PROJECT ABSTRACT

Handwritten Digit Recognition with Custom CNN (MNIST)

A custom convolutional neural network trained on MNIST for handwritten digit recognition, with a draw-and-recognize web demo, training notebook and complete documentation.

Introduction / Background

Handwritten digit recognition is the entry point to deep learning: the MNIST dataset of 70,000 labeled 28x28 digit images has benchmarked every generation of neural networks since the 1990s. Convolutional neural networks dominate this task because convolution exploits the spatial structure of strokes, pooling builds translation tolerance, and stacked layers compose edges into digit shapes.

Students typically meet this project through tutorials that hand them a finished model. They can run the code but cannot explain why each layer exists, which collapses in the viva. A from-scratch custom CNN, with every architectural choice documented, turns the canonical tutorial into a genuine engineering exercise.

Problem Statement

Tutorial-following produces graduates who can execute a notebook but cannot defend an architecture: why 32 filters and not 128, why dropout after pooling, what the confusion matrix reveals. Without a designed-from-scratch model and a full evaluation, the student owns none of the decisions an examiner will probe.

Objectives

Design a custom CNN for 10-class digit classification on MNIST. Train it reproducibly with documented hyperparameters and augmentation. Evaluate it with accuracy, confusion matrix and per-class metrics. Deploy it behind a draw-and-recognize web demo. Document every decision for the viva.

Proposed Methodology

MNIST images are normalized to [0,1] and split 60,000/10,000 with a 10% validation split. Light augmentation (rotations, shifts, zoom) improves robustness to sloppy handwriting. The network stacks Conv2D(32)-MaxPool-Conv2D(64)-MaxPool-Dropout-Flatten-Dense(128)-Dropout-Soft max(10), compiled with Adam and categorical cross-entropy and trained for 25 epochs with checkpointing on validation accuracy. The test set is evaluated exactly once.

System Architecture / Working

The pipeline has four stages. Capture: the demo canvas records strokes at 280x280. Preprocess: the drawing is downscaled to 28x28, centered and normalized identically to training data. Inference: the saved .h5 model runs a forward pass in about 8 ms on CPU. Presentation: the app renders the predicted digit with the full 10-class probability distribution. Training curves and the confusion matrix feed the report.

Hardware / Software Requirements

Hardware: any laptop with 8 GB RAM; CPU training completes in roughly 15-25 minutes, GPU optional. Software: Python 3, TensorFlow/Keras, NumPy, Matplotlib, scikit-learn, Jupyter; the demo needs only a modern browser.

Expected Outcomes

A trained CNN reaching the 99.2% test-accuracy design target with a documented confusion matrix (expected confusions: 4/9, 3/8, 5/6), a working draw-and-recognize demo, and a report the student can defend end to end.

Applications

Postal OCR, bank-cheque reading, form digitization, and as the teaching foundation for any image-classification work the student attempts later.

Future Scope

Multi-digit string segmentation, cursive and rotated handwriting, and extending the demo to camera-captured digits with perspective correction.