SOC2025 – Assignment 1

Handwritten Digit Recognition using CNN

Submitted by
Esha Yindukuri
24B2119

Objective

Building a model which can recognize handwritten digits from the **MNIST** dataset using a **Convolutional Neural Network (CNN)**, involving layers such as **convolutional, pooling, and fully connected layers** to automatically extract features from the images and classify the digits (0-9)

Dataset

We used the **MNIST dataset**, which consists of:

- **60,000 training images** and **10,000 test images**
- Each image is a **28×28 grayscale** image
- Labels represent digits from **0 to 9**
- Each pixel has a value between 0 and 255, where 0 is black and 255 is white.

The dataset was directly loaded from **TensorFlow's Keras API**.

Data Preprocessing

Before training the model, cleaned and prepared the data using these steps:

1. **Normalization**:
   Scaled the pixel values from 0–255 down to **0–1** by dividing each pixel by 255. This helps the model learn faster.

2. **Reshaping**:
   Reshaped the images to have one more dimension (28×28×1) so that they can be used in a CNN model.

3. **One-Hot Encoding**:
   The labels (like 0, 1, 2, ..., 9) were changed into a format suitable for classification using one-hot vectors. For example, the digit "3" becomes [0, 0, 0, 1, 0, 0, 0, 0, 0, 0].

CNN Model Architecture

We created the CNN model using the **Sequential API** in TensorFlow Keras. The model includes several layers:

1. **First Convolutional Layer**
    - 32 filters of size 3×3
    - Uses ReLU activation
    - Finds simple shapes and edges

2. **First MaxPooling Layer**
    - Pool size 2×2
    - Reduces the size of the feature map

3. **Second Convolutional Layer**
    - 64 filters of size 3×3
    - ReLU activation again
    - Detects more complex patterns

4. **Second MaxPooling Layer**
    - Pool size 2×2
    - Further reduces size

5. **Flatten Layer**
    - Converts 2D output into a 1D vector

6. **Fully Connected Dense Layer**
    - 128 neurons
    - ReLU activation
    - Learns important combinations of features

7. **Dropout Layer**
    - 50% of neurons turned off randomly during training to prevent overfitting

8. **Output Layer**
    - 10 neurons (for digits 0–9)
    - Softmax activation to give probability for each class

Model Compilation

Before training, we compiled the model with these settings:

- **Optimizer**: Adam (a popular optimizer that adjusts learning rate automatically)
- **Loss Function**: Categorical Crossentropy (used for multi-class classification)
- **Metric**: Accuracy (to check how many predictions are correct)

Training the Model

Trained the model using:

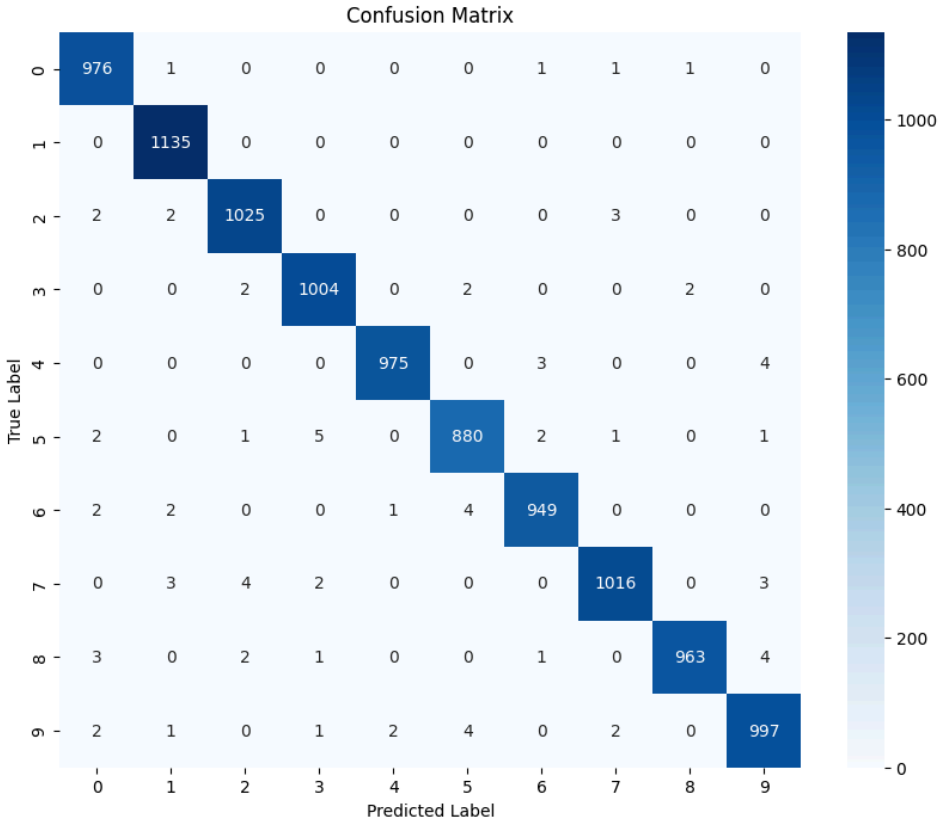
- **10 epochs** (10 times through the full training data)
- **Batch size of 128** (128 images are processed before updating the weights)
- **Validation split of 20%** (part of training data used to check performance after every epoch)

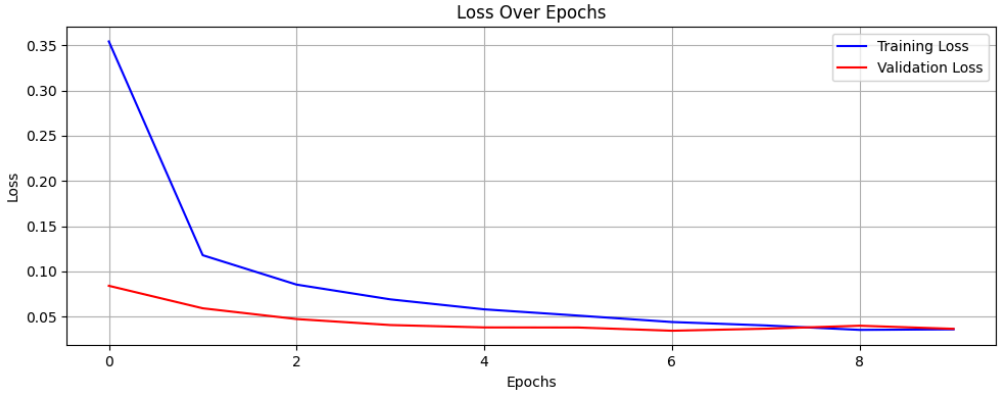
During training, tracked both **training and validation accuracy/loss**, and plotted graphs to check how the model was learning.

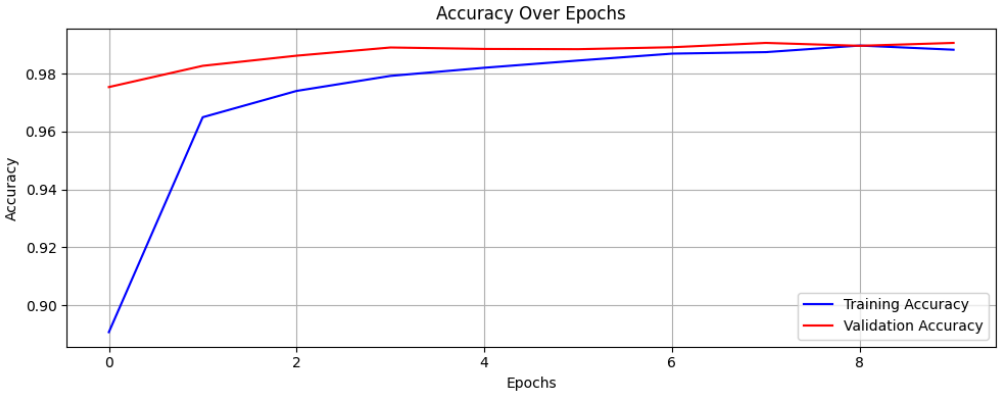
Result

After training, tested the model on the test set and got an accuracy of around **99.20%**. Also

- Made predictions on a few test images and showed both the real and predicted labels.
- Plotted a **confusion matrix** to see which digits were getting confused with each other.


Confusion Matrix
True Label
Predicted Label
0 976 1 0 0 0 0 1 1 1 0
1 0 1135 0 0 0 0 0 0 0 0
2 2 2 1025 0 0 0 0 3 0 0
3 0 0 2 1004 0 2 0 0 2 0
4 0 0 0 0 975 0 3 0 0 4
5 2 0 1 5 0 880 2 1 0 1
6 2 2 0 0 1 4 949 0 0 0
7 0 3 4 2 0 0 0 1016 0 3
8 3 0 2 1 0 0 1 0 963 4
9 2 1 0 1 2 4 0 2 0 997
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0


Loss Over Epochs
Training Loss
Validation Loss
Loss
Epochs
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0
2
4
6
8


Accuracy Over Epochs
Training Accuracy
Validation Accuracy
Accuracy
Epochs
0.98
0.96
0.94
0.92
0.90
0
2
4
6
8