LENET CODE : Lenet is a convolutional neural network (CNN) architecture, designed for handwritten digit recognition, consisting of convolutional, pooling, and fully connected layers.

1. LOAD THE DATASET¶

In [ ]:

```
# 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 [ ]:

```
# 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()
```

```
Downloading data from
https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

In [ ]:

```
# 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.

A. Channel Dimension: Convolutional Neural Networks (CNNs) expect a 4D input tensor for image data, with the dimensions being (batch_size, height, width, channels). The MNIST images are grayscale, so they have only one channel. By reshaping the data, you add this channel dimension,

making the data compatible with CNN input requirements.

B.Data Type: Converting the data to float32 ensures compatibility with TensorFlow and helps maintain numerical precision during computations.

- **Normalize**: Scales pixel values to [0, 1] for stable and faster training.

A. Scaling: Normalizing pixel values to the range [0, 1] (by dividing by 255) helps in speeding up the training process. It ensures that the model's input features have a consistent scale, which is crucial for neural network training.

B. Stability: Normalization reduces the risk of numerical instability and helps the optimizer converge faster by avoiding issues with large input values.

- **One-hot encode**: Converts labels into vectors for multi-class classification.

A. Multi-Class Classification: The MNIST dataset has 10 classes (digits 0-9). One-hot encoding converts the labels into a binary matrix where each label is represented by a vector of length 10, with a 1 indicating the correct class and 0s elsewhere. This format is required for the categorical cross-entropy loss function used in multi-class classification problems.

B. Training: The neural network will use this one-hot encoded format to compute the loss and update weights correctly.

In [ ]:

```
# 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 [ ]:

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

```
X_test /= 255.0
```

In [ ]:

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

In [ ]:

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

```
Training data shape after reshaping: (60000, 28, 28, 1)
Test data shape after reshaping: (10000, 28, 28, 1)
```

3. TRAIN THE MODEL¶

In [ ]:

```
# Define the LeNet-5 model
def create_lenet_model():
    model = models.Sequential()

    # Input layer
    model.add(layers.Input(shape=(28, 28, 1)))

    # Layer 1: Convolutional Layer C1
    model.add(layers.Conv2D(filters=6, kernel_size=(5, 5),
activation='relu'))

    # Average Pooling Layer P2
    model.add(layers.AveragePooling2D(pool_size=(2, 2)))

    # Layer 2: Convolutional Layer C3
    model.add(layers.Conv2D(filters=16, kernel_size=(5, 5),
activation='relu'))

    # Average Pooling Layer P4
    model.add(layers.AveragePooling2D(pool_size=(2, 2)))

    # Layer 3: Fully Connected Layer C5
    model.add(layers.Flatten())
    model.add(layers.Dense(120, activation='relu'))

    # Layer 4: Fully Connected Layer F6
    model.add(layers.Dense(84, activation='relu'))

    # Output Layer: Fully Connected Layer F7
    model.add(layers.Dense(10, activation='softmax'))

    return model

# Instantiate the model
model = create_lenet_model()

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

```
# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=128,
validation_split=0.1)
```

```
Epoch 1/10
422/422 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 27s 56ms/step - accuracy: 0.7665 -
loss: 0.8239 - val_accuracy: 0.9608 - val_loss: 0.1438
Epoch 2/10
422/422 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 41s 56ms/step - accuracy: 0.9510 -
loss: 0.1632 - val_accuracy: 0.9717 - val_loss: 0.0986
Epoch 3/10
422/422 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 42s 57ms/step - accuracy: 0.9680 -
loss: 0.1050 - val_accuracy: 0.9772 - val_loss: 0.0800
Epoch 4/10
422/422 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 23s 55ms/step - accuracy: 0.9744 -
loss: 0.0863 - val_accuracy: 0.9798 - val_loss: 0.0745
Epoch 5/10
422/422 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 41s 55ms/step - accuracy: 0.9798 -
loss: 0.0648 - val_accuracy: 0.9820 - val_loss: 0.0636
Epoch 6/10
422/422 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 40s 54ms/step - accuracy: 0.9829 -
loss: 0.0549 - val_accuracy: 0.9830 - val_loss: 0.0623
Epoch 7/10
422/422 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 43s 58ms/step - accuracy: 0.9840 -
loss: 0.0491 - val_accuracy: 0.9835 - val_loss: 0.0562
Epoch 8/10
422/422 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 41s 59ms/step - accuracy: 0.9874 -
loss: 0.0416 - val_accuracy: 0.9830 - val_loss: 0.0564
Epoch 9/10
422/422 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 42s 61ms/step - accuracy: 0.9876 -
loss: 0.0381 - val_accuracy: 0.9830 - val_loss: 0.0573
Epoch 10/10
422/422 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 40s 59ms/step - accuracy: 0.9891 -
loss: 0.0333 - val_accuracy: 0.9882 - val_loss: 0.0458
```

Out[ ]:

```
<keras.src.callbacks.history.History at 0x7918307329e0>
```

In [ ]:

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

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 24, 24, 6)	156
average_pooling2d_2 (AveragePooling2D)	(None, 12, 12, 6)	0
conv2d_4 (Conv2D)	(None, 8, 8, 16)	2,416
average_pooling2d_3 (AveragePooling2D)	(None, 4, 4, 16)	0
flatten_1 (Flatten)	(None, 256)	0
dense_3 (Dense)	(None, 120)	

```
30,840 │
├──────────────────────────────────────┼──────────────────────
──────┼─────────────────┤
│ dense_4 (Dense)                      │ (None, 84)            │
10,164 │
├──────────────────────────────────────┼──────────────────────
──────┼─────────────────┤
│ dense_5 (Dense)                      │ (None, 10)            │
850 │
└──────────────────────────────────────┴──────────────────────
──────┴─────────────────┘
```

Total params: 133,280 (520.63 KB)

Trainable params: 44,426 (173.54 KB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 88,854 (347.09 KB)

4. TEST THE MODEL¶

In [ ]:

```
# 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 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9827 -
loss: 0.0495
Test accuracy: 0.9877
```

In [ ]:

```
# 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 96ms/step
```

In [ ]:

```
#@title Convert ipynb to HTML in Colab
# Upload ipynb
from google.colab import files
f = files.upload()

# Convert ipynb to html
import subprocess
file0 = list(f.keys())[0]
_ = subprocess.run(["pip", "install", "nbconvert"])
_ = subprocess.run(["jupyter", "nbconvert", file0, "--to", "html"])

# download the html
files.download(file0[:-5]+"html")
```

Upload widget is only available when the cell has been executed in the current browser session.
Please rerun this cell to enable.

```
Saving ShreyaVemana_LenetCode (1).ipynb to ShreyaVemana_LenetCode (1).ipynb
```