Ex.No: 3 Roll.No: 210701158

PYTHON PROGRAM TO CREATE A NEURAL NETWORK TO RECOGNIZE HANDWRITTEN DIGITS USING MNIST DATASET

Aim:

To create a neural network to recognize handwritten digits using MNIST dataset in python.

Procedure:

1. Import TensorFlow, Keras, and Matplotlib for building the model and plotting.
2. Load the MNIST dataset, consisting of handwritten digits.
3. Reshape and normalize the training and test images to have pixel values between 0 and 1.
4. Convert the training and test labels to one-hot encoded vectors.
5. Build a Sequential model and add a Conv2D layer with 32 filters and ReLU activation.
6. Add a MaxPooling layer, followed by another Conv2D layer with 64 filters and ReLU activation.
7. Add a third Conv2D layer, flatten the output, and add a Dense layer with 64 units and ReLU activation.
8. Add a final Dense layer with 10 units and softmax activation for classification.
9. Compile the model with Adam optimizer and categorical cross-entropy loss, and train it for 5 epochs with 20% validation split.
10. Evaluate the model on test data and plot the training and validation accuracy and loss over epochs.

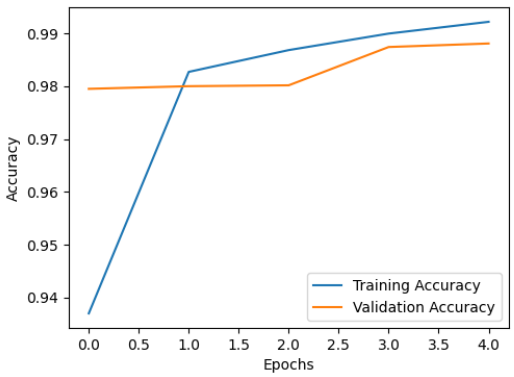
Code:

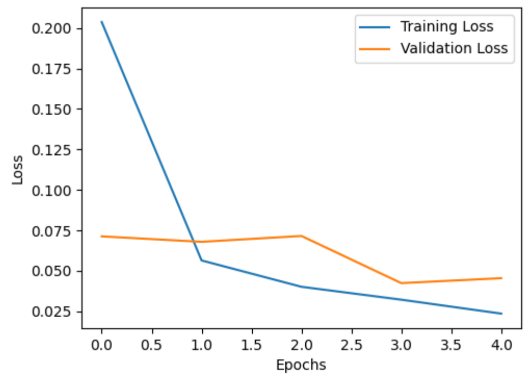
```
# Import necessary libraries
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

# Load the MNIST dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Preprocess the data
train_images = train_images.reshape((60000, 28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32') / 255

# Convert labels to one-hot encoding
train_labels = tf.keras.utils.to_categorical(train_labels)
test_labels = tf.keras.utils.to_categorical(test_labels)

# Build the neural network 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'))

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

```
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(train_images, train_labels, epochs=5, batch_size=64,
validation_split=0.2)

# Evaluate the model on test data
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test accuracy: {test_acc}')

# Plot the accuracy and loss over epochs
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

Output:

```
Epoch 1/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 9s 11ms/step - accuracy: 0.8431 - loss: 0.5073 - val_accuracy: 0.9792 - val_loss: 0.0715
Epoch 2/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 8s 11ms/step - accuracy: 0.9789 - loss: 0.0655 - val_accuracy: 0.9812 - val_loss: 0.0605
Epoch 3/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 9s 12ms/step - accuracy: 0.9865 - loss: 0.0421 - val_accuracy: 0.9861 - val_loss: 0.0473
Epoch 4/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 9s 12ms/step - accuracy: 0.9889 - loss: 0.0347 - val_accuracy: 0.9886 - val_loss: 0.0408
Epoch 5/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 9s 13ms/step - accuracy: 0.9913 - loss: 0.0287 - val_accuracy: 0.9888 - val_loss: 0.0410
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9866 - loss: 0.0403
Test accuracy: 0.9901000261306763
2024-09-16 12:59:58.143 Python[99718:1483561] +[IMKClient subclass]: chose IMKClient_Legacy
2024-09-16 12:59:58.143 Python[99718:1483561] +[IMKInputSession subclass]: chose IMKInputSession_Legacy
```





Result:

Thus, to implement neural network to recognize handwritten digits using MNIST dataset in python has been completed successfully.