Handwritten Digit Classification with a CNN

Matthew Snyman

August 12, 2025

1 Project Overview

This project implements a Convolutional Neural Network (CNN) from scratch to classify images of handwritten digits (0-9) using the MNIST dataset. This project involves data pre-processing, neural network design, model training and validation, evaluation, performance visualization and results interpretation.

2 Dataset

The MNIST dataset contains 70,000 grayscale images of handwritten digits (0–9), each of size 28×28 pixels. It is split into 50,000 training images, 10,000 validation images and 10,000 test images for a 72%/14%/14% training/validation/test split. The images were normalized to the range [0, 1] and converted to PyTorch tensors. Figure 1 shows some example images from the MNIST dataset.

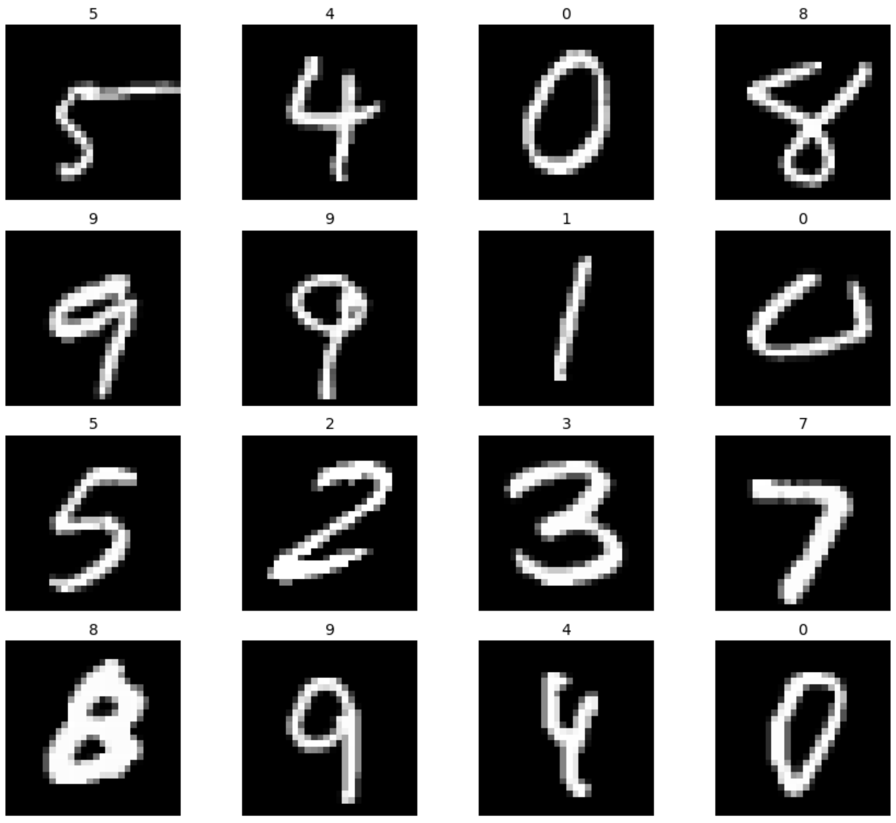

Figure 1: Images of handwritten digits from the MNIST dataset

3 Model Architecture

The model is a convolutional neural network consisting of:

- A first convolutional layer taking one input (since grayscale images have only one channel) with 32 outputs, kernel size 3×3 with padding of 1 pixel and a ReLU activation function
- A second convolutional layer taking 32 inputs (output of layer 1), 64 outputs, kernel size 3×3, with padding of 1 pixel and a ReLU activation function
- A max pooling layer to reduce compute and suppress weaker, noisier features (kernel size 3×3 with stride of 2, resulting in 14×14 pixel intermediate downscaled image)
- A first fully connected ”dense” layer taking $64 \times 14 \times 14$ inputs with 128 features, a ReLU activation function and dropout applied
- A second fully connected output layer taking 128 inputs and 10 outputs (for the 10 digit classes)

Figure 2 shows a schematic representation of the neural network.

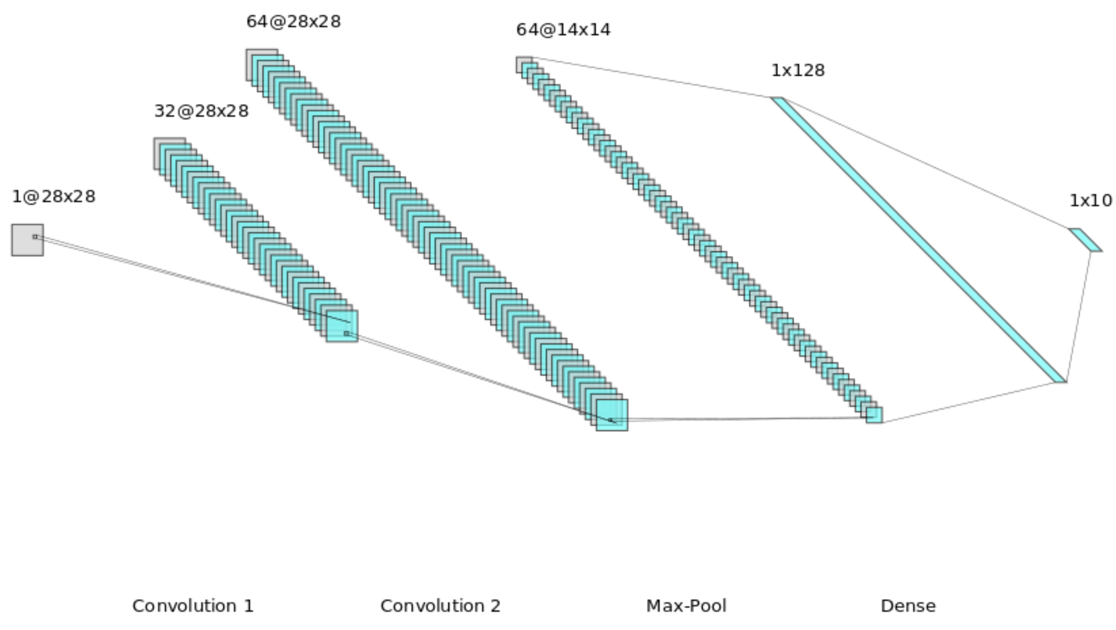

Figure 2: Schematic of the CNN

4 Training Process

The model was trained using the cross-entropy loss function and the Adam optimizer. Dropout was used on the first fully connected layer to reduce overfitting. Details about the training process are given by Table 1. Training was carried out on an Intel Core i7 CPU - total training time $\approx$ 10 minutes.

Table 1: Network training parameters

Parameter	Value
#Epochs	10
Learning rate	0.001
Dropout rate	0.25
Weight decay	0
Batch size	64

5 Results

Figure 3 shows the evolution of the training and validation losses and accuracies with each epoch during training. After training, the model was tested using the testing dataset. Table 2 summarizes the accuracy values of the model.

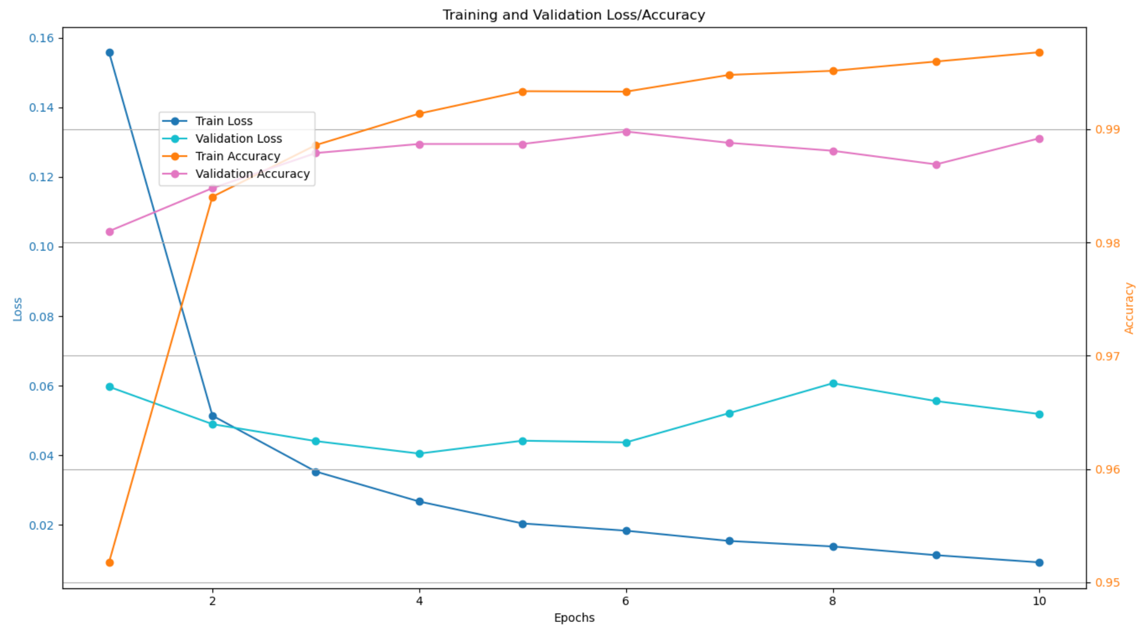

Figure 3: Training and validation accuracies and losses

Table 2: Train, validation and test accuracies

Metric	Value
Train accuracy	99.7%
Validation accuracy	98.9%
Test accuracy	99.1%

Figure 4 shows a subset of images that the network incorrectly predicted. Note that while for most of these images, a human would most likely guess correctly, others are much more ambiguous and would likely be difficult to guess correctly, even for a human (e.g., the top middle digit in Figure 4).

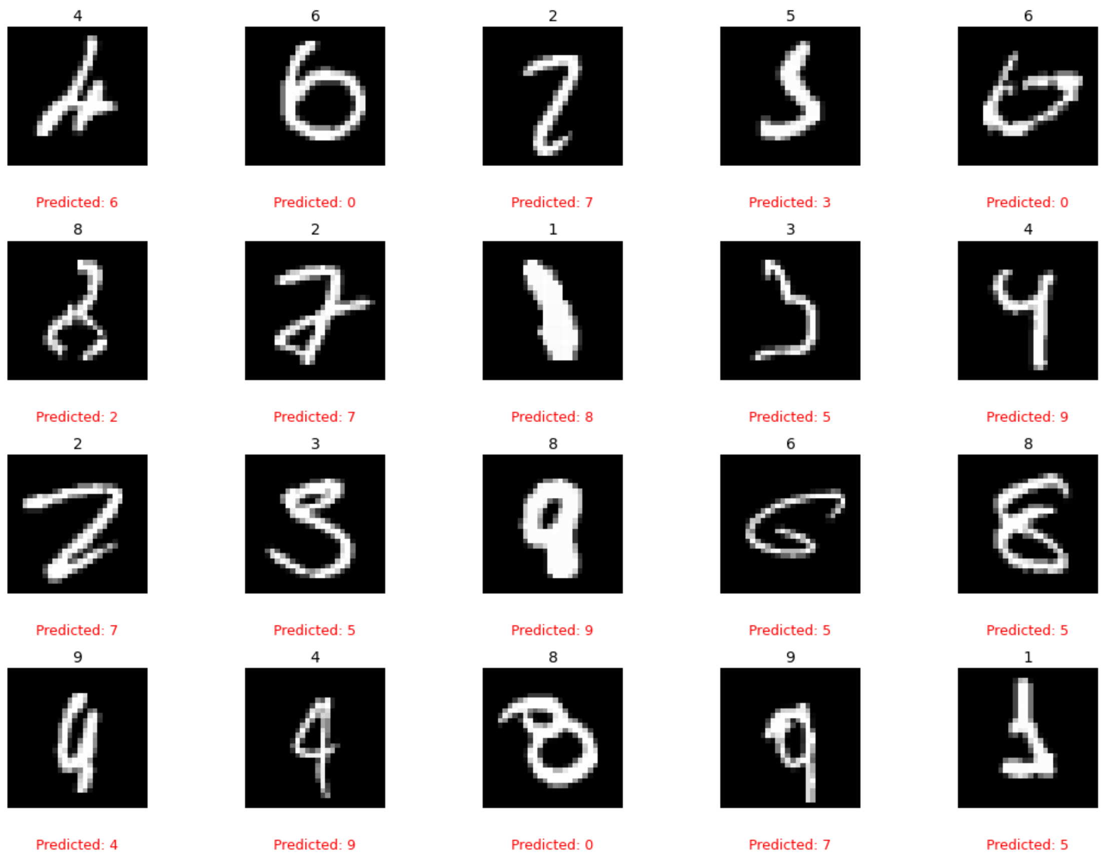

Figure 4: Incorrect predictions by the trained model

6 Conclusions & Next Steps

This simple CNN was quick to train and achieved strong generalization with 99.1% test accuracy. There was no sign of overfitting according to the evolution of the training and validation accuracies during training. Judging by some of the incorrect predictions, this model would likely perform slightly worse than a human at the same test. However, this is expected for such a small model, and it has still shown admirable results given its simplicity.

Future work includes:

- Training on the EMNIST dataset (letters and digits).
- Experimenting with deeper architectures.
- Deploying the model as a simple web application.