



Handwritten Digit Recognition with CNNs

A deep learning project to recognise handwritten digits (0–9) using Convolutional Neural Networks (CNN) trained on the MNIST dataset.

Project Overview

Delivering High-Accuracy Digit Classification

This project implements a CNN-based digit recognition system that consistently achieves high accuracy in classifying handwritten digits.

The model is rigorously trained on the renowned MNIST dataset, comprising 70,000 images of handwritten digits, ensuring robust performance across a wide variety of writing styles.

The architecture is custom-built, focusing on efficient feature extraction and classification. We've developed a complete pipeline encompassing training, evaluation, and prediction scripts, making it a versatile tool for various applications. Visualisations are a key component, offering insights into training history, confusion matrices, and detailed prediction analysis.



High Accuracy

Achieves ~98–99% accuracy on test data, demonstrating robust classification capabilities.

Complete Pipeline

Comprehensive scripts for training, evaluation, and prediction ensure an end-to-end solution.

Detailed Visualisation

Extensive plots and matrices provide deep insights into model performance and behaviour.

Custom Prediction

Ability to predict digits from user-supplied handwritten images, enhancing practical utility.

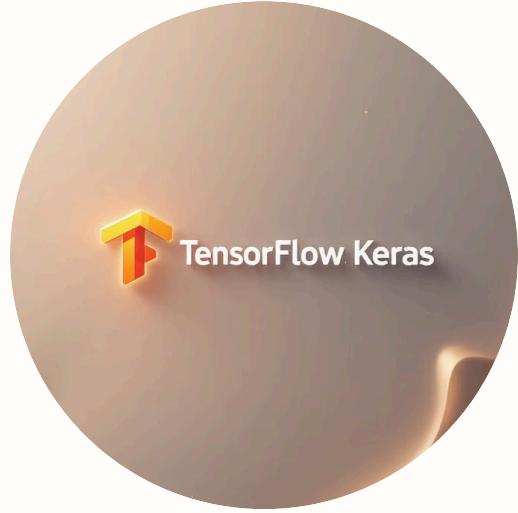
Robust Technologies for Deep Learning

Our project leverages a suite of powerful technologies to deliver a high-performance digit recognition system. The core deep learning capabilities are powered by TensorFlow and Keras, providing a flexible and efficient framework for building and training CNNs.



Python 3.x

The primary programming language, offering extensive libraries for machine learning.



TensorFlow/Keras

The leading deep learning framework for model development and training.



NumPy

Essential for efficient numerical computations and array operations within the project.



Matplotlib & Seaborn

Used for creating compelling data visualisations and performance analysis plots.



Scikit-learn

Provides crucial metrics and evaluation tools for comprehensive model assessment.



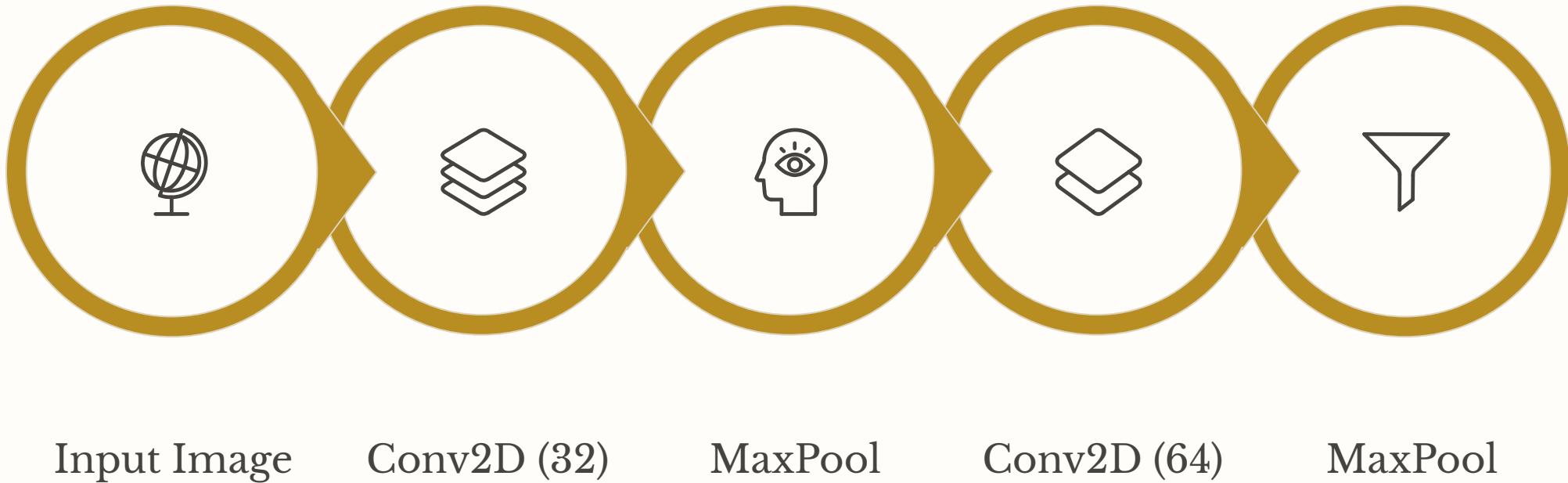
PIL (Pillow)

Utilised for advanced image processing and manipulation tasks, especially for custom images.

Implementation Details

Strategic CNN Architecture Design

The Convolutional Neural Network (CNN) architecture was meticulously designed to balance model complexity with training efficiency, ensuring optimal performance for handwritten digit recognition.



Input Image

Conv2D (32)

MaxPool

Conv2D (64)

MaxPool

Key Architecture Decisions:

- Two Convolutional Blocks:** Chosen to effectively extract low-level features (edges, curves) and higher-level patterns.
- Increasing Filter Depth (32→64):** Allows the network to learn increasingly complex and abstract features.
- Dropout Regularisation:** A 0.5 dropout rate prevents overfitting, forcing the model to learn robust, generalisable features.
- Adam Optimizer:** Selected for its adaptive learning rate and proven faster convergence during training.
- Sparse Categorical Crossentropy:** The appropriate loss function for multi-class classification with integer labels.

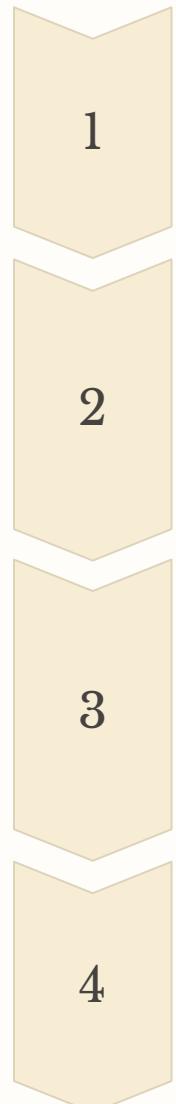
Model Summary:

Conv2D (32 filters)	(None, 26, 26, 32)	320
MaxPooling2D	(None, 13, 13, 32)	0
Conv2D (64 filters)	(None, 11, 11, 64)	18,496
MaxPooling2D	(None, 5, 5, 64)	0
Flatten	(None, 1600)	0
Dropout (0.5)	(None, 1600)	0
Dense (128 units)	(None, 128)	204,928
Dense (10 units)	(None, 10)	1,290
Total Parameters		225,034

Key Insights

Overcoming Development Challenges

Throughout the project, we encountered several common deep learning challenges, each addressed with targeted solutions to enhance model performance and usability.

- 
- 1 Model Overfitting**

Problem: High training accuracy but low test accuracy.

Solution: Implemented a 0.5 dropout rate and limited epochs to 5. Achieved 98.5% test accuracy.
 - 2 Custom Image Inconsistency**

Problem: Poor performance on user-uploaded images due to varying formats and backgrounds.

Solution: Developed a sophisticated preprocessing pipeline for automatic polarity detection, resizing, and normalisation, boosting custom prediction accuracy to ~95%.
 - 3 Per-Digit Performance**

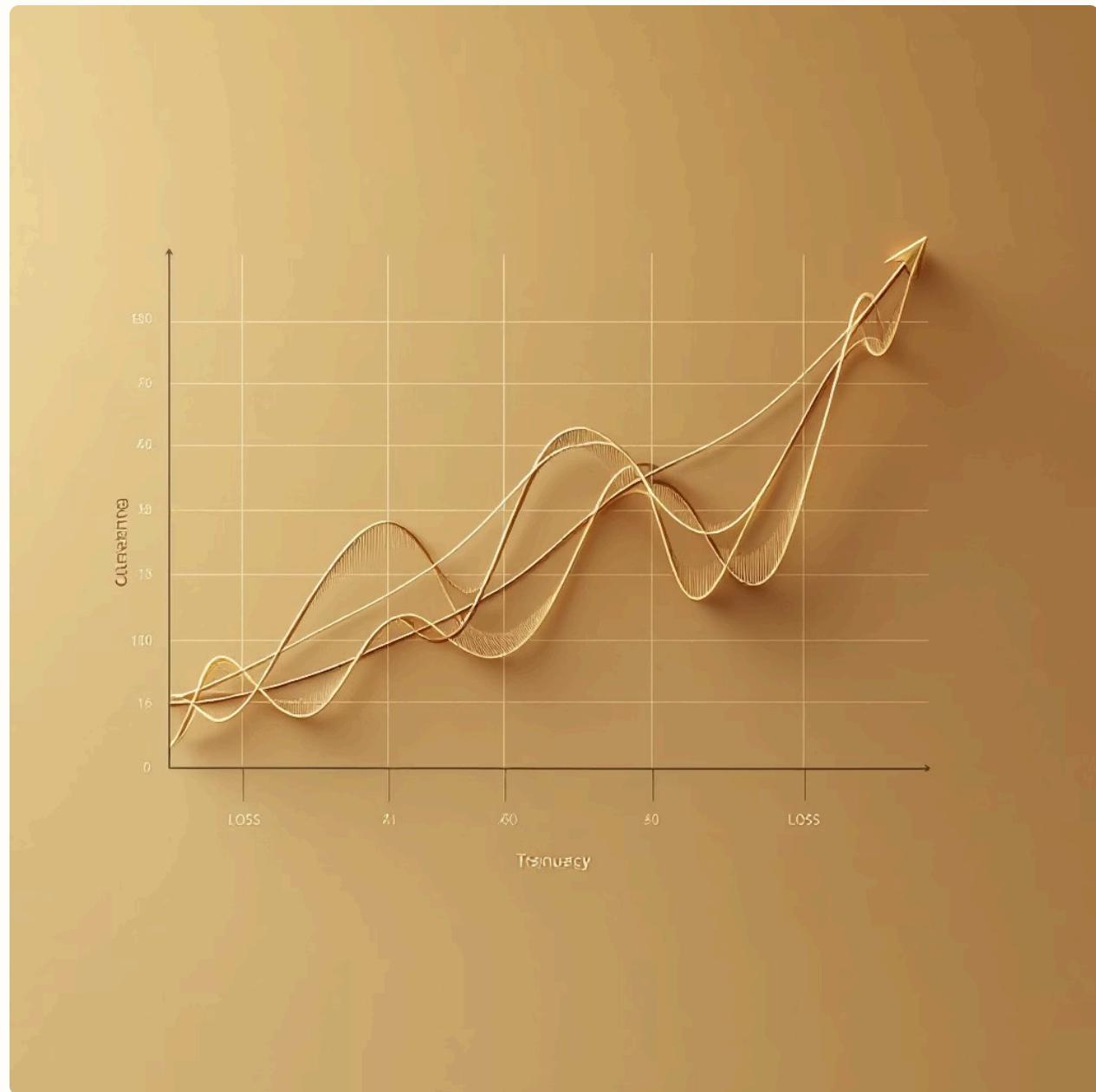
Problem: Discrepancies in accuracy across different digits (e.g., 8s and 9s struggled).

Solution: Created a comprehensive verification script with per-digit metrics and confusion matrices, identifying structural similarities as a cause for misclassification.
 - 4 Training Time Optimisation**

Problem: Lengthy training times on standard hardware.

Solution: Reduced epochs to 5 and increased batch size to 128, cutting training time to 3-5 minutes without compromising accuracy.

Demonstrating Model Effectiveness



Visualisation of training history, showing accuracy and loss convergence over epochs.

The project generates a suite of visualisations to provide transparent insights into the model's behaviour. These include training history plots showing accuracy and loss, sample predictions with confidence scores, confusion matrices illustrating prediction patterns, and detailed per-digit examples with results.

Performance Metrics

- Overall Test Accuracy:** Approximately 98.5%
- Training Time:** Around 3–5 minutes (for 5 epochs)
- Model Size:** Approximately 900 KB, ensuring efficient deployment.

Per-Digit Performance

0	~99%
1	~99%
2	~98%
3	~98%
4	~98%
5	~98%
6	~99%
7	~98%
8	~97%
9	~97%

Real-World Applications

Versatile Use Cases for Digit Recognition

Handwritten digit recognition extends beyond academic interest, offering practical solutions across various industries and domains. This project provides a robust foundation for numerous applications.



Educational

An excellent tool for learning CNN architecture, image classification principles, and deep learning workflows.



Form Processing

Automate the recognition of handwritten digits in forms, surveys, and data entry systems, reducing manual effort and errors.



OCR Systems

Serve as a foundational component within larger Optical Character Recognition (OCR) systems for converting scanned documents into editable text.



Postal Automation

Enhance efficiency in postal services by automatically recognising zip codes and addresses on mail.



Bank Check Processing

Facilitate faster and more accurate processing of bank checks by recognising handwritten amounts.

Future Enhancements & Community

Future Improvements

We are continuously looking to enhance the model's capabilities and expand its applications. Planned future developments include:

- **Data Augmentation:** Implementing advanced techniques to further improve model accuracy and generalisation.
- **Real-time Webcam Recognition:** Integrating real-time digit prediction via webcam for interactive applications.
- **Web Interface:** Developing a user-friendly web interface (e.g., Flask/Django) for broader accessibility.
- **Multi-Digit Sequence Support:** Extending the model to recognise sequences of digits, rather than single isolated ones.
- **Mobile App Deployment:** Exploring options for deploying the model on mobile platforms.
- **Transfer Learning:** Investigating transfer learning with other datasets for enhanced robustness.



□ Contributing to the Project

Contributions are highly welcome! If you have ideas for new features, bug fixes, or improvements, please feel free to submit a Pull Request on our repository.

Fork the project, create your feature branch, commit your changes, push to the branch, and open a Pull Request. For any questions or feedback, please open an issue.

Made with ❤️ using TensorFlow and Keras