) model capable of classifying handwritten digits using the MNIST dataset. This project demonstrates practical application of data preprocessing, CNN design, training loops, evaluation metrics and model generalization through data augmentation.

2. Background

The MNIST dataset contains 70,000 images of handwritten digits (0 to 9), where each image is a 28x28 grayscale pixel grid. It is widely used as a benchmark for image classification models. Applications include digit recognition in postal services, banking and OCR systems.

3. Data Preparation and Augmentation

Loading and Exploring

We used the PyTorch `torchvision.datasets.MNIST` class to download and load the dataset into training and test loaders.

Normalization

All images were converted to tensors and normalized using:

$$\text{Mean} = 0.1307, \quad \text{Standard Deviation} = 0.3081$$

Normalization ensures that input features are on a similar scale, improving model convergence during training.

Data Augmentation

To reduce overfitting and enhance generalization, the following augmentations were applied to the training set:

- **RandomRotation(10°)**: Rotates the image randomly between -10 and +10 degrees.

- **RandomAffine with translation**: Applies horizontal and vertical shifts up to 10%.

4. CNN Architecture Design

Model Summary

The architecture used is a simple but effective CNN with two convolutional layers followed by two fully connected layers.

Layer-wise Description

- **Conv1:** 1 input channel, 32 output channels, kernel size 3x3, stride 1
- **ReLU Activation**
- **MaxPool1:** 2x2 kernel, stride 2
- **Conv2:** 32 input channels, 64 output channels, kernel size 3x3
- **ReLU Activation**
- **MaxPool2:** 2x2 kernel, stride 2
- **Flatten**: $64 \times 5 \times 5 = 1600$ neurons
- **FC1:** Fully connected layer with 128 output features
- **FC2:** Output layer with 10 neurons (for 10 digit classes)

Activation Function

`ReLU` was used after each convolutional and FC1 layer due to its simplicity and effectiveness.

Output Layer

The final layer outputs raw scores (logits). We used `CrossEntropyLoss`, which internally applies `LogSoftmax`, making explicit activation unnecessary.

5. Model Training and Evaluation

Loss Function

We used `nn.CrossEntropyLoss()`, suitable for multi-class classification with raw logits.

Optimizer

The `Adam` optimizer was used with a learning rate of 0.001 for its fast convergence.

Training Loop

For each epoch:

1. Forward pass the images through the model
2. Compute loss
3. Perform backpropagation
4. Update weights using optimizer

Results per Epoch

Epoch	Loss	Test Accuracy
1	0.2514	98.52%
2	0.0942	98.42%
3	0.0712	99.19%
4	0.0587	99.02%
5	0.0537	**99.31%**

Prediction Examples

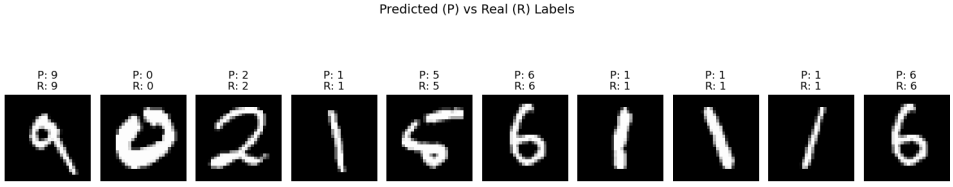

Figure 1: Random test images with Predicted (P) and Real (R) labels.

6. Analysis and Conclusion

Final Test Accuracy

The model achieved a test accuracy of **99.31%**, which indicates excellent performance for a basic CNN on the MNIST dataset.

Confusion Matrix

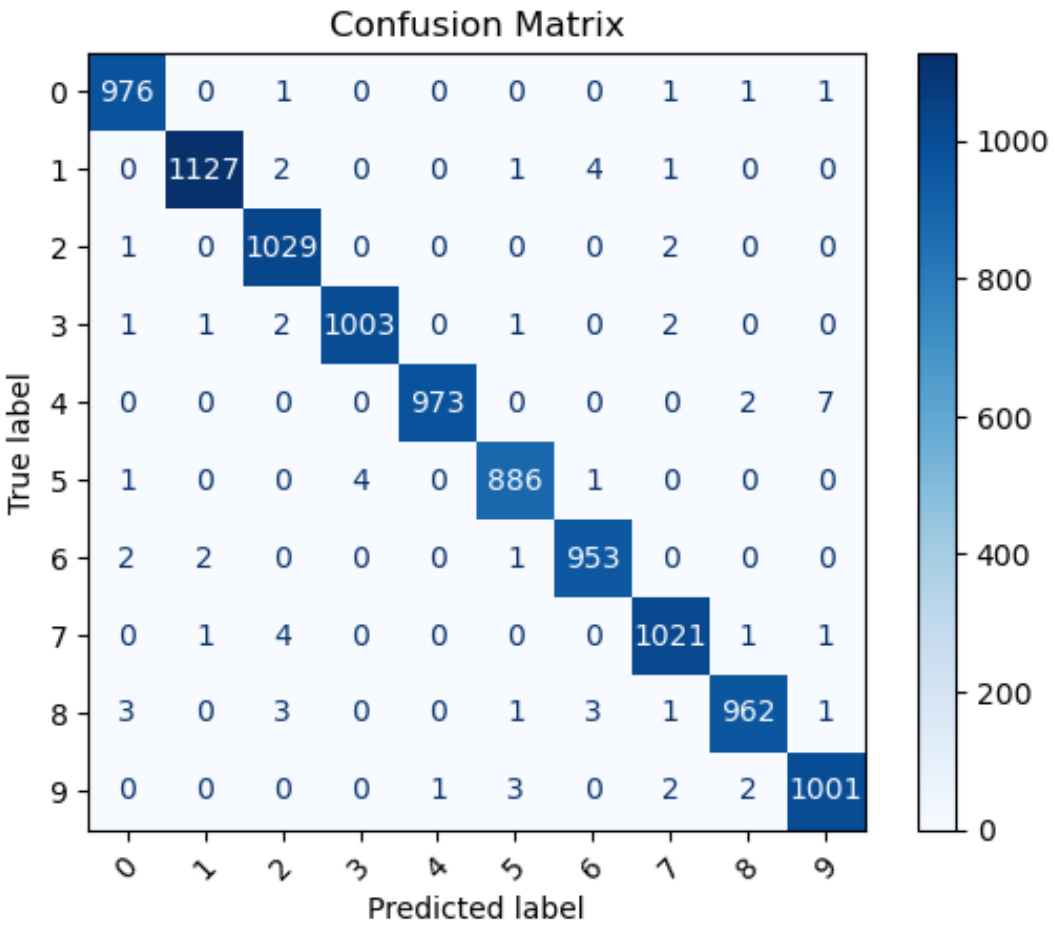

Figure 2: Confusion Matrix of Model Predictions

Performance Discussion

The model performs exceptionally well on clean handwritten digits. The few misclassifications typically occur between visually similar digits (e.g., 4 and 9, or 3 and 5), which is consistent with human-level confusion.

Overfitting Analysis

Despite high accuracy, the model does not appear overfitted:

- No gap between training and test performance
- Low training loss
- Good generalization due to augmentation

Suggested Improvements

- Add `Dropout` to FC layers to reduce overfitting on larger datasets.
- Use a learning rate scheduler to dynamically adjust learning rate.
- Incorporate early stopping to avoid unnecessary epochs.

7. Conclusion

We successfully built, trained and evaluated a CNN that achieves high accuracy on the MNIST digit classification task. Through proper preprocessing, data augmentation and model design, we created a classifier that is both efficient and effective.