Task 14: Build and Evaluate a LeNet-5 Model on MNIST

Build a Convolutional Neural Network based on the LeNet-5 architecture. Train the model on the MNIST dataset (handwritten digits) and evaluate the model's performance on test data.

1. Load the Dataset:

- Import the MNIST dataset using a framework like TensorFlow or PyTorch.
- Since the images are grayscale, they have a single channel. Add channel to ensure the data shape matches the CNN's expected input shape of (batch, height, width, channels)
- Normalize the images to a [0, 1] range and reshape them into 32x32 size.

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

(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = tf.expand_dims(x_train, axis=-1)
x_test = tf.expand_dims(x_test, axis=-1)

x_train = tf.image.resize_with_crop_or_pad(x_train, 32, 32)
x_test = tf.image.resize_with_crop_or_pad(x_test, 32, 32)

x_train = x_train/255
x_test = x_test/255

y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)
```

2. Define the LeNet-5 Architecture:

- Build the following layers:
- Input Layer: 32 x 32.
- Conv1 Layer: Convolutional layer with 6 filters, 5 x 5 kernel, stride = 1, and padding.
- Pool1 Layer: Subsampling (average pooling) with 2 x 2 kernel and stride = 2.
- Conv2 Layer: Convolutional layer with 16 filters, 5 x 5 kernel, stride = 1, and no padding.
- Pool2 Layer: Subsampling (average pooling) with 2 x 2 kernel and stride = 2.
- Flatten Layer: Flatten the feature maps.

Fully Connected Layers:

- Dense layer with 120 neurons.
- Dense layer with 84 neurons.

- Output layer with 10 neurons (for digit classification).

Use activation functions like ReLU for all layers except the output layer, which should use Softmax.

```
model = models.Sequential([
    # Conv1: Convolutional Layer
    layers.Conv2D(6, kernel_size=(5, 5), strides=1, padding='valid', activation='relu', input_shape=(32, 32, 1)),

    # Pool1: Average Pooling Layer
    layers.AveragePooling2D(pool_size=(2, 2), strides=2),

.
.
.

    # Flatten the feature maps
    layers.Flatten(),

    # Fully Connected Layer:s
    layers.Dense(….)
.
.
.
   # Output Layer

])
```

3. Compile the Model:

- Choose an appropriate loss function (categorical crossentropy for multi-class classification).
- Select an optimizer (e.g., Adam or SGD).
- Use accuracy as the evaluation metric.

```
model.compile(.....)
```

4. Train the Model:

- Split the dataset into training and validation sets.
- Train the model for 10-20 epochs and observe the accuracy and loss.

```
history = model.fit(x_train, y_train, batch_size=128, epochs=10, validation_split=0.2, verbose=2)
```

5. Evaluate the Model:

- Test the model on unseen data (test set).
- Report metrics such as accuracy, confusion matrix, or classification report.

6. Analyze the Results:

- Plot training and validation accuracy/loss curves.

Challenges:

1. Experiment with Hyperparameters:

a. Change the number of filters or kernel sizes.
b. Use different activation functions.
c. Compare results with max pooling instead of average pooling.

2. Add Data Augmentation:

a) Augment the training data (rotation, scaling, flipping, etc.) and observe its impact on performance.

3. Train a Custom CNN:

Modify the LeNet-5 architecture to create your custom CNN. For example:

b) Add an extra convolutional layer.
c) Use dropout layers for regularization.