Handwritten Digit Recognition Using CNN Report

Meet Agrawal

July 6, 2025

1 Introduction

The goal of this project is to train a Convolutional Neural Network (CNN) to classify handwritten digits from the MNIST dataset.

2 Data Preparation and Augmentation

Normalization is a critical preprocessing step because it ensures that all pixel values lie in a standard range, which stabilizes gradients and speeds up convergence.Following values were used in this project(standard metrics of the MNIST Dataset):

- Mean: 0.1307
- Standard Deviation: 0.3081

Data Augmentation

To prevent overfitting and improve generalization, we applied:

- **RandomRotation(10)**: Rotates digits at slight angles.
- **RandomAffine(translate=(0.1, 0.1))**: Introduces shifts in digit placement

3 CNN Architecture Design

Convolutional Layers

- Conv2d(1, 32, kernel=3, padding=1): Extracts 32 low-level features.
- Conv2d(32, 64, kernel=3, padding=1): Deeper feature extraction.

Activation Function

- ReLU is used after each layer to introduce non-linearity and prevent vanishing gradients.

Pooling Layer

- MaxPool2d(2, 2): Reduces spatial size by a factor of 2 and introduces translation invariance.

Fully Connected Layers

- Linear($64 \times 14 \times 14$, 128): Maps feature maps to dense representation.
- Linear(128, 10): Output layer with 10 neurons for 10 digit classes.

Output Activation

- LogSoftmax: Suitable for multi-class classification and used with NLLLoss.

4 Training and Evaluation

Training Configuration

Loss Function: We used `NLLLoss` (Negative Log Likelihood Loss) because our model's final layer outputs log probabilities using `log_softmax`. This loss function is suitable for multi-class classification tasks like MNIST.

Optimizer: We used the `Adam` optimizer with a learning rate of 0.001. It automatically adjusts learning rates for each parameter, helping the model learn faster and more reliably.

Training Process

Each time the model sees the full training data (one epoch), the following steps occur:

- **Forward pass:** The image is passed through the model to get a prediction.
- **Loss computation:** The model checks how different its prediction is from the actual label.
- **Backpropagation:** The model calculates how to adjust itself to reduce this error.
- **Optimizer step:** The model updates its weights slightly to improve.

5 Results and Analysis

Training and Test Metrics

- **Epoch 1**: Train Loss = 0.2549, Test Loss = 0.0433, Accuracy = 98.46%
- **Epoch 2**: Train Loss = 0.0946, Test Loss = 0.0453, Accuracy = 98.46%
- **Epoch 3**: Train Loss = 0.0751, Test Loss = 0.0308, Accuracy = 98.96%
- **Epoch 4**: Train Loss = 0.0638, Test Loss = 0.0308, Accuracy = 99.08%
- **Epoch 5**: Train Loss = 0.0575, Test Loss = 0.0274, Accuracy = 99.02%

Confusion Matrix Observations

The model occasionally misclassified:

- 9 as 4

Challenges Faced

- **Overfitting without data augmentation:** When data augmentation was removed, the model's training loss dropped sharply (e.g., 0.0142 by epoch 5), indicating that the model memorized training data. However, test loss fluctuated (e.g., 0.0457 in epoch 5), and accuracy stagnated around 98.7–98.9%, confirming overfitting.
- **Confusion between visually similar digits:** The model sometimes confused digits like 4 and 9as observed in the confusion matrix.

Proposed Improvements

- Use Dropout layers to reduce overfitting.
- Train for more epochs with learning rate decay.
- Try deeper models like ResNet or apply BatchNorm.