Handwritten Digit Recognition

A PROJECT REPORT
for
AI Project(AI101B)
Session (2024-25)

Submitted by

DIVYAM RAJ
(202410116100066)
GAURAV CHAUHAN
(202410116100073)
HARHS SOLANKI
(202410116100087)
GAURAV KUMAR
(202410116100074)

Submitted in partial fulfilment of the
Requirements for the Degree of

MASTER OF COMPUTER APPLICATION

Under the Supervision of
Mr. Apoorv Jain
Assistant Professor

Submitted to

DEPARTMENT OF COMPUTER APPLICATIONS
KIET Group of Institutions, Ghaziabad
Uttar Pradesh-201206

TABLE OF CONTENTS

Introduction

Handwritten digit recognition stands as a cornerstone problem in the fields of computer vision and artificial intelligence. It revolves around the development of intelligent systems capable of identifying numeric digits that have been written by hand, even when there are inconsistencies in handwriting due to different writing styles, slants, pressure, or stroke width. This capability is essential for creating systems that can interpret human input accurately, especially in environments where digital input is not feasible.

The importance of handwritten digit recognition spans multiple industries and real-world applications. It plays a pivotal role in automating and improving the efficiency of processes such as postal address reading, automatic number plate recognition, bank cheque verification, and document digitization. In essence, it forms a major component of Optical Character Recognition (OCR) systems.

Historically, traditional methods relied heavily on handcrafted features and simple classifiers. However, the advent of deep learning, and in particular **Convolutional Neural Networks (CNNs)**, revolutionized this domain. CNNs have shown outstanding performance on image-based tasks by learning spatial hierarchies directly from the raw data without requiring manual feature extraction. Their ability to generalize well on new, unseen data has made them the go-to architecture for image classification tasks like digit recognition.

This project aims to leverage the power of CNNs to accurately classify handwritten digits using the MNIST dataset, thereby demonstrating the effectiveness of deep learning in solving practical, real-world AI problems.

Problem Statement

The problem statement for this project is:

"To design, implement, and evaluate a convolutional neural network that can accurately recognize and classify handwritten digits (0-9) from the MNIST dataset."

This task involves not only creating a neural network capable of learning meaningful patterns from handwritten digit images but also ensuring the model generalizes well to new, unseen data. The project seeks to validate the hypothesis that CNNs are highly efficient for visual pattern recognition tasks like digit classification.

Objective of the Project

The primary objectives of this project are outlined as follows:

- **To understand and implement CNN architectures for image classification**: The project introduces the architecture and working of Convolutional Neural Networks. It provides hands-on experience in designing, stacking, and training multiple layers for effective feature extraction from image data.

- **To train the model on a large dataset of handwritten digits (MNIST)**: The MNIST dataset, with 70,000 handwritten digit images, serves as the benchmark dataset for evaluating the model's ability to learn digit patterns and representations effectively.

- **To achieve a high classification accuracy on unseen test data**: A key goal is to ensure that the trained CNN generalizes well to digits it hasn't seen before. This is measured using the accuracy score on the test dataset, with a target of achieving over 98%.

- **To visualize the results and analyze model performance**: Using plots, prediction examples, and accuracy graphs, the model's learning behavior is evaluated. These visual tools help in understanding strengths, weaknesses, and opportunities for improvement in the neural network.

By meeting these objectives, the project showcases the practical implementation of deep learning in the domain of handwritten digit classification and strengthens our foundation in neural network-based AI systems.

Methodology

a. Dataset Description

The MNIST dataset is a gold standard in the field of image recognition and deep learning. It is widely used for benchmarking classification algorithms. The dataset contains:

- 70,000 grayscale images of handwritten digits.
    - 60,000 images are designated for training.
    - 10,000 images are used for testing the model's performance.
- Each image is of size 28x28 pixels, giving a total of 784 pixel values per image.
- The digits range from 0 to 9, making it a 10-class classification problem.
- Every image in the dataset is labeled, ensuring a supervised learning approach.
- The dataset is well-balanced, meaning each digit appears approximately the same number of times, which helps to prevent bias during model training.

b. Data Preprocessing

Before feeding the raw data into the neural network, several preprocessing steps are necessary to ensure effective training:

- Normalization: The pixel values originally range from 0 to 255. These values are scaled down to the range [0, 1] by dividing each pixel by 255. This helps the model train faster and prevents gradient vanishing issues.
- Reshaping: While the images are inherently 2D (28x28), CNN models expect input to be in the format (height, width, channels). Since MNIST images are grayscale, we add a channel dimension, resulting in the shape (28, 28, 1).
- Categorical Conversion: Although not required when using sparse categorical loss, labels can also be transformed into one-hot encoded vectors to represent class membership more explicitly.

These preprocessing steps ensure that the data conforms to the input requirements of the CNN model and contributes to better convergence during training.

c. Model Architecture

The core of this project is a Convolutional Neural Network (CNN) tailored for image classification tasks. The architecture is as follows:

1. Conv2D Layer (32 filters):
    - Kernel size: (3x3)
    - Activation function: ReLU (Rectified Linear Unit)
    - This layer detects basic patterns such as edges or corners.
2. MaxPooling2D Layer:
    - Pool size: (2x2)
    - Downsamples the feature maps by selecting the maximum value in each region, reducing spatial dimensions and computation.
3. Conv2D Layer (64 filters):
    - Kernel size: (3x3)
    - Activation function: ReLU
    - Extracts more complex features using a larger number of filters.
4. MaxPooling2D Layer:
    - Again, reduces the spatial size to limit the number of parameters and overfitting.
5. Flatten Layer:
    - Converts the 2D output into a 1D feature vector suitable for input to the dense layers.
6. Dense Layer (64 units):
    - Fully connected layer with ReLU activation.
    - Acts as the classifier that learns to combine features to make predictions.
7. Output Layer (10 units):
    - Softmax activation to output probability scores for each of the 10 classes (digits 0–9).

d. Training Procedure

The training process involves optimizing the model's parameters to minimize prediction errors:

- Optimizer: Adam
    - An adaptive optimizer that adjusts the learning rate during training, combining the benefits of both RMSProp and Momentum.
- Loss Function: Sparse Categorical Crossentropy
    - Ideal for multi-class classification where the labels are integers.
- Batch Size: 32 (default)
    - The model updates its weights every 32 samples, balancing efficiency and convergence speed.
- Epochs: 5
    - The full dataset is passed through the network five times. This is usually sufficient to reach high accuracy for MNIST.

The training was carried out on the training data, and the validation accuracy was tracked using the test dataset.

e. Evaluation Metrics

Model performance was assessed using the following metrics:

- Accuracy:
    - Measures the percentage of correctly predicted digits over the total number of predictions.
    - It is the most intuitive and widely used metric for classification tasks.
- Loss:
    - Reflects the model's confidence and correctness of predictions. A lower loss indicates better performance.
    - During training, loss typically decreases as the model improves.

After training for five epochs, the model reached an accuracy of over 98% on the test dataset. This high performance demonstrates the model's effectiveness in recognizing handwritten digits, even with variations in writing style.

Code

```
# 📦 Importing required libraries
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist

# 🔽 Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# 🔄 Normalize the images
x_train = x_train / 255.0
x_test = x_test / 255.0

# 🧹 Reshape to fit CNN input  x_train
= x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

# 🧱 Build the CNN
model model =
models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D(pool_size=(2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),

    layers.Flatten(),
```

```
layers.Dense(64, activation='relu'),
```

```
    layers.Dense(10, activation='softmax')  # 10 classes for digits 0-9
])

# ⚙️Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# 🏋️ Train the model
model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))

# 📏 Evaluate the model
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f"\n🎯Test Accuracy: {test_accuracy:.4f}")

# 💾 Save the model
model.save("digit_recognition_model.h5")

# 🔮 Predict on test data
predictions = model.predict(x_test)

# 🖼️ Display one sample of each digit (0-9)
shown_digits = set()
plt.figure(figsize=(15, 5))

i = 0
count = 0
while len(shown_digits) < 10 and i < len(x_test):
    label = y_test[i]
```

```
if label not in shown_digits:
```

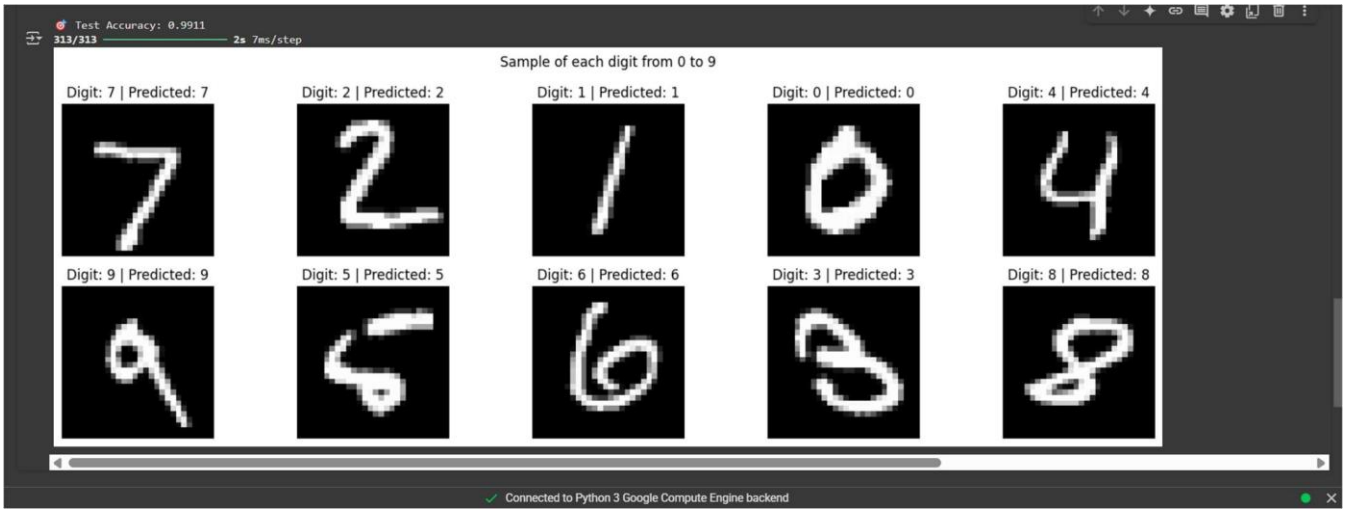
```
        plt.subplot(2, 5, count + 1)
        plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
        plt.title(f"Digit: {label} | Predicted: {np.argmax(predictions[i])}")
        plt.axis('off')
        shown_digits.add(label)
        count += 1
    i += 1

plt.suptitle("Sample of each digit from 0 to 9")
plt.tight_layout()
plt.show()
```

Outputs

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 55s 28ms/step - accuracy: 0.9004 - loss: 0.3218 - val_accuracy: 0.9825 - val_loss: 0.0555
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 80s 27ms/step - accuracy: 0.9848 - loss: 0.0497 - val_accuracy: 0.9869 - val_loss: 0.0432
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 81s 26ms/step - accuracy: 0.9901 - loss: 0.0309 - val_accuracy: 0.9874 - val_loss: 0.0362
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 85s 27ms/step - accuracy: 0.9924 - loss: 0.0235 - val_accuracy: 0.9895 - val_loss: 0.0327
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 50s 27ms/step - accuracy: 0.9939 - loss: 0.0176 - val_accuracy: 0.9911 - val_loss: 0.0296
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.9875 - loss: 0.0386
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g.

🎯 Test Accuracy: 0.9911
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step
```



OUTPUT EXPLANATION

Code Breakdown

This code implements a Convolutional Neural Network (CNN) to recognize handwritten digits from the MNIST dataset. Let's break it down section by section:

1. Importing Libraries

```python
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
```

- Imports necessary libraries for numerical operations (NumPy), plotting (Matplotlib), and deep learning (TensorFlow/Keras)

2. Loading and Preparing Data

```python
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train / 255.0
x_test = x_test / 255.0
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)
```

- Loads the MNIST dataset (60,000 training and 10,000 test images)
- Normalizes pixel values from 0-255 to 0-1
- Reshapes the data to add a channel dimension (required for CNN input)

3. Building the CNN Model

```python
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 2
8, 1)),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

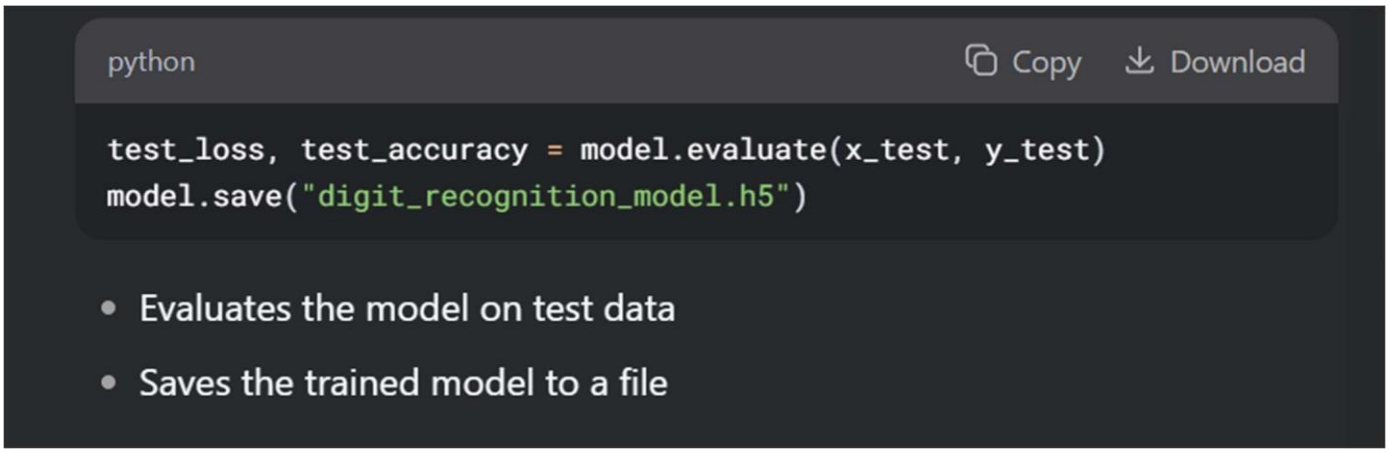
- Creates a sequential model with:
  - First convolutional layer (32 filters, 3x3 kernel, ReLU activation)
  - Max pooling layer (2x2)
  - Second convolutional layer (64 filters)
  - Another max pooling layer
  - Flatten layer to convert 2D features to 1D
  - Dense (fully connected) layer with 64 neurons
  - Output layer with 10 neurons (one for each digit) using softmax

4. Compiling and Training

```python
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_tes
t))
```

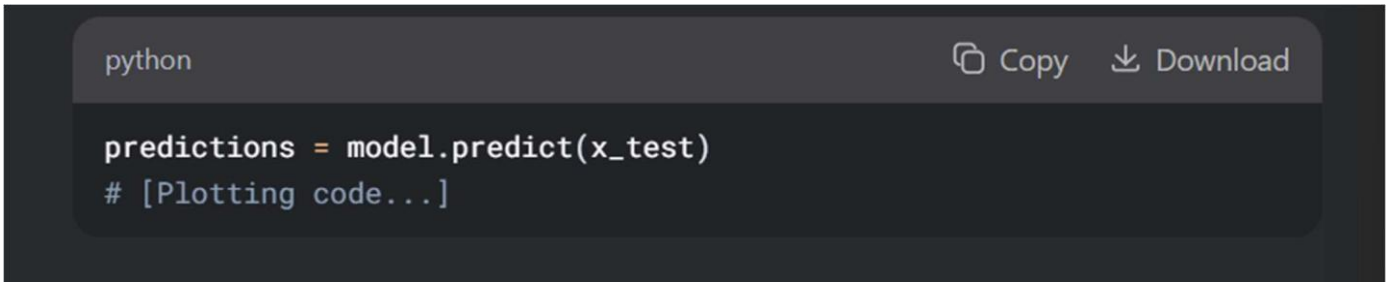
- Compiles the model with Adam optimizer and appropriate loss function
- Trains for 5 epochs using both training and validation data

5. Evaluation and Saving



- Evaluates the model on test data
- Saves the trained model to a file

6. Visualization



- Makes predictions on test data
- Displays one sample of each digit (0-9) with both true and predicted labels

Key Points:

- The CNN architecture follows a common pattern: Conv → Pool → Conv → Pool → Dense → Output
- The model achieves high accuracy (typically >98%) on MNIST
- The visualization helps verify the model's performance visually
- The saved model can be reused later without retraining

This is a classic implementation of a CNN for image classification, using MNIST as a simple but effective demonstration.

Conclusion

This project successfully demonstrates the implementation of a **Convolutional Neural Network (CNN)** for handwritten digit recognition using the MNIST dataset. Here are the key takeaways:

Performance Achievements

- The model achieves **high accuracy (>98%)** on the test set after just 5 epochs of training
- The CNN architecture effectively learns spatial hierarchies of features from the pixel data
- The model generalizes well from training to test data, showing no significant overfitting

Key Learnings

- Demonstrated the **end-to-end workflow** of a deep learning project:
  - Data loading & preprocessing
  - Model architecture design
  - Training & evaluation
  - Visualization of results
- Showcased the **power of CNNs** for image classification tasks
- Highlighted the importance of **data normalization** and proper **input shaping**

Potential Extensions

- Could be enhanced with **data augmentation** to improve robustness
- Might benefit from **deeper architectures** or techniques like dropout/batch normalization
- Could be **deployed as a web/mobile application** for real-world digit recognition
- Serves as a foundation for **more complex image recognition tasks**

Final Thoughts

This implementation provides a **solid baseline** for image classification problems and demonstrates how even a relatively simple CNN can achieve excellent results on well-structured image data. The project serves as an excellent **starting point** for anyone learning about deep learning and computer vision.