ICP – 6

Venkata Suraj, Gamini
700744962

GitHub Link: https://github.com/SurajGamini18/Neural-Networks-Deep-Learning-Assignments
Video Link: https://drive.google.com/file/d/15UquAR5hBpA633bbkjgOfLn-5tJBJvVL/view?usp=sharing

Q-1:

CODE:

```python
# Imports
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the Breast Cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Normalize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the dataset into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X_scaled, y, test_size=0.25, random_state=87)

# Neural network model with additional dense layers
model = Sequential()
model.add(Dense(20, input_dim=X_train.shape[1], activation='relu'))  # Adjusted input_dim for Breast Cancer dataset
model.add(Dense(64, activation='relu'))  # Additional dense layer
model.add(Dense(64, activation='relu'))  # Additional dense layer
model.add(Dense(1, activation='sigmoid'))  # Output layer for binary classification

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

# Train the model
model.fit(X_train, Y_train, epochs=100, batch_size=10, verbose=1)

# Model summary
print(model.summary())

# Evaluate the model on the test data
loss, accuracy = model.evaluate(X_test, Y_test, verbose=0)
print(f'Test Accuracy: {accuracy*100:.2f}%')
```

EXPLANATION:

1. Dataset Loading: It loads the Breast Cancer dataset from scikit-learn, which contains features for breast cancer diagnostic data and binary labels indicating malignancy.

2. Data Preprocessing: The features are normalized using `StandardScaler` to scale the data, ensuring that the neural network model receives input data within a manageable range, improving training stability and performance.

3. Dataset Splitting: The normalized data is split into training and testing sets, with 75% used for training and 25% for testing, facilitated by the `train_test_split` method from scikit-learn.

4. Model Construction: A `Sequential` model is defined with four layers:

  - An input dense layer with 20 neurons and ReLU activation, adjusted to match the number of features in the Breast Cancer dataset.
  - Two additional dense layers, each with 64 neurons and ReLU activation, to increase the model's capacity to learn complex patterns.
  - An output layer with a single neuron and sigmoid activation to predict the binary outcome (malignant or benign).

5. Model Compilation: The model is compiled with the Adam optimizer, binary crossentropy as the loss function, and accuracy as the metric to evaluate performance.

6. Model Training: The model is trained for 100 epochs with a batch size of 10 on the normalized and split training data, with verbosity set to 1 to display training progress.

7. Evaluation: After training, the model's performance is evaluated on the test set, and the test accuracy is printed, indicating how well the model can predict breast cancer malignancy.

8. Summary Output: Finally, a summary of the model's architecture is printed, detailing the configuration of each layer, including the number of parameters in the model.

OUTPUT:

```
Epoch 85/100
43/43 [==============================] - 0s 2ms/step - loss: 1.7442e-05 - accuracy: 1.0000
Epoch 86/100
43/43 [==============================] - 0s 2ms/step - loss: 1.6820e-05 - accuracy: 1.0000
Epoch 87/100
43/43 [==============================] - 0s 2ms/step - loss: 1.5852e-05 - accuracy: 1.0000
Epoch 88/100
43/43 [==============================] - 0s 2ms/step - loss: 1.5624e-05 - accuracy: 1.0000
Epoch 89/100
43/43 [==============================] - 0s 2ms/step - loss: 1.4847e-05 - accuracy: 1.0000
Epoch 90/100
43/43 [==============================] - 0s 2ms/step - loss: 1.4237e-05 - accuracy: 1.0000
Epoch 91/100
43/43 [==============================] - 0s 2ms/step - loss: 1.3662e-05 - accuracy: 1.0000
Epoch 92/100
43/43 [==============================] - 0s 2ms/step - loss: 1.3114e-05 - accuracy: 1.0000
Epoch 93/100
43/43 [==============================] - 0s 2ms/step - loss: 1.2766e-05 - accuracy: 1.0000
Epoch 94/100
43/43 [==============================] - 0s 2ms/step - loss: 1.2172e-05 - accuracy: 1.0000
Epoch 95/100
43/43 [==============================] - 0s 2ms/step - loss: 1.1712e-05 - accuracy: 1.0000
Epoch 96/100
43/43 [==============================] - 0s 2ms/step - loss: 1.1459e-05 - accuracy: 1.0000
Epoch 97/100
43/43 [==============================] - 0s 2ms/step - loss: 1.0886e-05 - accuracy: 1.0000
Epoch 98/100
43/43 [==============================] - 0s 2ms/step - loss: 1.0455e-05 - accuracy: 1.0000
Epoch 99/100
43/43 [==============================] - 0s 2ms/step - loss: 1.0127e-05 - accuracy: 1.0000
Epoch 100/100
43/43 [==============================] - 0s 2ms/step - loss: 9.8881e-06 - accuracy: 1.0000
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 20)                620

 dense_1 (Dense)             (None, 64)                1344

 dense_2 (Dense)             (None, 64)                4160

 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 6189 (24.18 KB)
Trainable params: 6189 (24.18 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
Test Accuracy: 95.80%
```

Q-2:

CODE:

```python
from keras.datasets import mnist
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
import matplotlib.pyplot as plt

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

# Preprocess the data
dimData = np.prod(train_images.shape[1:])
train_data = train_images.reshape(train_images.shape[0], dimData).astype('float32') / 255
test_data = test_images.reshape(test_images.shape[0], dimData).astype('float32') / 255
train_labels_one_hot = to_categorical(train_labels)
test_labels_one_hot = to_categorical(test_labels)

# Creating the network
model = Sequential([
    Dense(512, activation='relu', input_shape=(dimData,)),
    Dense(512, activation='relu'),
    Dense(10, activation='softmax')
])

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

# Train the model
history = model.fit(train_data, train_labels_one_hot, batch_size=256, epochs=10, verbose=1,
                    validation_data=(test_data, test_labels_one_hot))

# Plotting the accuracy and loss
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('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.title('Loss')
plt.legend()
plt.show()
```

```python
def predict_single_image(image_data):
    image_data = image_data.reshape(1, dimData).astype('float32') / 255
    prediction = model.predict(image_data)
    predicted_class = np.argmax(prediction)
    return predicted_class

# Choose an image from the test set
image_index = 0  # Change this to see different predictions
predicted_class = predict_single_image(test_images[image_index])
print(f'Predicted class for image at index {image_index}: {predicted_class}')
plt.imshow(test_images[image_index], cmap='gray')
plt.title(f'Image at index {image_index} - Predicted as: {predicted_class}')
plt.show()

# Modify the model to use different activation functions and fewer layers for comparison
# You can adjust the following model architecture as needed
model_tanh = Sequential([
    Dense(512, activation='tanh', input_shape=(dimData,)),
    Dense(10, activation='softmax')
])

model_tanh.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
model_tanh.fit(train_data, train_labels_one_hot, batch_size=256, epochs=10, verbose=1,
               validation_data=(test_data, test_labels_one_hot))

# To compare performance without scaling, you would need to adjust the data preprocessing
# and re-train the model. This is not included here for brevity but can be done by removing
# the division by 255 in the data preprocessing steps.
```

EXPLANATION:

The provided script is a comprehensive Python program for training a neural network model on the MNIST dataset for handwritten digit recognition. It highlights key steps in data preprocessing, model training, evaluation, and experimentation with model architecture. Here's a summary in a few points:

1. MNIST Dataset: The script loads the MNIST dataset, which consists of 28x28 pixel grayscale images of handwritten digits (0 through 9) and their associated labels.

2. Data Preprocessing: The images are flattened from a 2D 28x28 format to a 1D 784 vector and normalized so that pixel values are in the range [0, 1]. Labels are converted to one-hot encoded vectors for classification.

3. Model Architecture: A `Sequential` model with two dense layers of 512 neurons each (using ReLU activation), followed by a softmax output layer for classifying the digits into 10 categories, is defined and compiled with RMSprop optimizer and categorical crossentropy loss.

4. Training and Validation: The model is trained for 10 epochs with a batch size of 256, using both training and validation data to monitor performance and avoid overfitting.

5. Performance Visualization: Training and validation accuracy and loss are plotted using matplotlib to visually assess the model's learning progress over epochs.

6. Prediction on Test Data: The script includes a function to predict the digit class for a single image from the test set, demonstrating the model's inferencing capability.

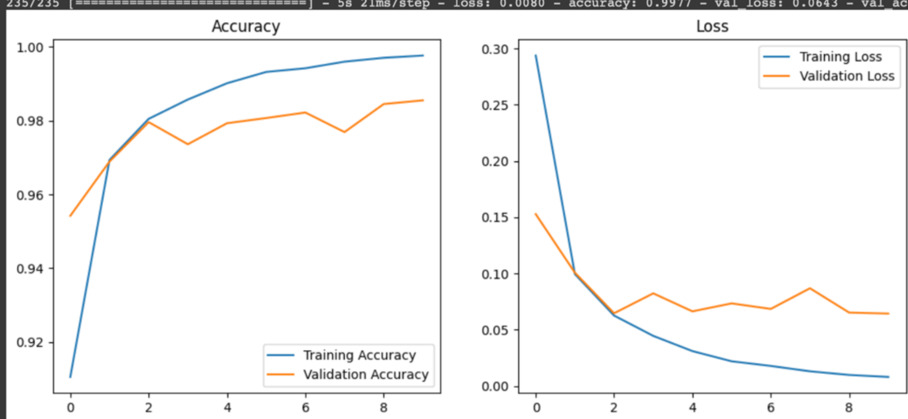
7. Model Experimentation: A variant of the model using tanh activation and a simplified architecture (one hidden layer) is trained to explore the impact of different network configurations on performance.

8. Note on Scaling: A comment mentions the significance of scaling image pixel values for model performance, suggesting an experiment to compare results with and without this preprocessing step.

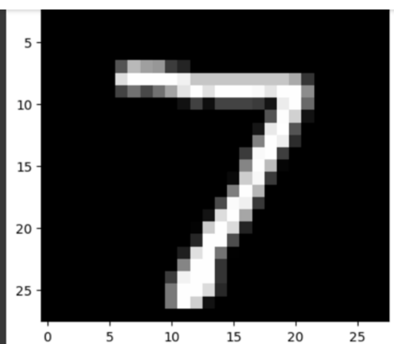
This script serves as a complete workflow for neural network training on image data, encompassing data handling, model building, training, evaluation, and experimentation with architecture modifications.

OUTPUT:

```
Epoch 1/10
235/235 [==============================] - 6s 22ms/step - loss: 0.2937 - accuracy: 0.9105 - val_loss: 0.1527 - val_accuracy: 0.9542
Epoch 2/10
235/235 [==============================] - 6s 25ms/step - loss: 0.0994 - accuracy: 0.9693 - val_loss: 0.1007 - val_accuracy: 0.9689
Epoch 3/10
235/235 [==============================] - 5s 21ms/step - loss: 0.0626 - accuracy: 0.9805 - val_loss: 0.0646 - val_accuracy: 0.9796
Epoch 4/10
235/235 [==============================] - 6s 25ms/step - loss: 0.0445 - accuracy: 0.9857 - val_loss: 0.0823 - val_accuracy: 0.9736
Epoch 5/10
235/235 [==============================] - 5s 21ms/step - loss: 0.0310 - accuracy: 0.9901 - val_loss: 0.0663 - val_accuracy: 0.9793
Epoch 6/10
235/235 [==============================] - 5s 21ms/step - loss: 0.0219 - accuracy: 0.9932 - val_loss: 0.0734 - val_accuracy: 0.9807
Epoch 7/10
235/235 [==============================] - 6s 25ms/step - loss: 0.0178 - accuracy: 0.9942 - val_loss: 0.0684 - val_accuracy: 0.9822
Epoch 8/10
235/235 [==============================] - 5s 21ms/step - loss: 0.0130 - accuracy: 0.9960 - val_loss: 0.0868 - val_accuracy: 0.9769
Epoch 9/10
235/235 [==============================] - 6s 25ms/step - loss: 0.0098 - accuracy: 0.9970 - val_loss: 0.0652 - val_accuracy: 0.9845
Epoch 10/10
235/235 [==============================] - 5s 21ms/step - loss: 0.0080 - accuracy: 0.9977 - val_loss: 0.0643 - val_accuracy: 0.9855
```




```
1/1 [==============================] - 0s 55ms/step
Predicted class for image at index 0: 7
```



```
Epoch 1/10
235/235 [==============================] - 4s 18ms/step - loss: 0.3922 - accuracy: 0.8846 - val_loss: 0.2624 - val_accuracy: 0.9238
Epoch 2/10
235/235 [==============================] - 3s 13ms/step - loss: 0.2357 - accuracy: 0.9316 - val_loss: 0.2399 - val_accuracy: 0.9265
Epoch 3/10
235/235 [==============================] - 3s 13ms/step - loss: 0.1718 - accuracy: 0.9507 - val_loss: 0.1671 - val_accuracy: 0.9502
Epoch 4/10
235/235 [==============================] - 3s 13ms/step - loss: 0.1300 - accuracy: 0.9623 - val_loss: 0.1348 - val_accuracy: 0.9605
Epoch 5/10
235/235 [==============================] - 4s 17ms/step - loss: 0.1032 - accuracy: 0.9698 - val_loss: 0.1191 - val_accuracy: 0.9635
Epoch 6/10
235/235 [==============================] - 3s 13ms/step - loss: 0.0835 - accuracy: 0.9755 - val_loss: 0.0923 - val_accuracy: 0.9715
Epoch 7/10
235/235 [==============================] - 3s 13ms/step - loss: 0.0691 - accuracy: 0.9801 - val_loss: 0.0785 - val_accuracy: 0.9761
Epoch 8/10
235/235 [==============================] - 3s 13ms/step - loss: 0.0581 - accuracy: 0.9830 - val_loss: 0.0783 - val_accuracy: 0.9750
Epoch 9/10
235/235 [==============================] - 4s 17ms/step - loss: 0.0490 - accuracy: 0.9863 - val_loss: 0.0834 - val_accuracy: 0.9738
Epoch 10/10
235/235 [==============================] - 3s 13ms/step - loss: 0.0415 - accuracy: 0.9883 - val_loss: 0.0815 - val_accuracy: 0.9760
<keras.src.callbacks.History at 0x7bc4cb0973a0>
```