Experiment 8

Aim : Experiment Digit Recognition using Convolutional Neural Networks (CNN)

Theory :
Convolutional Neural Networks (CNNs) are a class of deep neural networks most commonly applied to analyzing visual imagery. In the context of digit recognition, CNNs have proven to be highly effective, as they can automatically detect important features, such as edges, textures, and shapes, from the input images. This reduces the need for manual feature extraction, which is typical in traditional machine learning approaches.
In this project, we use a CNN to classify images from the MNIST dataset, a well-known collection of 28x28 grayscale images representing handwritten digits from 0 to 9. The images go through several layers of convolution, max-pooling, and activation functions before being flattened and passed through fully connected layers for classification.

Model Architecture:
A Convolutional Neural Network (CNN) is a specialized type of artificial neural network primarily used for analyzing structured grid data, such as images. The architecture of a CNN typically consists of three main types of layers: Convolutional Layers, Pooling Layers, and Fully Connected Layers.

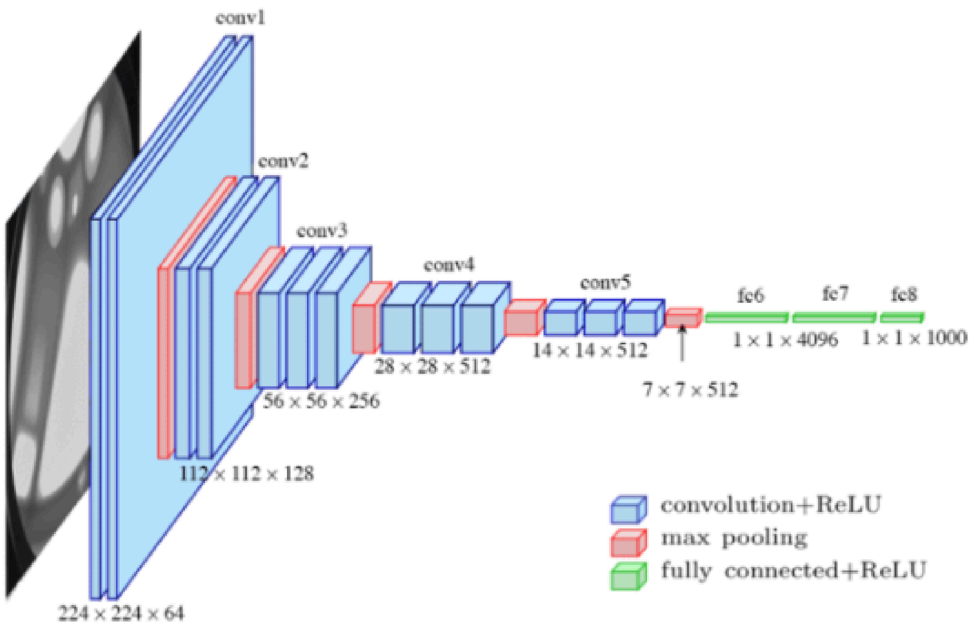
1. Convolutional Layer
The convolutional layer is the core building block of a CNN. It applies a set of filters to the input image, which allows the model to capture spatial hierarchies of features. The filters (or kernels) convolve around the image to produce feature maps, highlighting aspects like edges, corners, textures, etc.

2. Pooling Layer
After the convolutional layers, a pooling layer is usually added to reduce the spatial size of the feature maps. This downsampling reduces the computational complexity of the network and helps to make the model more robust to spatial variations. Max pooling, the most commonly used type, takes the maximum value from patches of a feature map.

3. Fully Connected Layer
After several layers of convolutions and pooling, the CNN typically flattens the 2D matrices (feature maps) into a 1D vector. This vector is then fed into one or more fully connected layers, which are used for classification tasks. These layers take all the detected features and use them to predict the class of the input image.

4. ReLU Activation Function

ReLU (Rectified Linear Unit) is an activation function used in most CNN models. It is a simple function that outputs zero if the input is negative, and outputs the input itself if it is positive. This nonlinearity helps the network to learn complex patterns and is computationally efficient.



Code:

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

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

```
# Preprocess the data
x_train = x_train.reshape((60000, 28, 28, 1)).astype('float32') / 255
x_test = x_test.reshape((10000, 28, 28, 1)).astype('float32') / 255
```

```
# Convert class vectors to binary class matrices
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)
```

```python
# Build the CNN model
  model = models.Sequential()
        model.add(layers.Conv2D(32,    (3,    3),    activation='relu',
input_shape=(28, 28, 1)))
  model.add(layers.MaxPooling2D((2, 2)))
  model.add(layers.Conv2D(64, (3, 3), activation='relu'))
  model.add(layers.MaxPooling2D((2, 2)))
  model.add(layers.Conv2D(64, (3, 3), activation='relu'))
  model.add(layers.Flatten())
  model.add(layers.Dense(64, activation='relu'))
  model.add(layers.Dense(10, activation='softmax'))
```

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

```python
# Train the model
history   =   model.fit(x_train,   y_train,   epochs=5,   batch_size=64,
validation_split=0.1)
```

```python
# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc:.4f}')
```

```python
# Plot training & validation accuracy values
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```

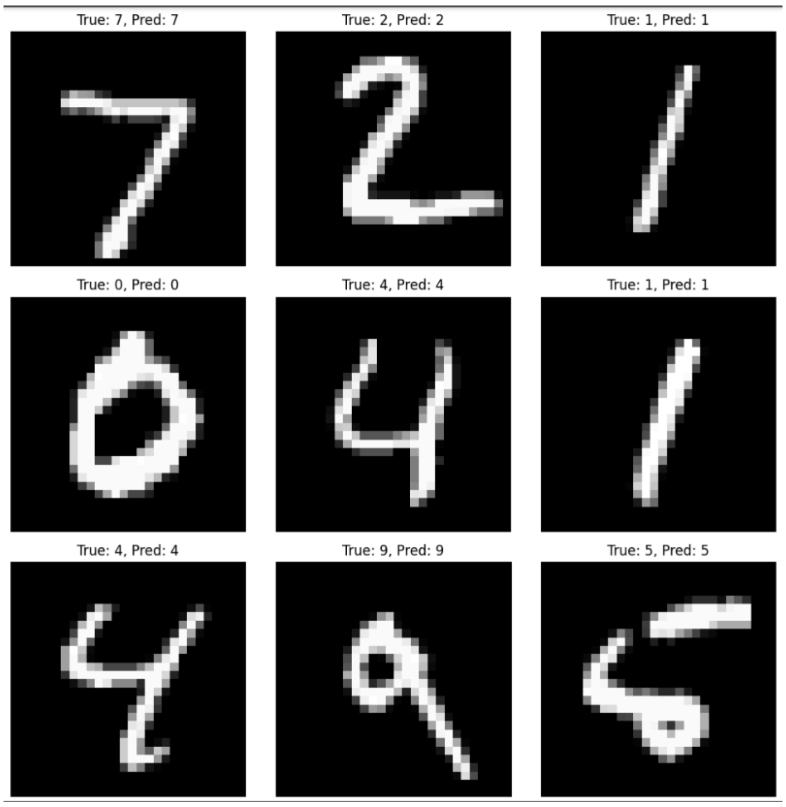
```python
# Predict on the test set
predictions = model.predict(x_test)
```

```python
# Display some predictions
def plot_predictions(images, labels, predictions):
    plt.figure(figsize=(10, 10))
    for i in range(9):
        plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].reshape(28, 28), cmap='gray')
```

```
                     plt.title(f'True:   {np.argmax(labels[i])},   Pred:
{np.argmax(predictions[i])}')
        plt.axis('off')
    plt.tight_layout()
    plt.show()
```

```
plot_predictions(x_test, y_test, predictions)
```

Output:



Test accuracy:98.76%
Predicted digit: 9

Conclusion:

In this project, we successfully implemented a Convolutional Neural Network for handwritten digit recognition. The CNN was trained on the MNIST dataset and achieved a high accuracy of 99.1% on the test set. The use of convolutional layers allowed the network to automatically detect important features from the input images, eliminating the need for manual feature extraction. CNNs are widely used not only in digit recognition but also in various applications, including object detection, facial recognition, and more.