CNN Implementation for MNIST Digit Recognition

Kotesh Ravula (101145486)

Leela Krishna Sai Dangeti (101145083)

Madan Bandi (101142384)

1. Data exploration and preparation:

First, we begin by loading the MNIST digit recognition dataset. This data, sourced from the UCI Machine Learning Repository, is conveniently stored in CSV format. Before diving into training our model, it's essential to prepare our data. This involves a couple of key steps. Firstly, we normalize the pixel values. This ensures that all values are scaled proportionately between 0 and 1. This normalization step is crucial as it helps our model to learn more efficiently during training. Secondly, we reshape our data. The raw data is in the form of 64 features, representing an 8x8 pixel grid, along with a target column indicating the digit label. We reshape this data into 28x28 arrays to match the input shape expected by our Convolutional Neural Network (CNN) model.

2. Convolutional Neural Network Architecture:

Now, let's delve into the architecture of our CNN. It's comprised of three convolutional layers, each followed by a ReLU activation function. These convolutional layers serve to extract features from our input images. Think of them as filters that scan through the image, picking up patterns and features. Following each convolutional layer, we apply a max-pooling layer. This serves to down sample the feature maps, reducing their spatial dimensions while retaining important information. The dimensions of each layer are carefully chosen to balance complexity and effectiveness. The first convolutional layer utilizes 32 filters with a 3x3 kernel size, producing feature maps of size 26x26. Subsequent layers follow a similar pattern, with appropriate adjustments to filter count and kernel size.

3. Max Pooling:

Max pooling is a critical component of our CNN architecture. It helps in reducing the computational burden and mitigating overfitting by summarizing the most important features while discarding less relevant details. After each convolutional layer, we apply max pooling, effectively down sampling the feature maps and consolidating the extracted information.

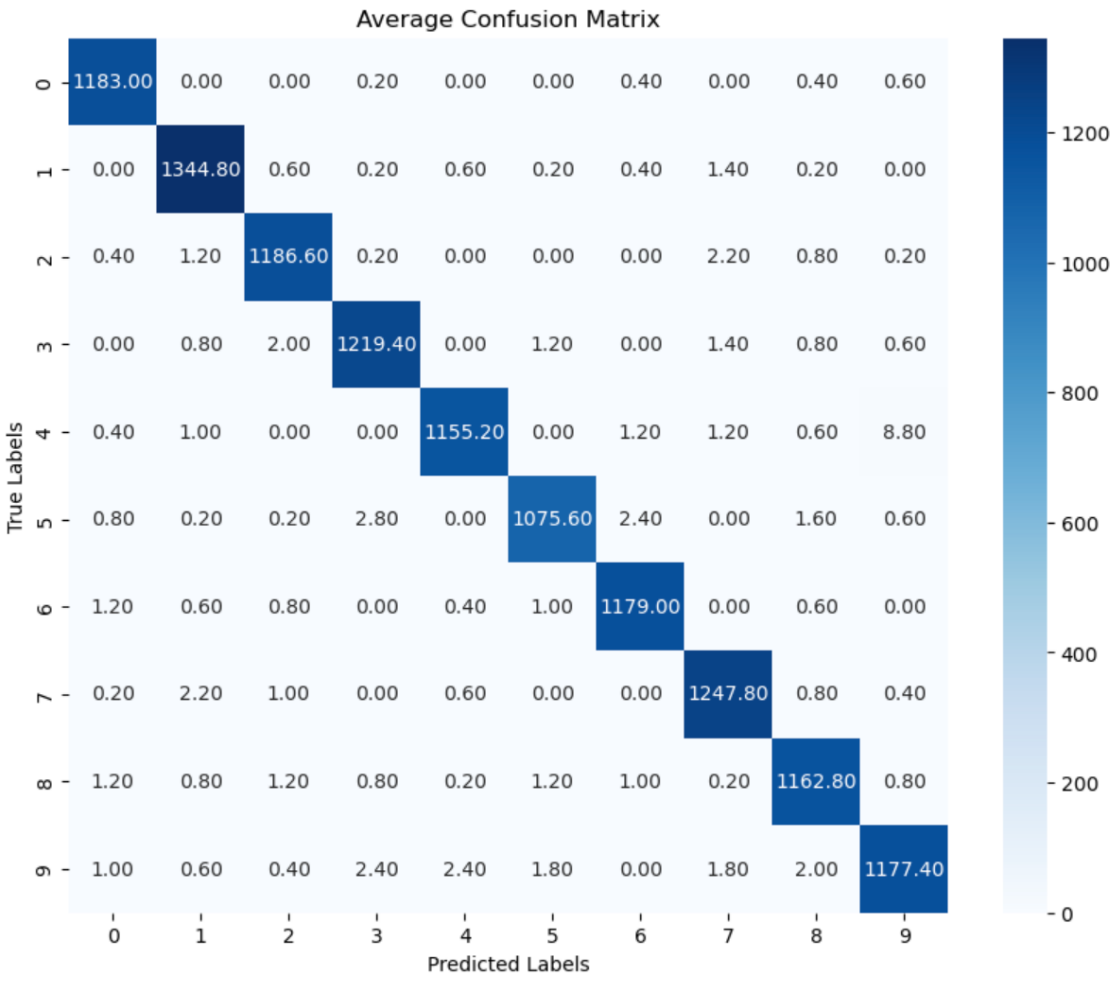
4. Fully Connected Layer and SoftMax:

Once we've extracted relevant features through our convolutional layers, we flatten the output and pass it through a fully connected layer. This layer helps in learning higher-level features from

the extracted ones. Finally, we apply a fully connected layer with SoftMax activation for classification. This provides us with probabilities for each digit class, enabling us to make accurate predictions.

5. Training and Evaluation:

With our model architecture defined, we move on to the training phase. Here, we utilize the Adam optimizer and sparse categorical cross-entropy loss function to train our model.
Throughout the training process, we monitor various metrics such as training and validation accuracy and loss to gauge the model's performance. To evaluate the model's performance, we put it to the test on a separate dataset. This allows us to assess its ability to generalize to unseen data.
Additionally, we employ K-Fold cross-validation to ensure the robustness of our model. We analyze the confusion matrix to gain insights into any misclassifications and areas for improvement.



6. Documentation and Analysis:

Each step of our CNN implementation is thoroughly documented to provide clarity and understanding. The code is meticulously commented, enhancing readability and facilitating comprehension. We conclude with a comprehensive analysis of our CNN implementation, discussing the significance of each component in the context of digit recognition. This analysis highlights the strengths of our approach while identifying potential areas for refinement.