

**A2**

# Handwritten Digit Recognition System

# Introduction:

In the fields of computer vision and machine learning, handwritten digit recognition is a classic problem that is especially important to applications like handwritten form automation, bank check processing, and postal mail sorting. Correctly identifying a digit (0–9) from an image with handwritten numbers is the aim. In order to classify handwritten digits from the MNIST dataset, a convolutional neural network (CNN) model based on the LeNet-5 architecture was developed using Python, TensorFlow, and OpenCV. In addition to using core TensorFlow functions for model training, OpenCV was employed for image preprocessing to standardize image formats, normalize pixel values, and handle background inconsistencies.The main goal is training a model that can correctly classify images of digits (0 through 9) and assessing its performance with a validation set.

# Dataset:

One popular benchmark for classifying handwritten digits is the MNIST dataset. Each of its 10,000 test and 60,000 training images is a 28x28 greyscale picture of a number between 0 and 9. Each digit class (i.e., 0 to 9) is used to name the folders into which the dataset is organised. There are several image files in each folder that correspond to that numeral.The dataset was locally stored for this project, and it was read, preprocessed, and appropriately labelled.

# Data Preprocessing:

Before feeding the images into the neural network, the following preprocessing steps were applied:

1. **Grayscale Conversion**: Each image was read in grayscale format using OpenCV to reduce complexity and processing time.
2. **Resizing**: Images were resized to 28x28 pixels to match the input shape required by the LeNet-5 architecture.
3. **Normalization**: The pixel values were normalized to the range [0,1] by dividing by 255.0.
4. **Label Encoding**: Labels were one-hot encoded using to\_categorical() from TensorFlow.
5. **Reshaping**: Images were reshaped to (28, 28, 1) to add a channel dimension, which is required by Keras CNN layers.
6. **Shuffling**: The training data was shuffled to improve model generalization and avoid learning order bias.

# Model Architecture:

This project's model architecture is based on the famous LeNet-5, one of the first CNNs created especially for digit recognition. The following layers are part of the architecture:

* **Conv2D (6 filters, 5x5 kernel, ReLU)**: Extracts local features using six convolution filters.
* **AveragePooling2D (2x2)**: Reduces the spatial dimension while retaining important features.
* **Conv2D (16 filters, 5x5 kernel, ReLU)**: Further extracts higher-level features.
* **AveragePooling2D (2x2)**: Further downsampling of feature maps.
* **Flatten**: Converts the 2D feature maps into a 1D feature vector.
* **Dense (120 units, ReLU)**: Fully connected layer with 120 neurons.
* **Dense (84 units, ReLU)**: Another fully connected layer with 84 neurons.
* **Dense (10 units, Softmax)**: Output layer with 10 units, one for each digit class, using softmax activation to output probability distributions.

The model was compiled using the **Adam optimizer** with categorical\_crossentropy as the loss function, which is suitable for multi-class classification problems.

# Training:

A batch size of 128 was used to train the model across 30 epochs using the training dataset. Validation on the test dataset was part of the training procedure to track generalization performance. The model.fit() function, which manages training and validation internally, was used to do the training.  
  
Accuracy and loss measures were tracked during the training process. Before training, the dataset was shuffled to make sure batches weren't biased towards any particular class.

# Evaluation:

The model.evaluate() function was used to assess the model on the test set after training. The model's performance on unseen data was clearly demonstrated by printing the final accuracy and loss on the test data.

Example output:

A screenshot of a test

Description automatically generated

This shows that the model performed well on this kind of task, with a test accuracy of about 98.65%.

# Prediction and Visualization:

The model.predict() function was utilised to provide predictions for specific test images. By determining the index of the maximum value in the prediction vector that is, the class with the highest probability the predicted label for every image was determined.  
  
The input image and its predicted label were shown together in matplotlib to visualise our results. The model's performance can be visually verified and debugged with the help of this step.

# Model Saving:

The trained model was saved to disk using the model.save() function  in order to preserve it for later usage . The model was saved as "handwritten\_digit\_model.h5" in HDF5 format. This avoids the requirement for retraining and makes loading and inference simple in the future.

# Custom Image Testing:

A handwritten digit in a custom image was utilized to further test the model's robustness. After loading and displaying the image to make sure that the model understands not just memorizing , a custom function was used to preprocess it. The following were handled by this function:  
A screenshot of a computer

Description automatically generated  
1- examining the greyscale image.  
2- Resize the image to 28 x 28 pixels.  
3- If the background was white, colours would be inverted (using a basic brightness threshold).  
4- Pixel values are normalised.  
5- transforming the picture into the input shape that is needed.

After that, the model was used to predict the preprocessed image, and the result was shown in the console as well as through a matplotlib visualization. The resulting model offers a foundation for advanced handwritten character recognition systems and showed good generalization on unseen digit samples.

# Issues and Solutions:

* Background Inconsistencies: In contrast to the training data, several test images featured black digits on white backgrounds. Inverting such photos during preprocessing solved this.
* Monitoring Validation: To keep an eye on model performance and identify overfitting, a validation set was added.

# Conclusion:

In conclusion, This study used a convolutional neural network that was modelled by LeNet-5 to successfully create a handwritten digit recognition system. The method was tested using customised images and demonstrated good accuracy on the MNIST dataset. Data preparation, model construction, training, evaluation, prediction, and future-use storage were all covered by the pipeline. This study provides a fundamental illustration of how deep learning may be successfully used to solve actual image classification issues, It effectively illustrates how to use a CNN and a custom dataset to create and train a digit recognition system. The accuracy and robustness of the model were greatly increased by the application of validation and appropriate preprocessing.