# Document Recognition using CNN

## A Deep Learning Approach on MNIST Dataset

Shraddha Chavan
IISc,Bangalore

**Abstract**

This report outlines the implementation of a deep learning model using Convolutional Neural Networks (CNN) for handwritten digit recognition, applied to the MNIST dataset. The project employs various data preprocessing techniques, model optimizations, and evaluation metrics to achieve high accuracy and performance. Detailed analysis of the model architecture, learning dynamics, and evaluation metrics such as accuracy and F1 score is presented. A confusion matrix and sample test predictions are visualized to highlight the model's strengths.

# Contents

**1 Introduction** **2**

**2 Dataset and Preprocessing** **2**
2.1 Dataset Overview . . . . . 2
2.2 Data Augmentation and Normalization . . . . . 2

**3 Model Architecture** **3**
3.1 Convolutional Layers and Batch Normalization . . . . . 3
3.2 Fully Connected Layers and Dropout . . . . . 3
3.3 Optimization and Learning Rate Scheduler . . . . . 4

**4 Training and Results** **5**
4.1 Training Process . . . . . 5
4.2 Test Results . . . . . 5
4.3 Training Loss Curve . . . . . 5
4.4 Confusion Matrix . . . . . 5
4.5 Sample Test Predictions . . . . . 7

**5 Conclusion** **7**

**6 References** **8**

# 1 Introduction

Document recognition is an essential task in machine learning, especially for converting handwritten documents into digital text. In this project, we use the MNIST dataset, which contains grayscale images of handwritten digits, to develop a robust deep learning model based on Convolutional Neural Networks (CNN).

The goal is to achieve high recognition accuracy by applying data augmentation, advanced convolutional layers, and regularization techniques such as dropout. A CNN architecture is chosen due to its ability to capture local dependencies in the images, making it highly efficient for image classification tasks.

# 2 Dataset and Preprocessing

## 2.1 Dataset Overview

The MNIST dataset contains 60,000 training images and 10,000 test images, where each image is 28x28 pixels in size. Each image is labeled as one of the ten digits (0 to 9). The dataset has been a benchmark for handwritten digit recognition and is widely used for evaluating machine learning algorithms.

## 2.2 Data Augmentation and Normalization

To enhance the model's ability to generalize, data augmentation techniques are applied to the training data. Data augmentation artificially increases the diversity of the training set by creating modified versions of the original data, making the model more robust to variations such as orientation and position of the digits.

The following transformations were applied:

- **Random Horizontal Flip**: Flips images horizontally with a probability of 50%, helping the model generalize to different orientations.
- **Random Rotation**: Randomly rotates images by up to 10 degrees to simulate different handwriting styles and orientations.

- **Normalization**: Each pixel value is normalized to the range [-1, 1] using the formula:

$$\text{Normalized Value} = \frac{\text{Pixel Value} - 0.5}{0.5}$$

This standardizes the input data and accelerates the training process by stabilizing the gradient updates.

# 3 Model Architecture

We designed an advanced Convolutional Neural Network (CNN) to classify the digits. CNNs are known for their ability to capture spatial hierarchies in images through convolutional and pooling layers. The following sections explain the architecture in detail:

## 3.1 Convolutional Layers and Batch Normalization

The model contains three convolutional layers, each followed by batch normalization and max-pooling:

- **Convolutional Layers**: Convolutions extract spatial features such as edges, corners, and textures from the input images. The first convolution layer extracts 32 feature maps, followed by 64 and 128 in the second and third layers respectively.
- **Batch Normalization**: Batch normalization normalizes the activations within a mini-batch, improving the convergence speed and reducing sensitivity to initialization. This reduces internal covariate shift and regularizes the model.
- **Max Pooling**: Max pooling is applied after each convolutional layer to reduce the spatial dimensions of the feature maps by half, retaining only the most important information.

## 3.2 Fully Connected Layers and Dropout

After flattening the feature maps, the model passes the features through two fully connected layers:

- **First Fully Connected Layer**: This layer has 256 neurons and applies the ReLU activation function to introduce non-linearity.

- **Dropout**: Dropout is used to randomly deactivate neurons during training with a probability of 50%. This helps prevent overfitting by forcing the model to learn redundant representations.
- **Output Layer**: The final fully connected layer outputs 10 values, corresponding to the class probabilities for each digit (0-9). Softmax is applied to convert these values into probabilities.

### 3.3 Optimization and Learning Rate Scheduler

The Adam optimizer is used with an initial learning rate of 0.001. To further optimize the learning process, a learning rate scheduler reduces the learning rate by a factor of 10 after 5 epochs, ensuring that the model fine-tunes the weights as training progresses.

Listing 1: CNN Architecture

```
class AdvancedDocumentRecognitionCNN(nn.Module):
    def __init__(self):
        super(AdvancedDocumentRecognitionCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1, 1)
        self.bn1 = nn.BatchNorm2d(32)
        self.conv2 = nn.Conv2d(32, 64, 3, 1, 1)
        self.bn2 = nn.BatchNorm2d(64)
        self.conv3 = nn.Conv2d(64, 128, 3, 1, 1)
        self.bn3 = nn.BatchNorm2d(128)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(128 * 3 * 3, 256)
        self.fc2 = nn.Linear(256, 10)
        self.dropout = nn.Dropout(0.5)
    def forward(self, x):
        x = self.pool(F.relu(self.bn1(self.conv1(x))))
        x = self.pool(F.relu(self.bn2(self.conv2(x))))
        x = self.pool(F.relu(self.bn3(self.conv3(x))))
        x = x.view(-1, 128 * 3 * 3)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

# 4 Training and Results

## 4.1 Training Process

The model was trained for 10 epochs using a batch size of 64. During each epoch, the optimizer adjusted the weights to minimize the cross-entropy loss function, which measures the difference between the predicted probabilities and the true labels.

The following table shows the training loss for each epoch:

```
Epoch [1/10], Loss: 0.2732
Epoch [2/10], Loss: 0.1192
Epoch [3/10], Loss: 0.0953
Epoch [4/10], Loss: 0.0838
Epoch [5/10], Loss: 0.0720
Epoch [6/10], Loss: 0.0493
Epoch [7/10], Loss: 0.0446
Epoch [8/10], Loss: 0.0422
Epoch [9/10], Loss: 0.0382
Epoch [10/10], Loss: 0.0373
```

## 4.2 Test Results

Upon completion of training, the model was evaluated on the test set, achieving the following results:

- **Test Accuracy**: 98.84%
- **F1 Score**: 0.9884

These high accuracy and F1 scores reflect the model's strong generalization ability and robustness, which is attributed to the use of data augmentation, dropout, and batch normalization.

## 4.3 Training Loss Curve

Figure **??** shows the training loss curve over the 10 epochs. The loss steadily decreases, indicating that the model is learning effectively.

## 4.4 Confusion Matrix

The confusion matrix in Figure **??** illustrates the model's performance across different classes. It shows that most predictions are correct, with very few misclassifications.

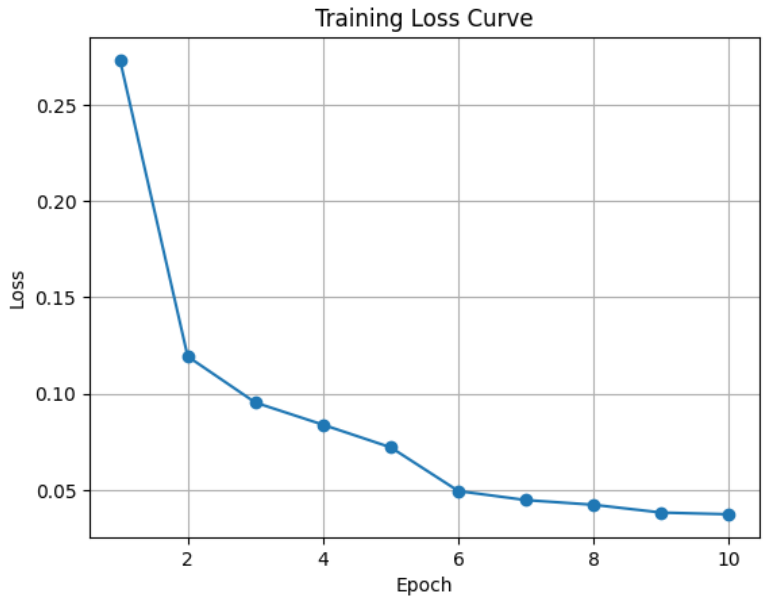


Figure 1: Training Loss Curve over 10 Epochs

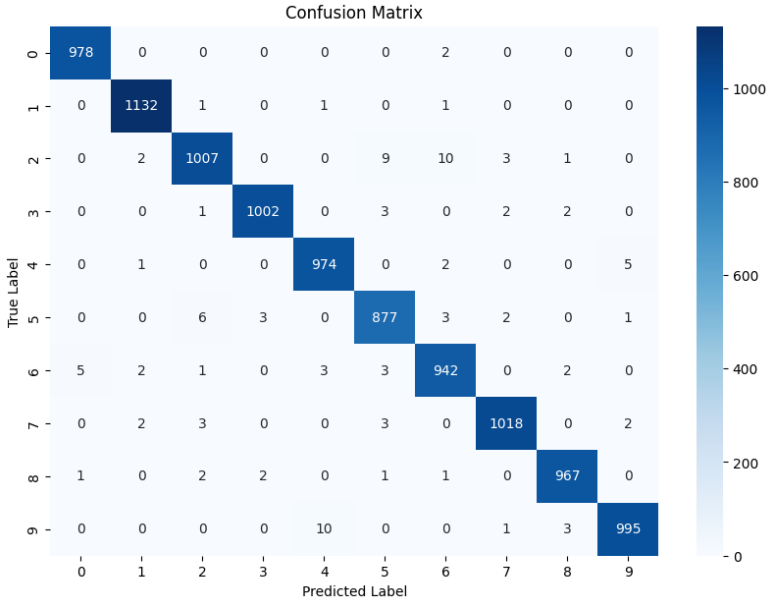


Figure 2: Confusion Matrix

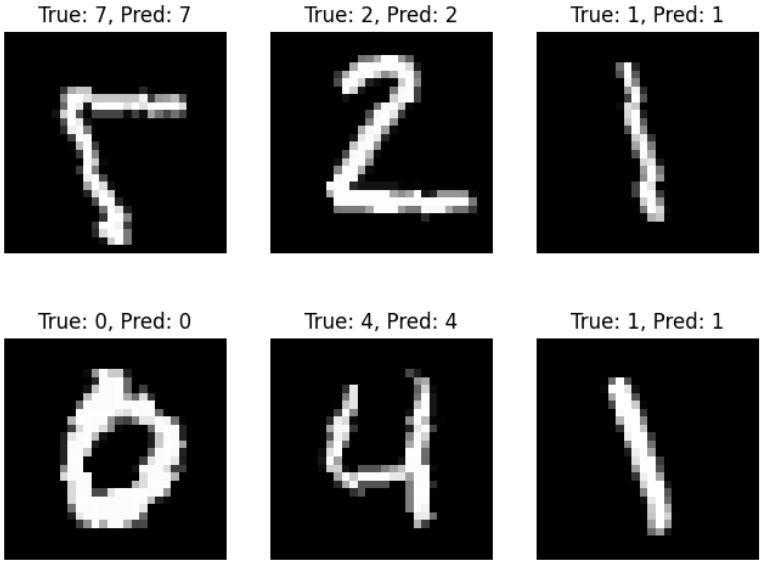


Figure 3: Sample Test Predictions

### 4.5 Sample Test Predictions

Sampletest predictions are shown in Figure **??**. The model successfully identifies digits from the test set, and these examples show the model's accuracy in practice.

## 5 Conclusion

The CNN-based approach for handwritten digit recognition demonstrated excellent performance, achieving a test accuracy of 98.84% and an F1 score of 0.9884. By using data augmentation, batch normalization, dropout, and learning rate scheduling, the model achieved robust generalization on unseen test data.

The main contributors to the success of the model are:

- **Convolutional Layers**: These extract essential features from images and detect patterns such as edges and corners.
- **Batch Normalization**: This helped stabilize and accelerate the training process.
- **Dropout**: Applied to avoid overfitting and ensure better generalization.

- **Data Augmentation**: Techniques such as random flipping and rotation augmented the dataset and improved robustness.

Overall, the project successfully built an accurate and efficient handwritten digit recognition system. Future work could explore extending this architecture to more complex datasets, such as COCO-Text, for recognizing multiple characters in more challenging real-world settings.

# 6 References


- Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.
- Goodfellow, I., Bengio, Y., Courville, A. (2016). Deep Learning. MIT Press.