AI PROJECT REPORT(AI101B)

EVEN SEMESTER

SESSION 2024-25

Handwritten Digit Recognition

SHALENDRA SHARMA (202410116100191)
RISHU AGARWAL (202410116100168)
RITIK (202410116100169)
SANIDHYA GARG (202410116100182)

PROJECT SUPERVISOR

KOMAL SALGOTRA
ASSISTANT PROFESSOR

1. INTRODUCTION

Handwritten Digit Recognition is a classic and fundamental task in the field of Artificial Intelligence (AI), especially in computer vision. The goal of this task is to develop a model that can correctly identify and classify digits written by hand from images. Such systems have wide applications including automated form reading, postal code recognition, banking (check processing), and more.

To address this problem, we use the MNIST dataset, a well-known benchmark dataset consisting of 60,000 training images and 10,000 testing images of digits (0 to 9). Each image is a 28x28 pixel grayscale image. To effectively handle image data, Convolutional Neural Networks (CNNs) are employed due to their excellent performance in image classification tasks.

2. OBJECTIVES

- **Develop an AI system**: Create a machine learning model that can recognize handwritten digits with high accuracy.
- **Understand CNN architecture**: Gain insights into how convolutional layers work and how they contribute to image classification.
- **Train and test the model**: Use a dataset (MNIST) to train the model and evaluate its performance on unseen data.
- **Visualize the results**: Display model predictions for better interpretability and verification of performance.

3. TOOLS & TECHNOLOGIES USED

- **Python**: A high-level programming language known for its simplicity and large ecosystem.
- **TensorFlow and Keras**: Popular open-source libraries for building and training

deep learning models.

- **NumPy**: Useful for numerical operations on data arrays.
- **Matplotlib**: Used for visualizing the data and model predictions.
- **Jupyter Notebook / Google Colab**: Interactive development environments used for running and testing code.

4. AGENTS AND ENVIRONMENT

In AI, an agent is an entity that perceives its environment and acts upon it to achieve a goal. In this project:

- **Agent**: The CNN model which processes image inputs and outputs predictions.
- **Sensors**: The pixel data from input images.
- **Actuators**: The output prediction (digit 0–9).
- **Environment**: The set of images and labels in the MNIST dataset.

This environment is:

- **Deterministic**: Given the same input, the model always outputs the same prediction.
- **Static**: The dataset does not change during model execution.
- **Fully Observable**: All information required for decision-making is available to the agent.
- **Discrete**: There is a finite number of possible actions (digit classes).

5. PROBLEM SPECIFICATION

- **Problem Statement**: Design a system that can automatically recognize and classify handwritten digits from 0 to 9.
- **Inputs**: Grayscale images of size 28x28 pixels.
- **Output**: A single digit (0–9) representing the model's classification.
- **Constraints**:
    - Images must be preprocessed (normalized and reshaped).
    - The system should generalize well to unseen data.
- **Goal**: Achieve high classification accuracy using an efficient and effective CNN model.

6. METHODOLOGY

6.1 Data Preprocessing

Data preprocessing involves preparing the raw image data for input into the model:

- **Normalization**: Scale pixel values from [0, 255] to [0, 1] to improve convergence speed.
- **Reshaping**: Convert 2D images to 4D arrays by adding a channel dimension (28x28x1) required for CNNs.

6.2 CNN Architecture

A Convolutional Neural Network is composed of several layers designed to detect patterns:

- **Conv2D Layer 1**: Applies 32 filters of size 3x3 to extract low-level features like edges.
- **MaxPooling2D Layer 1**: Reduces the spatial dimensions (downsampling) to

retain dominant features.

- **Conv2D Layer 2**: Applies 64 filters to extract more complex features.
- **MaxPooling2D Layer 2**: Further reduces the image size.
- **Flatten Layer**: Converts the 3D output into a 1D vector for the fully connected layer.
- **Dense Layer**: A fully connected layer with 128 neurons to combine features for final classification.
- **Output Layer**: Uses softmax activation to output probabilities for each of the 10 classes.

6.3 Model Compilation and Training

The model is compiled and trained using:

- **Optimizer**: Adam, an adaptive optimizer that adjusts learning rates.
- **Loss Function**: Sparse Categorical Crossentropy, suitable for multiclass classification with integer labels.
- **Metrics**: Accuracy is used to measure how often predictions match labels.
- **Training**: The model is trained over 5 epochs with training and validation sets.

6.4 Evaluation and Prediction

After training, the model is evaluated on the test dataset to determine its performance:

- **Evaluation Metrics**: Loss and accuracy.
- **Prediction**: The model outputs probabilities for each digit class.
- **Visualization**: Matplotlib is used to display images with predicted labels.

7. FUTURE WORK

- **Transfer Learning**: Use pre-trained models to improve accuracy and training speed.
- **Deploy on Web/Mobile**: Integrate the model with a user-friendly interface.
- **Multi-language Recognition**: Expand recognition to other scripts and languages.
- **Advanced Architectures**: Experiment with deeper CNNs, dropout, and batch normalization to enhance performance.

8. CONCLUSION

The Handwritten Digit Recognition project successfully applied CNNs to solve a real-world image classification problem. The model was trained and evaluated on the MNIST dataset, achieving high accuracy and robust performance. The project also highlighted the importance of data preprocessing, model design, and evaluation techniques in developing AI systems.

This system can serve as a foundation for more complex optical character recognition systems and can be extended to recognize alphabets, symbols, and even cursive handwriting.

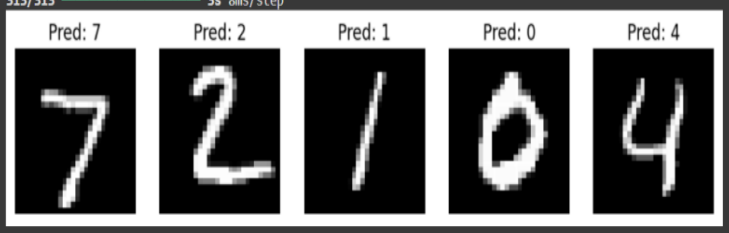
CODE:

```
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
# Normalize the pixel values to [0, 1] range
x_train, x_test = x_train / 255.0, x_test / 255.0
# Reshape data to add a channel dimension (required for CNNs)
```

```
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)
# Build CNN model
model = keras.Sequential([
keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28,
1)),keras.layers.MaxPooling2D((2, 2)),
keras.layers.Conv2D(64, (3, 3), activation='relu'),
keras.layers.MaxPooling2D((2, 2)),
keras.layers.Flatten(),
keras.layers.Dense(128, activation='relu'),
keras.layers.Dense(10, activation='softmax')
])
# 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_acc = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {test_acc:.4f}")
# Make predictions
predictions = model.predict(x_test)
# Display a few predictionsfig, axes = plt.subplots(1, 5, figsize=(10, 2))
for i, ax in enumerate(axes):
ax.imshow(x_test[i].reshape(28, 28), cmap='gray')
ax.set_title(f"Pred: {predictions[i].argmax()}")
ax.axis('off')
plt.show()
```

OUTPUT

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 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
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 66s 34ms/step - accuracy: 0.9117 - loss: 0.2874 - val_accuracy: 0.9872 - val_loss: 0.0405
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 79s 33ms/step - accuracy: 0.9866 - loss: 0.0433 - val_accuracy: 0.9880 - val_loss: 0.0340
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 82s 33ms/step - accuracy: 0.9918 - loss: 0.0269 - val_accuracy: 0.9901 - val_loss: 0.0282
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 82s 33ms/step - accuracy: 0.9933 - loss: 0.0200 - val_accuracy: 0.9887 - val_loss: 0.0382
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 83s 33ms/step - accuracy: 0.9962 - loss: 0.0121 - val_accuracy: 0.9904 - val_loss: 0.0336
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 0.9870 - loss: 0.0419
Test Accuracy: 0.9904
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step
```



6m 43s completed at 9:56 AM

EXPLANATION

1. Loading the Dataset

The MNIST dataset is a standard benchmark in AI and machine learning. It contains:

- **60,000 training images** used to teach the model.
- **10,000 test images** used to evaluate how well the model performs on unseen data.

2. Preprocessing the Data

Before feeding the images into the CNN:

- **Normalization**: All pixel values are scaled from the range `[0–255]` to `[0–1]`. This helps the model train faster and perform better.
- **Reshaping**: Since the model expects 3D image input (height, width, channels), the grayscale images are reshaped to add a "channel" dimension (28x28x1).

3. Building the CNN Model

The model is made up of several layers:

. Convolutional Layers

- The first layer scans the image with **32 filters** (small 3x3 windows) to detect features like edges and corners.
- A **Max Pooling layer** reduces the size of the image, keeping the most important information.
- A second set of **64 filters** is applied to detect more complex patterns (shapes, curves, etc.).
- Another **Max Pooling** step simplifies the feature maps even further.

. Flattening and Dense Layers

- The 2D output from the previous layers is **flattened** into a 1D vector.
- A **Dense layer** with 128 neurons processes this data to find patterns useful for classification.
- The **final layer** has **10 neurons**, each representing a digit from 0 to 9. It uses the **softmax function** to output probabilities for each digit.

4. Compiling the Model

The model is now ready to be trained. It’s configured with:

- An **optimizer (Adam)** to adjust the internal weights during training.
- A **loss function** that measures how far off the predictions are from the true labels.
- An **accuracy metric** to keep track of how well it's performing.

5. Training the Model

The model is trained for **5 cycles (epochs)** through the entire training dataset. It uses the test set after each epoch to check how well it’s generalizing to new data.

6. Evaluating the Model

After training, the model is tested on the **10,000 test images**. It calculates:

- The **loss** (how wrong the predictions are).
- The **accuracy** (how many predictions were correct), which is then printed.

7. Making Predictions

The model then processes the test images to **predict the digits** it sees. It doesn't just give one answer—it assigns a probability to each digit (0–9), and the one with the highest probability is chosen as the final prediction.

8. Vislizing the Results

To help you **see how the model performs**, five test images are displayed. For each image:

- The digit is shown.
- The predicted digit is labeled on top of the image. This is a great way to quickly verify if the model is predicting correctly or not.