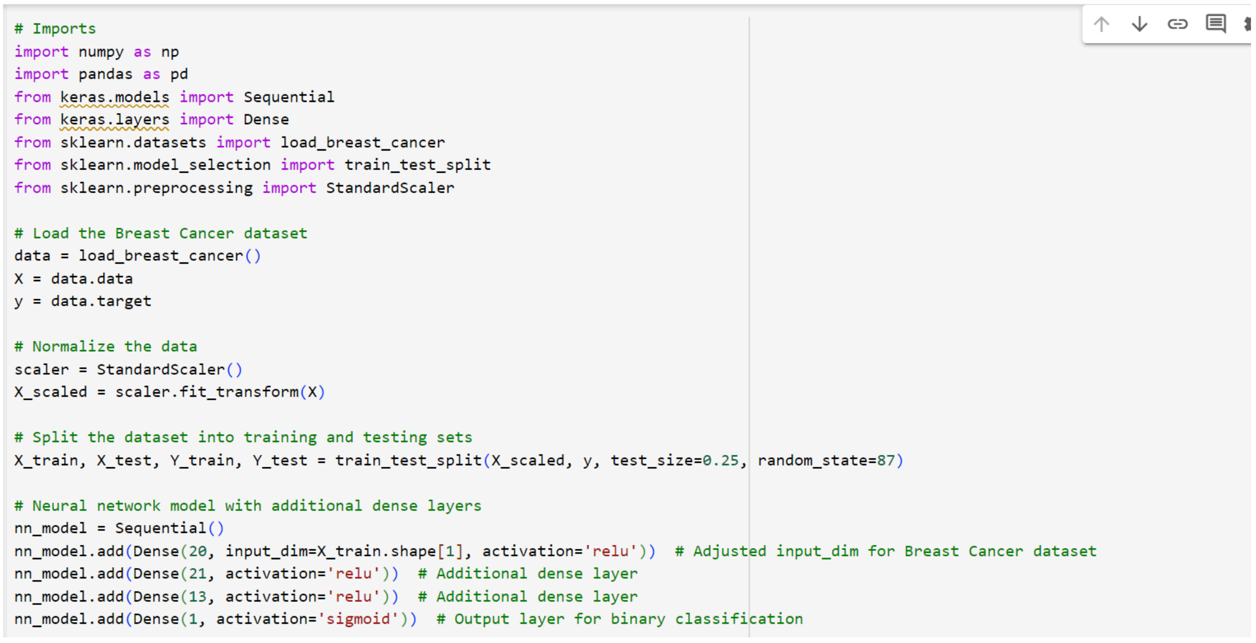
ICP – 4

Kavya Reddy Nagulapally
700759486

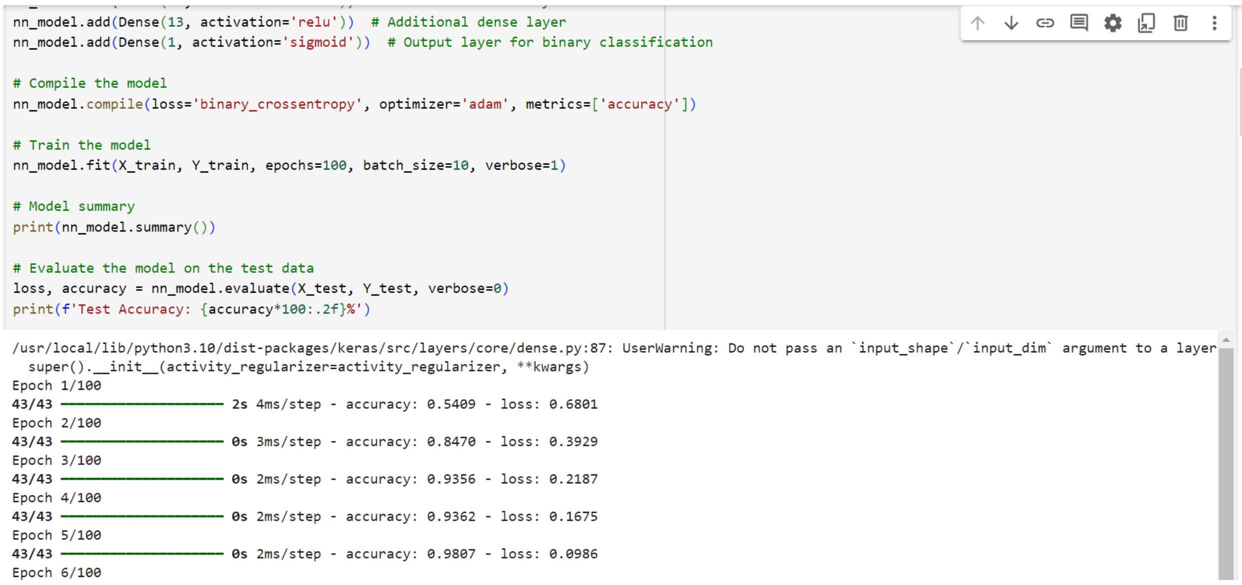
GitHub Link: https://github.com/Kavyareddy03/ICP4

Question 1

Code and Output:

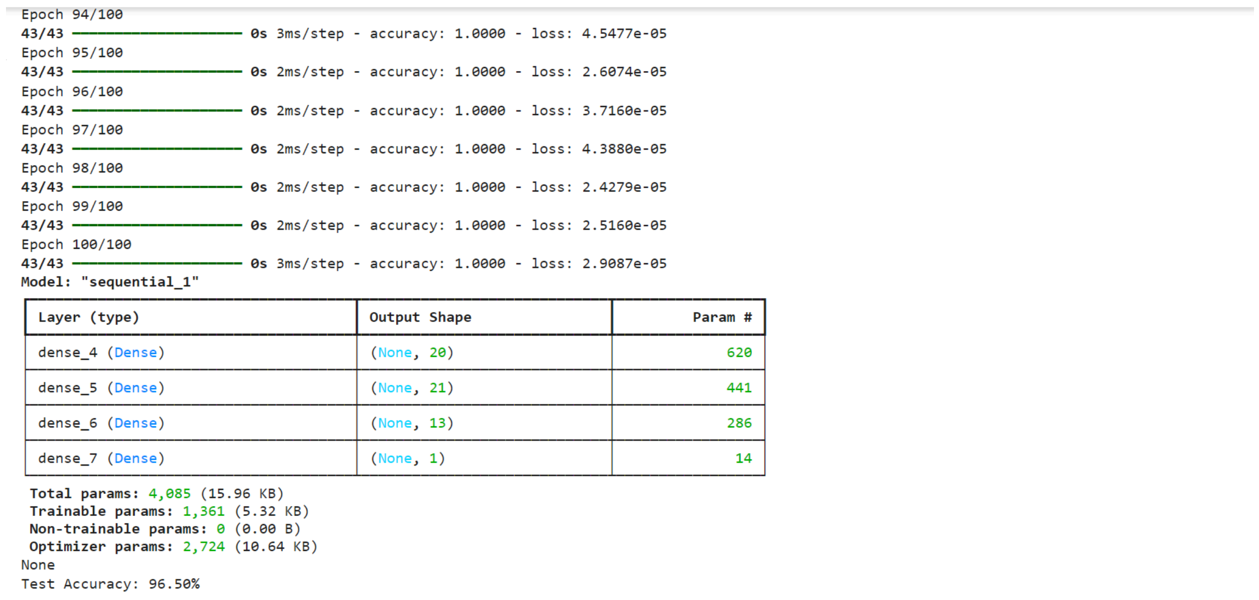


```python
# Imports
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# 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
nn_model = Sequential()
nn_model.add(Dense(20, input_dim=X_train.shape[1], activation='relu'))  # Adjusted input_dim for Breast Cancer dataset
nn_model.add(Dense(21, activation='relu'))  # Additional dense layer
nn_model.add(Dense(13, activation='relu'))  # Additional dense layer
nn_model.add(Dense(1, activation='sigmoid'))  # Output layer for binary classification
```



```python
nn_model.add(Dense(13, activation='relu'))  # Additional dense layer
nn_model.add(Dense(1, activation='sigmoid'))  # Output layer for binary classification

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

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

# Model summary
print(nn_model.summary())

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

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/100
43/43 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.5409 - loss: 0.6801
Epoch 2/100
43/43 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8470 - loss: 0.3929
Epoch 3/100
43/43 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9356 - loss: 0.2187
Epoch 4/100
43/43 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9362 - loss: 0.1675
Epoch 5/100
43/43 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9807 - loss: 0.0986
Epoch 6/100
```



```
Epoch 94/100
43/43 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 4.5477e-05
Epoch 95/100
43/43 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 2.6074e-05
Epoch 96/100
43/43 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 3.7160e-05
Epoch 97/100
43/43 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 4.3880e-05
Epoch 98/100
43/43 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 2.4279e-05
Epoch 99/100
43/43 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 2.5160e-05
Epoch 100/100
43/43 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 2.9087e-05
Model: "sequential_1"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape               ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ dense_4 (Dense)                      │ (None, 20)                 │             620 │
├──────────────────────────────────────┼────────────────────────────┼─────────────────┤
│ dense_5 (Dense)                      │ (None, 21)                 │             441 │
├──────────────────────────────────────┼────────────────────────────┼─────────────────┤
│ dense_6 (Dense)                      │ (None, 13)                 │             286 │
├──────────────────────────────────────┼────────────────────────────┼─────────────────┤
│ dense_7 (Dense)                      │ (None, 1)                  │              14 │
└──────────────────────────────────────┴────────────────────────────┴─────────────────┘
 Total params: 4,085 (15.96 KB)
 Trainable params: 1,361 (5.32 KB)
 Non-trainable params: 0 (0.00 B)
 Optimizer params: 2,724 (10.64 KB)
None
Test Accuracy: 96.50%
```

Explanation:

1. Import Libraries:
   - `numpy` and `pandas` are imported for numerical and data manipulation.
   - `Sequential` and `Dense` from Keras are used to build and define the neural network.
   - Scikit-learn's `load_breast_cancer`, `train_test_split`, and `StandardScaler` are used for loading the dataset, splitting data, and scaling features.

2. Load the Breast Cancer Dataset:
   - The dataset is loaded into `data`, from which features (`X`) and labels (`y`) are extracted.

3. Normalize the Data:
   - The `StandardScaler` standardizes the features by removing the mean and scaling to unit variance.
   - `X_scaled` contains the scaled feature values.

4. Split the Dataset:
   - The data is split into training and testing sets. `X_train` and `Y_train` are used for training, while `X_test` and `Y_test` are used for evaluating the model. 25% of the data is reserved for testing.

5. Define the Neural Network Model:
   - A Sequential model is created, which is a linear stack of layers.
   - A Dense layer with 20 neurons and ReLU activation function is added. This layer takes the input features.
   - Two additional Dense layers with 21 and 13 neurons, respectively, and ReLU activation functions are added.

- The output layer is a Dense layer with 1 neuron and a sigmoid activation function for binary classification.

6. Compile the Model:
   - The model is compiled with binary crossentropy as the loss function (suitable for binary classification), Adam optimizer, and accuracy as the evaluation metric.

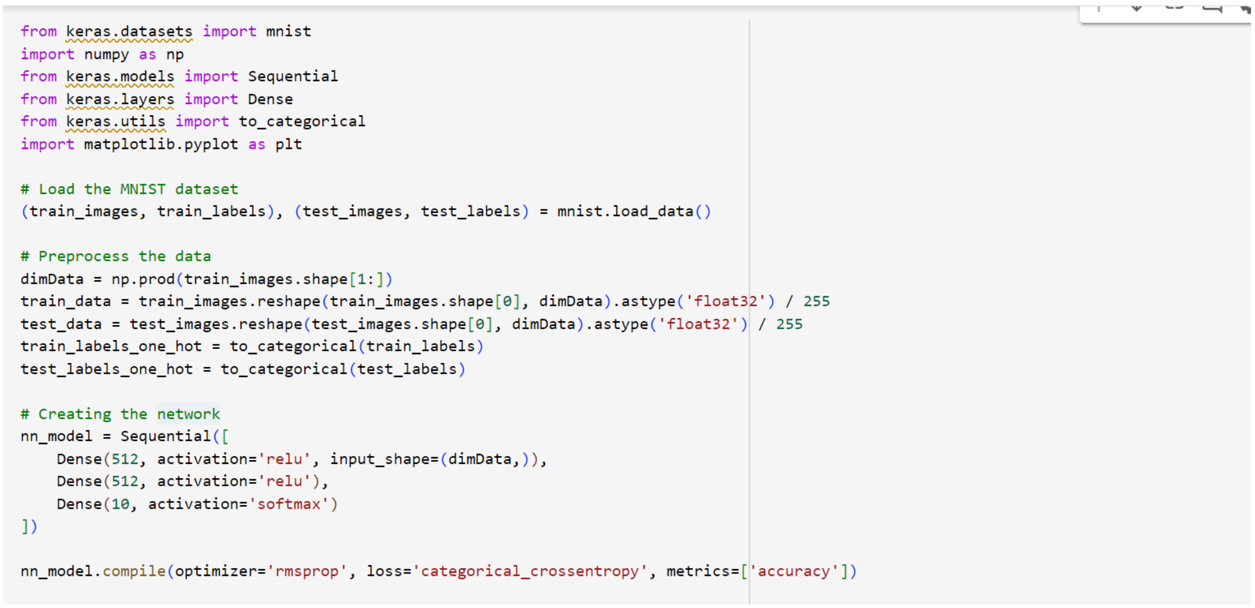
7. Train the Model:
   - The model is trained on the training data for 100 epochs with a batch size of 10. Training progress is shown with verbosity set to 1.

8. Print Model Summary:
   - The architecture and details of the model are printed, including the number of layers and parameters.

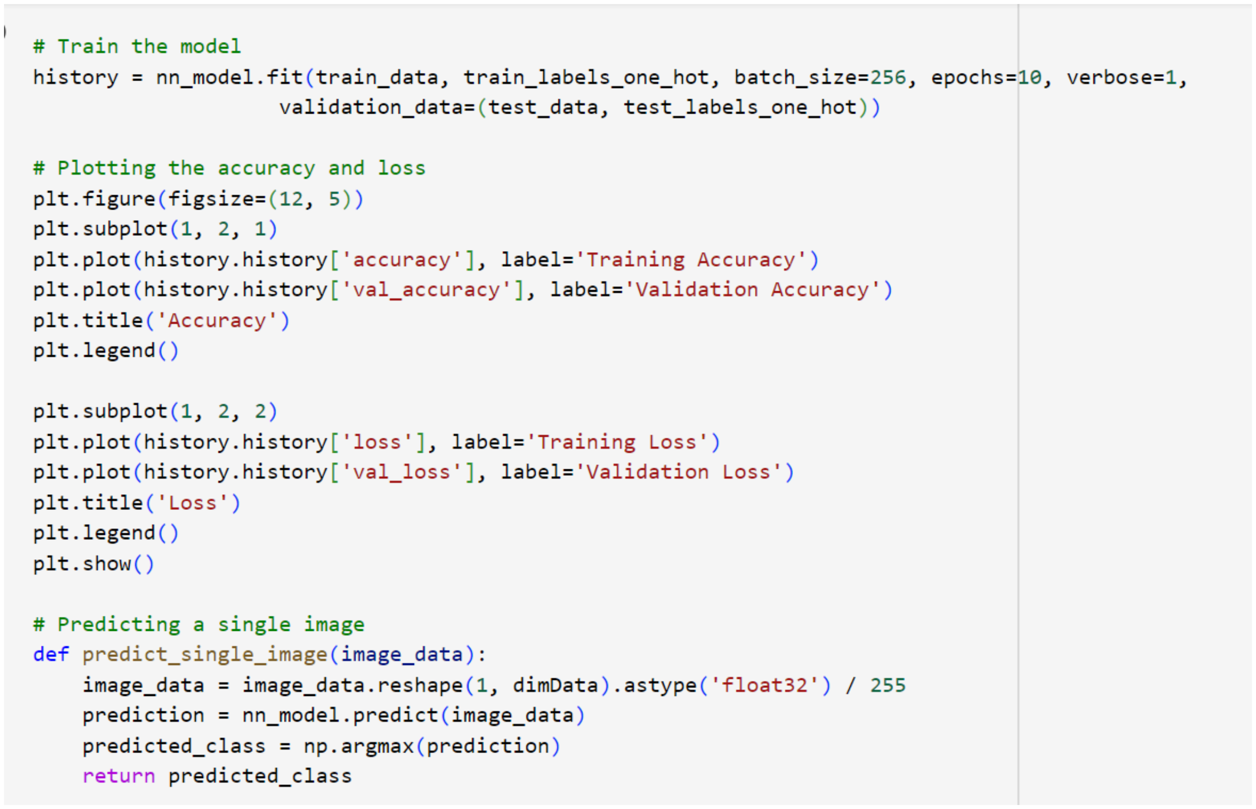
9. Evaluate the Model:
   - The model’s performance is assessed on the test data. The test accuracy is printed as a percentage.

Question 2

Code and Output:



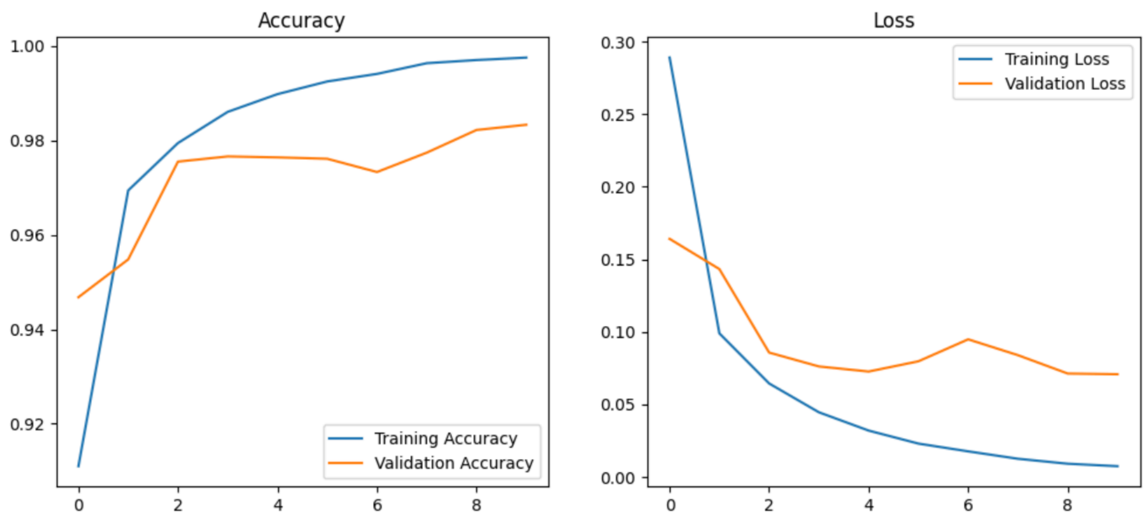
```python
from keras.datasets import mnist
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
import matplotlib.pyplot as plt

# 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
nn_model = Sequential([
    Dense(512, activation='relu', input_shape=(dimData,)),
    Dense(512, activation='relu'),
    Dense(10, activation='softmax')
])

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



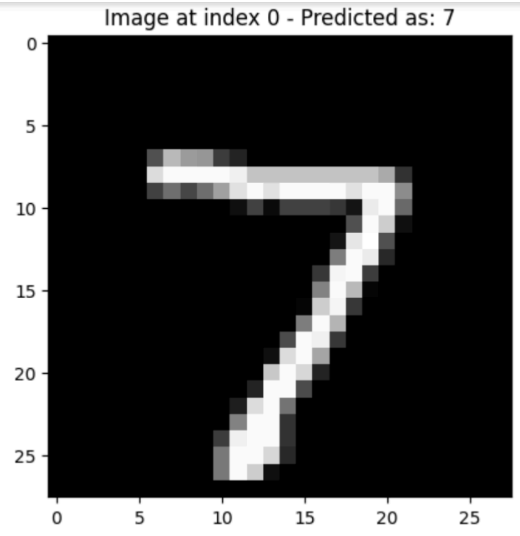
```python
# Train the model
history = nn_model.fit(train_data, train_labels_one_hot, batch_size=256, epochs=10, verbose=1,
                       validation_data=(test_data, test_labels_one_hot))

# 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()

# Predicting a single image
def predict_single_image(image_data):
    image_data = image_data.reshape(1, dimData).astype('float32') / 255
    prediction = nn_model.predict(image_data)
    predicted_class = np.argmax(prediction)
    return predicted_class
```

```python
# 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
nn_model_tanh = Sequential([
    Dense(512, activation='tanh', input_shape=(dimData,)),
    Dense(10, activation='softmax')
])

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

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Epoch 1/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 7s 26ms/step - accuracy: 0.8304 - loss: 0.5345 - val_accuracy: 0.9468 - val_loss: 0.1641
Epoch 2/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 8s 34ms/step - accuracy: 0.9679 - loss: 0.1060 - val_accuracy: 0.9548 - val_loss: 0.1433
Epoch 3/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 9s 30ms/step - accuracy: 0.9779 - loss: 0.0690 - val_accuracy: 0.9755 - val_loss: 0.0857
Epoch 4/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 27ms/step - accuracy: 0.9865 - loss: 0.0446 - val_accuracy: 0.9766 - val_loss: 0.0762
Epoch 5/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 11s 30ms/step - accuracy: 0.9911 - loss: 0.0294 - val_accuracy: 0.9764 - val_loss: 0.0727
Epoch 6/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 24ms/step - accuracy: 0.9933 - loss: 0.0211 - val_accuracy: 0.9761 - val_loss: 0.0797
Epoch 7/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 11s 25ms/step - accuracy: 0.9957 - loss: 0.0144 - val_accuracy: 0.9733 - val_loss: 0.0949
Epoch 8/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 11s 30ms/step - accuracy: 0.9965 - loss: 0.0118 - val_accuracy: 0.9774 - val_loss: 0.0839
Epoch 9/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 7s 29ms/step - accuracy: 0.9975 - loss: 0.0079 - val_accuracy: 0.9822 - val_loss: 0.0713
Epoch 10/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 10s 27ms/step - accuracy: 0.9976 - loss: 0.0073 - val_accuracy: 0.9833 - val_loss: 0.0708
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step
Predicted class for image at index 0: 7
```



```
Epoch 1/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step - accuracy: 0.8290 - loss: 0.5651 - val_accuracy: 0.9224 - val_loss: 0.2698
Epoch 2/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 15ms/step - accuracy: 0.9279 - loss: 0.2506 - val_accuracy: 0.9384 - val_loss: 0.2109
Epoch 3/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 18ms/step - accuracy: 0.9476 - loss: 0.1797 - val_accuracy: 0.9550 - val_loss: 0.1563
```

Explanation:

1. Import Libraries:
   - `mnist` from Keras is used to load the MNIST dataset.
   - `numpy` is used for numerical operations.
   - `Sequential` and `Dense` from Keras are used to build the neural network.
   - `to_categorical` from Keras converts labels to one-hot encoding.
   - `matplotlib.pyplot` is used for plotting.

2. Load the MNIST Dataset:
   - The MNIST dataset is loaded, providing training and testing images and their corresponding labels.

3. Preprocess the Data:
   - The training and testing images are reshaped from 3D arrays (28x28 pixels) to 1D arrays (784 pixels) and normalized to values between 0 and 1.
   - The labels are converted to one-hot encoded format to be used with the softmax output layer.

4. Create the Neural Network:
   - A Sequential model is defined with three Dense layers:
     - The first layer has 512 neurons with ReLU activation, taking the flattened image data as input.
     - The second layer also has 512 neurons with ReLU activation.

- The final layer has 10 neurons with a softmax activation function to produce probabilities for each of the 10 classes.

5. Compile the Model:
   - The model is compiled using the RMSprop optimizer and categorical crossentropy loss function, with accuracy as the evaluation metric.

6. Train the Model:
   - The model is trained on the training data with a batch size of 256 for 10 epochs.
   - Validation is done on the test data to monitor the performance.

7. Plot Training and Validation Metrics:
   - Accuracy and loss for both training and validation sets are plotted over epochs to visualize the training process and model performance.

8. Predict a Single Image:
   - A function is defined to preprocess a single image, predict its class using the trained model, and return the predicted class.
   - A specific image from the test set is chosen, and its predicted class is printed and displayed.

9. Modify and Train an Alternative Model:
   - An alternative neural network model is created with a single hidden layer of 512 neurons using the tanh activation function and a softmax output layer.
   - This alternative model is compiled and trained similarly to the first model for comparison.