DL: Build and Train a CNN for MNIST Handwritten Digit Recognition

The goal of this project is to build a Convolutional Neural Network (CNN) to recognize handwritten digits from the MNIST dataset. The MNIST dataset contains 70,000 grayscale images of digits (0-9), each 28x28 pixels in size. Using TensorFlow and Keras, we will create and train a deep learning model to classify these digits accurately. The project involves data preprocessing, model building, training, and evaluation to understand CNNs and their application

- Load the MNIST dataset

```
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
```

```
tf.keras.datasets.mnist.load_data()
```

```
          [0, 0, 0, ..., 0, 0, 0]],

         ...,

         [[0, 0, 0, ..., 0, 0, 0],
          [0, 0, 0, ..., 0, 0, 0],
          [0, 0, 0, ..., 0, 0, 0],
          ...,
          [0, 0, 0, ..., 0, 0, 0],
          [0, 0, 0, ..., 0, 0, 0],
          [0, 0, 0, ..., 0, 0, 0]],

         [[0, 0, 0, ..., 0, 0, 0],
          [0, 0, 0, ..., 0, 0, 0],
          [0, 0, 0, ..., 0, 0, 0],
          ...,
          [0, 0, 0, ..., 0, 0, 0],
          [0, 0, 0, ..., 0, 0, 0],
          [0, 0, 0, ..., 0, 0, 0]],

         [[0, 0, 0, ..., 0, 0, 0],
          [0, 0, 0, ..., 0, 0, 0],
          [0, 0, 0, ..., 0, 0, 0],
          ...,
          [0, 0, 0, ..., 0, 0, 0],
          [0, 0, 0, ..., 0, 0, 0],
          [0, 0, 0, ..., 0, 0, 0]]], dtype=uint8),
  array([7, 2, 1, ..., 4, 5, 6], dtype=uint8)))
```

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
(y_train)
```

```
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

- Normalize the pixel values between 0 and 1

```
x_train = x_train / 255
x_test = x_test / 255
```

```
x_train.shape[0]
```

```
60000
```

```
x_train = x_train.reshape((x_train.shape[0], 28, 28, 1))
x_test = x_test.reshape((x_test.shape[0], 28, 28, 1))
```

```
from tensorflow.keras.utils import to_categorical
```

```
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

- Build a CNN with Convolutional layers
- Include MaxPooling layers
- Add Dense layers and the correct output layer with 10 neurons and softmax activation

```
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')  # Output layer
])
```

- Use the 'adam' optimizer
- Set the loss function to 'categorical_crossentropy'.
- Track accuracy as the metric

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

- Train the model

```
history = model.fit(x_train, y_train, epochs=10,
                    batch_size=64,
                    validation_split=0.2)
```

```
Epoch 1/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 42s 55ms/step - accuracy: 0.8638 - loss: 0.4578 - val_accur
Epoch 2/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 81s 53ms/step - accuracy: 0.9817 - loss: 0.0629 - val_accur
Epoch 3/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 40s 52ms/step - accuracy: 0.9865 - loss: 0.0420 - val_accur
Epoch 4/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 41s 52ms/step - accuracy: 0.9906 - loss: 0.0307 - val_accur
Epoch 5/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 41s 52ms/step - accuracy: 0.9931 - loss: 0.0216 - val_accur
Epoch 6/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 41s 52ms/step - accuracy: 0.9939 - loss: 0.0184 - val_accur
Epoch 7/10
```

```
750/750 ━━━━━━━━━━━━━━━━━━━━ 41s 52ms/step - accuracy: 0.9952 - loss: 0.0152 - val_accur
Epoch 8/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 39s 52ms/step - accuracy: 0.9961 - loss: 0.0115 - val_accur
Epoch 9/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 41s 52ms/step - accuracy: 0.9972 - loss: 0.0092 - val_accur
Epoch 10/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 41s 51ms/step - accuracy: 0.9977 - loss: 0.0075 - val_accur
```

- Track both training and validation accuracy/loss (2 marks).
- Evaluate the model on test data

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

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.9864 - loss: 0.0577
Test accuracy: 0.9899
```

```
history.history['accuracy']
```

```
[0.9401458501815796,
 0.9826041460037231,
 0.9869999885559082,
 0.9906666874885559,
 0.9924583435058594,
 0.9942083358764648,
 0.9951249957084656,
 0.9959583282470703,
 0.996874988079071,
 0.9968958497047424]
```

```
history.history['val_accuracy']
```

```
[0.9744166731834412,
 0.9857500195503235,
 0.9856666922569275,
 0.987500011920929,
 0.9883333444595337,
 0.984250009059906,
 0.9893333315849304,
 0.9904999732971191,
 0.9883333444595337,
 0.9872499704360962]
```

- Plot accuracy and loss graphs

```
# Plot training & validation accuracy values
plt.figure(figsize=(12, 4))
```

```
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot training & validation loss values
plt.figure(figsize=(12, 4))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```