ALEXNET CODE : AlexNet is a deep CNN architecture, designed for image classification, consisting of multiple convolutional, pooling, and fully connected layers, with ReLU activation functions and dropout regularization.

1. LOAD THE DATASET¶

In [1]:

```
# Import all the required libraries
import tensorflow as tf
import keras
import numpy as np
from tensorflow.keras.datasets import mnist


# Load MNIST dataset
mnist = tf.keras.datasets.mnist
```

In [3]:

```
# Import the required libraries
from sklearn.model_selection import train_test_split

# Split the dataset - train and test datasets
```

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

In [4]:

```
# Print the shape of the datasets
print(f"Training data shape: {X_train.shape}")
print(f"Test data shape: {X_test.shape}")
```

```
Training data shape: (60000, 28, 28)
Test data shape: (10000, 28, 28)
```

2. PREPROCESS THE DATASET¶

- **Reshape**: Converts images to 4D tensors (batch, height, width, channels) required by CNNs.
- **Normalize**: Scales pixel values to [0, 1] for stable and faster training.
- **One-hot encode**: Converts labels into vectors for multi-class classification.

In [5]:

```
# Reshape the data to add the channel dimension (28, 28, 1)
X_train = X_train.reshape((X_train.shape[0], 28, 28, 1)).astype('float32')
```

```
X_test = X_test.reshape((X_test.shape[0], 28, 28, 1)).astype('float32')
```

In [6]:

```
# Normalize the data to the range [0, 1]
X_train /= 255.0
X_test /= 255.0
```

In [7]:

```
# Convert labels to one-hot encoding
y_train = tf.keras.utils.to_categorical(y_train)
y_test = tf.keras.utils.to_categorical(y_test)
```

3. TRAIN THE MODEL¶

In [12]:

```
# Import the required library
from tensorflow.keras import layers, models

# Define the AlexNet-like model
def create_alexnet_model():
```

```
    model = models.Sequential()

    # Input Layer (Added using Input Layer for compatibility)
    model.add(layers.Input(shape=(28, 28, 1)))

    # Layer 1: Convolutional Layer
    model.add(layers.Conv2D(filters=96, kernel_size=(11, 11), strides=(1,
1), activation='relu'))
    model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

    # Layer 2: Convolutional Layer
    model.add(layers.Conv2D(filters=256, kernel_size=(5, 5),
padding='same', activation='relu'))
    model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

    # Layer 3: Convolutional Layer
    model.add(layers.Conv2D(filters=384, kernel_size=(3, 3),
padding='same', activation='relu'))

    # Layer 4: Convolutional Layer
    model.add(layers.Conv2D(filters=384, kernel_size=(3, 3),
padding='same', activation='relu'))

    # Layer 5: Convolutional Layer
    model.add(layers.Conv2D(filters=256, kernel_size=(3, 3),
padding='same', activation='relu'))
    model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

    # Flatten and Fully Connected Layers
    model.add(layers.Flatten())
    model.add(layers.Dense(4096, activation='relu'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(4096, activation='relu'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(10, activation='softmax'))

    return model

# Instantiate the model
model = create_alexnet_model()

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
```

```
metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=1, batch_size=128, validation_split=0.1)

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

```
422/422 ━━━━━━━━━━━━━━━━━━━━ 1449s 3s/step - accuracy: 0.6663 -
loss: 0.8992 - val_accuracy: 0.9843 - val_loss: 0.0539
313/313 ━━━━━━━━━━━━━━━━━━━━ 56s 179ms/step - accuracy: 0.9807 -
loss: 0.0656
Test accuracy: 0.9840
```

In [13]:

```
# Print a summary of the model
model.summary()
```

Model: "sequential_3"

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ conv2d_15 (Conv2D)              │ (None, 18, 18, 96)     │
11,712 │
```

max_pooling2d_9 (MaxPooling2D)	(None, 9, 9, 96)	0
conv2d_16 (Conv2D)	(None, 9, 9, 256)	614,656
max_pooling2d_10 (MaxPooling2D)	(None, 4, 4, 256)	0
conv2d_17 (Conv2D)	(None, 4, 4, 384)	885,120
conv2d_18 (Conv2D)	(None, 4, 4, 384)	1,327,488
conv2d_19 (Conv2D)	(None, 4, 4, 256)	884,992
max_pooling2d_11 (MaxPooling2D)	(None, 2, 2, 256)	0
flatten_2 (Flatten)	(None, 1024)	0
dense_6 (Dense)	(None, 4096)	4,198,400
dropout_4 (Dropout)	(None, 4096)	0
dense_7 (Dense)	(None, 4096)	16,781,312
dropout_5 (Dropout)	(None, 4096)	0
dense_8 (Dense)	(None, 10)	40,970

```
Total params: 74,233,952 (283.18 MB)

Trainable params: 24,744,650 (94.39 MB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 49,489,302 (188.79 MB)
```

4. TEST THE MODEL¶

In [14]:

```python
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f'Test accuracy: {test_accuracy:.4f}')
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 57s 181ms/step - accuracy: 0.9807 -
loss: 0.0656
Test accuracy: 0.9840
```

In [15]:

```
# IMPORT THE REQUIRED LIBRARIES
import matplotlib.pyplot as plt

# Visualize some predictions
def plot_predictions(X, y_true, y_pred, num=5):
    plt.figure(figsize=(10, 5))
    for i in range(num):
        plt.subplot(1, num, i + 1)
        plt.imshow(X[i].reshape(28, 28), cmap='gray')
        plt.title(f"True: {np.argmax(y_true[i])}\nPred:
{np.argmax(y_pred[i])}")
        plt.axis('off')
    plt.show()


# Get predictions
y_pred = model.predict(X_test[:5])

# Plot predictions for the first 5 test images
plot_predictions(X_test[:5], y_test[:5], y_pred)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 149ms/step
```

In [ ]: