Ex-5: Build a Convolutional Neural Network for MNIST Handwritten Digit Classification

Objective

To design and implement a **Convolutional Neural Network (CNN)** for recognizing handwritten digits from the **MNIST dataset** using **TensorFlow/Keras**.

---

1. Introduction

The **MNIST dataset** consists of **28x28 grayscale images** of handwritten digits (0-9). The goal is to classify each image into one of these 10 classes using a **CNN model**.

Dataset Details:

- **60,000 training images**
- **10,000 test images**
- **10 classes (digits 0-9)**

Tools & Libraries Required:

- Python 3.x
- TensorFlow/Keras
- NumPy, Matplotlib, Seaborn
- Scikit-learn

---

2. Steps to Implement the CNN Model

Step 1: Install & Import Libraries

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
import seaborn as sns
```

Step 2: Load and Preprocess the Dataset

```
# Load MNIST dataset
from tensorflow.keras.datasets import mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Reshape data to match CNN input format (28x28x1)
X_train = X_train.reshape(-1, 28, 28, 1).astype('float32') / 255.0
X_test = X_test.reshape(-1, 28, 28, 1).astype('float32') / 255.0

# Convert labels to one-hot encoding
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

Explanation:

- The dataset is reshaped to **(28, 28, 1)** to match the CNN input format.
- Pixel values are **normalized** to the range [0,1].
- Labels are converted to **one-hot encoding** for multi-class classification.

Step 3: Define the CNN Model

```
# Define CNN architecture
model = Sequential([
    Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=(28,28,1)),
    MaxPooling2D(pool_size=(2,2)),
    Conv2D(64, kernel_size=(3,3), activation='relu'),
    MaxPooling2D(pool_size=(2,2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
```

```
])
```

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

Explanation:

- Conv2D(32, (3,3), activation='relu'): Extracts features using **32 filters** of size 3x3.
- MaxPooling2D(2,2): Reduces spatial dimensions to avoid overfitting.
- Conv2D(64, (3,3), activation='relu'): More filters to extract deeper features.
- Flatten(): Converts 2D feature maps into a **1D feature vector**.
- Dense(128, activation='relu'): Fully connected layer with **128 neurons**.
- Dropout(0.5): Regularization to **reduce overfitting**.
- Dense(10, activation='softmax'): Output layer for **multi-class classification**.
- Adam optimizer: Efficient optimization algorithm.
- Categorical Crossentropy: Loss function for multi-class classification.

Step 4: Train the Model

```
# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=128, validation_data=(X_test, y_test))
```

Key Parameters:

- epochs=10: Number of training cycles.
- batch_size=128: Number of samples processed per step.
- validation_data: Evaluates performance on test data.

Step 5: Evaluate Model Performance

```
# Evaluate on test data
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss}")
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

Accuracy (%): Measures model performance on unseen data.

Step 6: Visualize Training Results

```
# Plot accuracy trends
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

Interpretation:

- Higher accuracy indicates good model generalization.
- If validation accuracy is lower than training accuracy, **overfitting** may be occurring.

Step 7: Make Predictions & Visualize Results

```
import numpy as np
def predict_digit(index):
    img = X_test[index].reshape(1, 28, 28, 1)
    prediction = model.predict(img)
    predicted_label = np.argmax(prediction)
    actual_label = np.argmax(y_test[index])
    plt.imshow(X_test[index].reshape(28, 28), cmap='gray')
    plt.title(f"Predicted: {predicted_label}, Actual: {actual_label}")
    plt.show()
# Test prediction
predict_digit(0)  # Change index to test different images
```

---

3. Observations & Conclusions

1. **CNNs perform exceptionally well on image classification problems.**
2. **Increasing convolutional layers** can extract more complex features.
3. **Overfitting can be reduced** using dropout layers.
4. **Hyperparameter tuning** (e.g., learning rate, batch size) can improve performance.

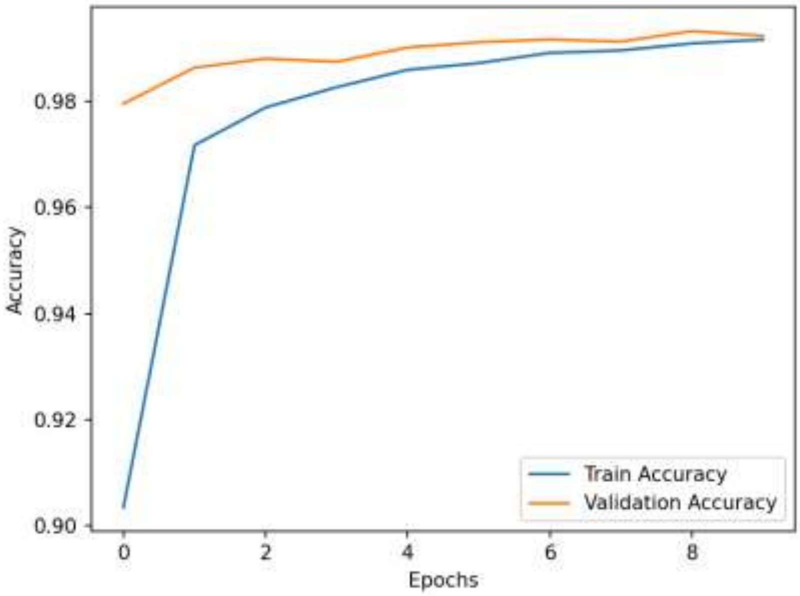
---

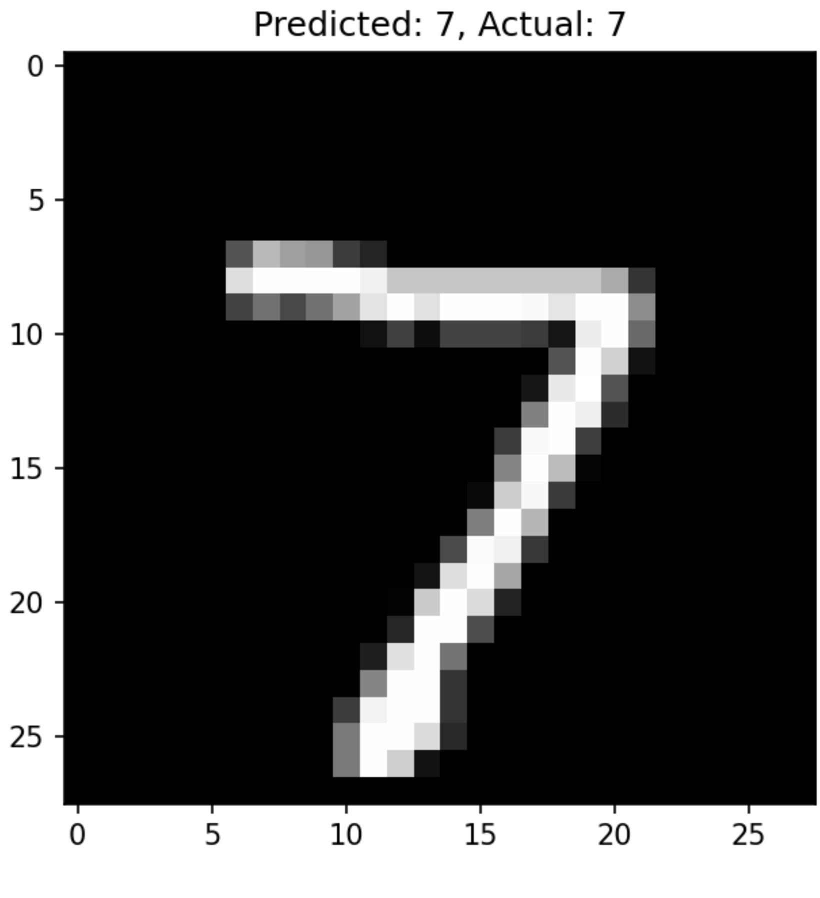
5. Summary Table

Step	Task
1	Import Libraries
2	Load and Preprocess Dataset
3	Define CNN Model Architecture
4	Train the Model
5	Evaluate Model Performance
6	Visualize Training Results
7	Make Predictions and Visualize

OutPut:

```
Epoch 1/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 14s 22ms/step - accuracy: 0.7913 - loss: 0.6556 - val_accuracy: 0.9795 - val_loss: 0.0647
Epoch 2/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 9s 20ms/step - accuracy: 0.9690 - loss: 0.1061 - val_accuracy: 0.9863 - val_loss: 0.0395
Epoch 3/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 10s 20ms/step - accuracy: 0.9773 - loss: 0.0747 - val_accuracy: 0.9880 - val_loss: 0.0348
Epoch 4/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 10s 20ms/step - accuracy: 0.9818 - loss: 0.0584 - val_accuracy: 0.9874 - val_loss: 0.0348
Epoch 5/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 10s 21ms/step - accuracy: 0.9855 - loss: 0.0492 - val_accuracy: 0.9901 - val_loss: 0.0293
Epoch 6/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 9s 20ms/step - accuracy: 0.9872 - loss: 0.0424 - val_accuracy: 0.9911 - val_loss: 0.0271
Epoch 7/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 9s 20ms/step - accuracy: 0.9892 - loss: 0.0350 - val_accuracy: 0.9916 - val_loss: 0.0254
Epoch 8/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 10s 22ms/step - accuracy: 0.9894 - loss: 0.0322 - val_accuracy: 0.9912 - val_loss: 0.0254
Epoch 9/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 10s 21ms/step - accuracy: 0.9904 - loss: 0.0287 - val_accuracy: 0.9932 - val_loss: 0.0232
Epoch 10/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 11s 23ms/step - accuracy: 0.9918 - loss: 0.0251 - val_accuracy: 0.9923 - val_loss: 0.0233
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9905 - loss: 0.0286
Test Loss: 0.023328689858317375
Test Accuracy: 99.23%
```





6. Result

This lab demonstrated how to build a **CNN for digit recognition** using the **MNIST dataset**. The model successfully classified handwritten digits with **high accuracy**. Future improvements can be made using **data augmentation, deeper networks, and hyperparameter tuning**.