ICP2_NNDL

Use Case Description:

Predicting the diabetes disease

Programming elements: Keras Basics

In class programming:

1. Use the use case in the class:

a. Add more Dense layers to the existing code and check how the accuracy changes.

\+ Code + Text

```
#1.Collecting the data
from google.colab import drive
drive.mount('/content/drive')
path_to_csv = '/content/drive/My Drive/Colab_Notebooks/Assignment2_NNDL/breastcancer.csv'
dataset = pd.read_csv('/content/drive/My Drive/Colab_Notebooks/Assignment2_NNDL/diabetes.csv')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

[4]

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Load breast cancer dataset
breast_cancer_data = pd.read_csv('breastcancer.csv')

# Separate features and target
X = breast_cancer_data.drop('diagnosis', axis=1)
y = breast_cancer_data['diagnosis']

# Convert categorical labels into numerical format using Label Encoding
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

# Normalize the data before feeding it to the model
sc = StandardScaler()
X_normalized = sc.fit_transform(X)
```

Github Link: https://github.com/DivyaReddy1999/NNDL_Assignment2.git

ICP2_NNDL

```python
# Normalize the data before feeding it to the model
sc = StandardScaler()
X_normalized = sc.fit_transform(X)

# Split the normalized data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_normalized, y, test_size=0.2, random_state=42)

# Create a Sequential model with two hidden layers and Relu activation
model = Sequential()
model.add(Dense(64, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(32, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

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

# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.1)

# Evaluate the model on the test set
_, accuracy = model.evaluate(X_test, y_test)
print("Accuracy on the test set:", accuracy)
```

Outputs:

```
Epoch 23/50
409/409 [==============================] - 0s 107us/sample - loss: nan - accuracy: 0.6210 - val_loss: nan - val_accuracy: 0.6957
Epoch 24/50
409/409 [==============================] - 0s 92us/sample - loss: nan - accuracy: 0.6210 - val_loss: nan - val_accuracy: 0.6957
Epoch 25/50
409/409 [==============================] - 0s 100us/sample - loss: nan - accuracy: 0.6210 - val_loss: nan - val_accuracy: 0.6957
Epoch 26/50
409/409 [==============================] - 0s 94us/sample - loss: nan - accuracy: 0.6210 - val_loss: nan - val_accuracy: 0.6957
Epoch 27/50
409/409 [==============================] - 0s 93us/sample - loss: nan - accuracy: 0.6210 - val_loss: nan - val_accuracy: 0.6957
Epoch 28/50
409/409 [==============================] - 0s 88us/sample - loss: nan - accuracy: 0.6210 - val_loss: nan - val_accuracy: 0.6957
Epoch 29/50
409/409 [==============================] - 0s 97us/sample - loss: nan - accuracy: 0.6210 - val_loss: nan - val_accuracy: 0.6957
Epoch 30/50
409/409 [==============================] - 0s 117us/sample - loss: nan - accuracy: 0.6210 - val_loss: nan - val_accuracy: 0.6957
Epoch 31/50
409/409 [==============================] - 0s 96us/sample - loss: nan - accuracy: 0.6210 - val_loss: nan - val_accuracy: 0.6957
Epoch 32/50
409/409 [==============================] - 0s 92us/sample - loss: nan - accuracy: 0.6210 - val_loss: nan - val_accuracy: 0.6957
Epoch 33/50
409/409 [==============================] - 0s 96us/sample - loss: nan - accuracy: 0.6210 - val_loss: nan - val_accuracy: 0.6957
Epoch 34/50
409/409 [==============================] - 0s 96us/sample - loss: nan - accuracy: 0.6210 - val_loss: nan - val_accuracy: 0.6957
Epoch 35/50
409/409 [==============================] - 0s 94us/sample - loss: nan - accuracy: 0.6210 - val_loss: nan - val_accuracy: 0.6957
Epoch 36/50
409/409 [==============================] - 0s 96us/sample - loss: nan - accuracy: 0.6210 - val_loss: nan - val_accuracy: 0.6957
Epoch 37/50
409/409 [==============================] - 0s 90us/sample - loss: nan - accuracy: 0.6210 - val_loss: nan - val_accuracy: 0.6957
Epoch 38/50
409/409 [==============================] - 0s 127us/sample - loss: nan - accuracy: 0.6210 - val_loss: nan - val_accuracy: 0.6957
Epoch 39/50
409/409 [==============================] - 0s 88us/sample - loss: nan - accuracy: 0.6210 - val_loss: nan - val_accuracy: 0.6957
```

```
Epoch 47/50
409/409 [==============================] - 0s 88us/sample - loss: nan - accuracy: 0.6210 - val_loss: nan - val_accuracy: 0.6957
Epoch 48/50
409/409 [==============================] - 0s 90us/sample - loss: nan - accuracy: 0.6210 - val_loss: nan - val_accuracy: 0.6957
Epoch 49/50
409/409 [==============================] - 0s 92us/sample - loss: nan - accuracy: 0.6210 - val_loss: nan - val_accuracy: 0.6957
Epoch 50/50
409/409 [==============================] - 0s 101us/sample - loss: nan - accuracy: 0.6210 - val_loss: nan - val_accuracy: 0.6957
Accuracy on the test set: 0.622807
```

Github Link: https://github.com/DivyaReddy1999/NNDL_Assignment2.git

ICP2_NNDL

2. Change the data source to Breast Cancer dataset * available in the source code folder and make required changes. Report accuracy of the model.

```
import pandas
from keras.models import Sequential
from keras.layers.core import Dense, Activation

# load dataset
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np

dataset = pd.read_csv('/content/drive/My Drive/Colab_Notebooks/Assignment2_NNDL/diabetes.csv')

# Separate features and target
X = dataset.iloc[:, 0:8].values
Y = dataset.iloc[:, 8].values

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=87)

np.random.seed(155)
my_first_nn = Sequential()  # create model
my_first_nn.add(Dense(20, input_dim=8, activation='relu'))  # hidden layer 1
my_first_nn.add(Dense(10, activation='relu'))  # hidden layer 2
my_first_nn.add(Dense(5, activation='relu'))   # hidden layer 3
my_first_nn.add(Dense(1, activation='sigmoid'))  # output layer

my_first_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
my_first_nn_fitted = my_first_nn.fit(X_train, Y_train, epochs=100, initial_epoch=0)

print(my_first_nn.summary())
print(my_first_nn.evaluate(X_test, Y_test))
```

Outputs:

Github Link: https://github.com/DivyaReddy1999/NNDL_Assignment2.git

```
Epoch 92/100
575/575 [==============================] - 0s 79us/sample - loss: 0.5350 - acc: 0.7635
Epoch 93/100
575/575 [==============================] - 0s 77us/sample - loss: 0.5386 - acc: 0.7461
Epoch 94/100
575/575 [==============================] - 0s 75us/sample - loss: 0.5363 - acc: 0.7461
Epoch 95/100
575/575 [==============================] - 0s 76us/sample - loss: 0.5332 - acc: 0.7530
Epoch 96/100
575/575 [==============================] - 0s 89us/sample - loss: 0.5348 - acc: 0.7409
Epoch 97/100
575/575 [==============================] - 0s 107us/sample - loss: 0.5369 - acc: 0.7513
Epoch 98/100
575/575 [==============================] - 0s 75us/sample - loss: 0.5444 - acc: 0.7374
Epoch 99/100
575/575 [==============================] - 0s 81us/sample - loss: 0.5386 - acc: 0.7322
Epoch 100/100
575/575 [==============================] - 0s 78us/sample - loss: 0.5402 - acc: 0.7287
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_12 (Dense)            (None, 20)                180

 dense_13 (Dense)            (None, 10)                210

 dense_14 (Dense)            (None, 5)                 55

 dense_15 (Dense)            (None, 1)                 6

=================================================================
Total params: 451
Trainable params: 451
Non-trainable params: 0
```

3. Normalize the data before feeding the data to the model and check how the normalization change your accuracy (code given below).

```python
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
import matplotlib.pyplot as plt

# load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# normalize pixel values to range [0, 1]
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255

# convert class labels to binary class matrices
num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# create a simple neural network model
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

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

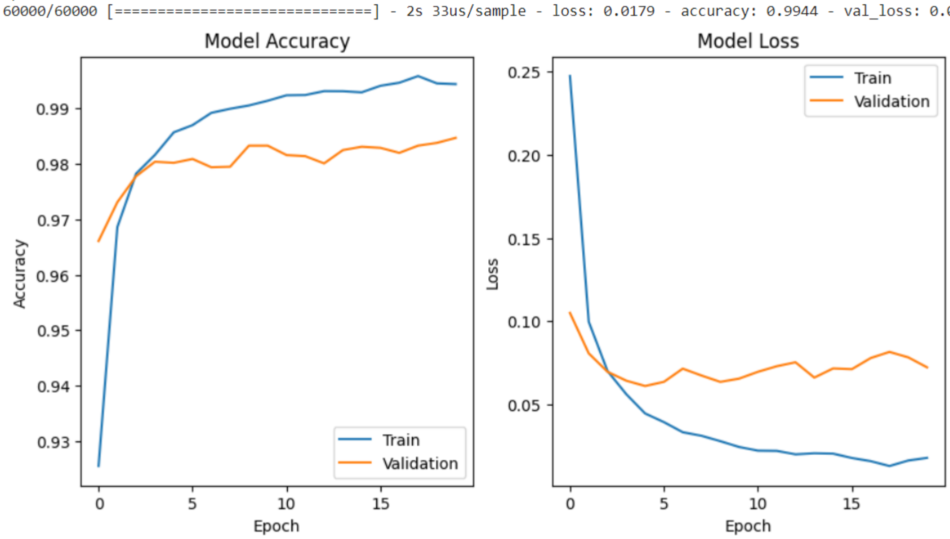
Github Link: https://github.com/DivyaReddy1999/NNDL_Assignment2.git

ICP2_NNDL

```python
# train the model and record the training history
history = model.fit(x_train.reshape(-1, 784), y_train, validation_data=(x_test.reshape(-1, 784), y_test),
                    epochs=20, batch_size=128)

# plot the training and validation accuracy and loss curves
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='lower right')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')

plt.show()
```

Outputs:

```
60000/60000 [==============================] - 2s 32us/sample - loss: 0.0163 - accuracy: 0.9945 - val_loss: 0.0783 - val_accuracy: 0.9838
Epoch 20/20
60000/60000 [==============================] - 2s 33us/sample - loss: 0.0179 - accuracy: 0.9944 - val_loss: 0.0723 - val_accuracy: 0.9847
```



Use Image Classification on the hand written digits data set (mnist)

1. Plot the loss and accuracy for both training data and validation data using the history object in the source code.

Github Link: https://github.com/DivyaReddy1999/NNDL_Assignment2.git

ICP2_NNDL

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

# Load the MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Preprocess the data and normalize the pixel values to [0, 1]
X_train = X_train.astype('float32') / 255
X_test = X_test.astype('float32') / 255

# Flatten the images from 28x28 to 784-dimensional vectors
X_train = X_train.reshape((len(X_train), 28 * 28))
X_test = X_test.reshape((len(X_test), 28 * 28))

# One-hot encode the target labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Create the model
model = Sequential()
model.add(Dense(128, activation='relu', input_shape=(784,)))
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))

# Compile the model
model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
```

Github Link: https://github.com/DivyaReddy1999/NNDL_Assignment2.git

ICP2_NNDL

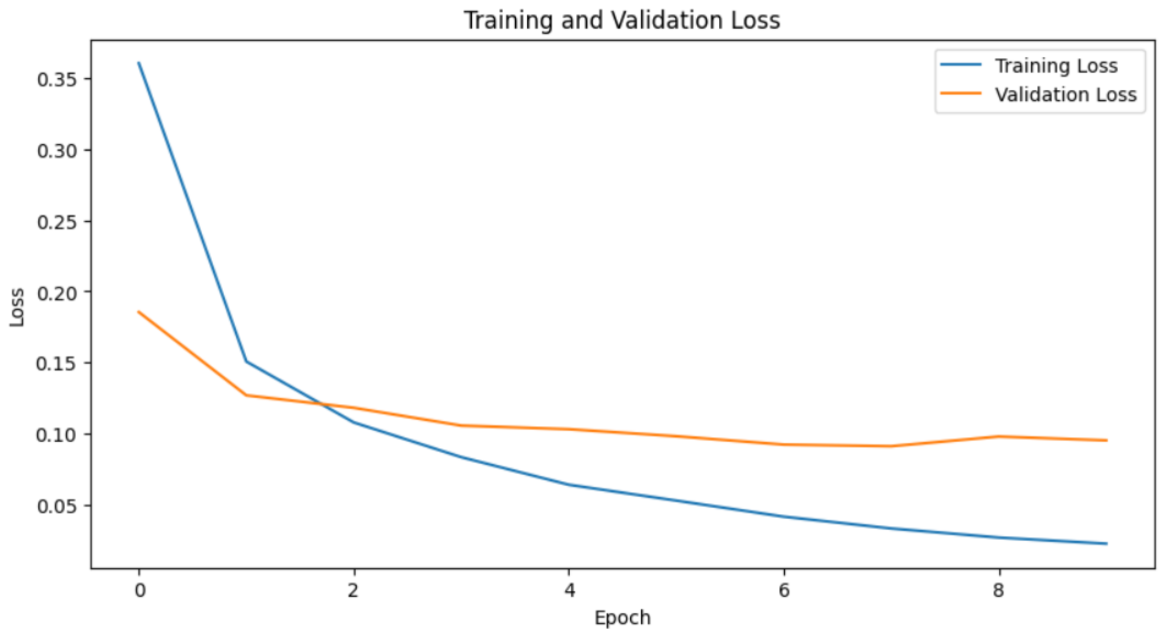
```python
# Train the model and store the training history
history = model.fit(X_train, y_train, epochs=10, batch_size=128, validation_split=0.2)

# Plot the training and validation loss
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')
plt.show()

# Plot the training and validation accuracy
plt.figure(figsize=(10, 5))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')
plt.show()
```

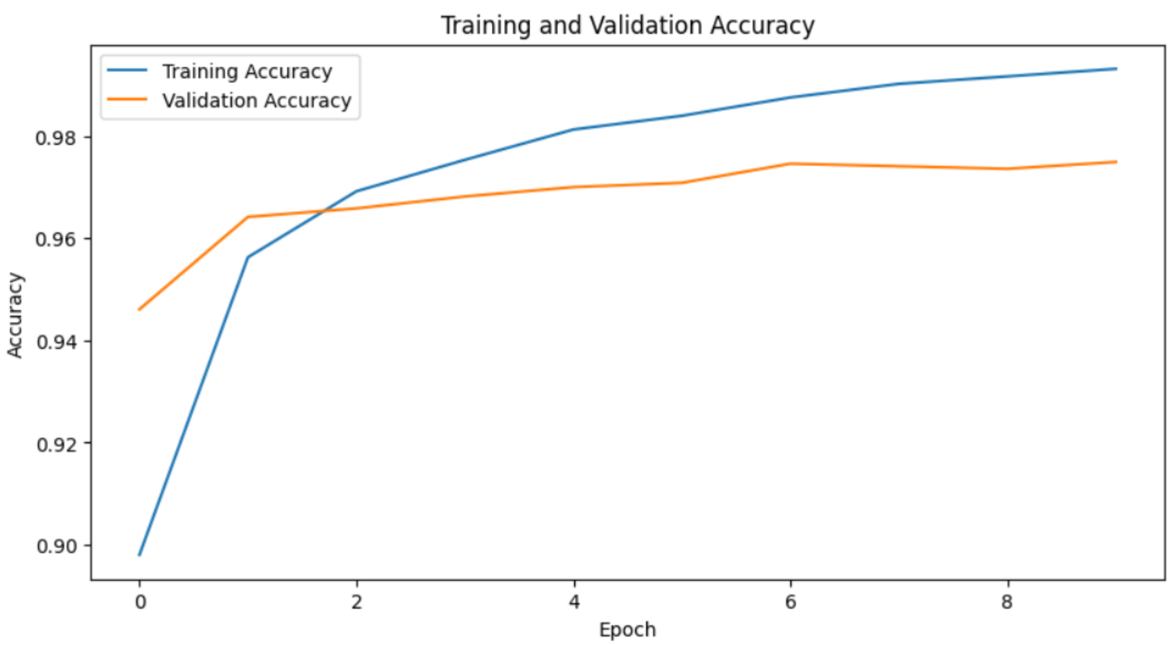
Outputs:

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/10
46976/48000 [============================>.] - ETA: 0s - loss: 0.3637 - accuracy: 0.8971/usr/local/lib/python3.10/dist-packages/keras/engine/train
  updates = self.state_updates
48000/48000 [==============================] - 2s 34us/sample - loss: 0.3602 - accuracy: 0.8980 - val_loss: 0.1855 - val_accuracy: 0.9461
Epoch 2/10
48000/48000 [==============================] - 1s 31us/sample - loss: 0.1508 - accuracy: 0.9563 - val_loss: 0.1270 - val_accuracy: 0.9642
Epoch 3/10
48000/48000 [==============================] - 1s 26us/sample - loss: 0.1080 - accuracy: 0.9692 - val_loss: 0.1183 - val_accuracy: 0.9658
Epoch 4/10
48000/48000 [==============================] - 1s 21us/sample - loss: 0.0836 - accuracy: 0.9753 - val_loss: 0.1058 - val_accuracy: 0.9682
Epoch 5/10
48000/48000 [==============================] - 1s 21us/sample - loss: 0.0643 - accuracy: 0.9813 - val_loss: 0.1033 - val_accuracy: 0.9700
Epoch 6/10
48000/48000 [==============================] - 1s 21us/sample - loss: 0.0531 - accuracy: 0.9840 - val_loss: 0.0983 - val_accuracy: 0.9708
Epoch 7/10
48000/48000 [==============================] - 1s 22us/sample - loss: 0.0419 - accuracy: 0.9875 - val_loss: 0.0925 - val_accuracy: 0.9746
Epoch 8/10
48000/48000 [==============================] - 1s 21us/sample - loss: 0.0336 - accuracy: 0.9902 - val_loss: 0.0913 - val_accuracy: 0.9741
Epoch 9/10
48000/48000 [==============================] - 1s 22us/sample - loss: 0.0272 - accuracy: 0.9917 - val_loss: 0.0981 - val_accuracy: 0.9736
Epoch 10/10
48000/48000 [==============================] - 1s 22us/sample - loss: 0.0229 - accuracy: 0.9931 - val_loss: 0.0954 - val_accuracy: 0.9749
```

Github Link: https://github.com/DivyaReddy1999/NNDL_Assignment2.git

ICP2_NNDL





2. Plot one of the images in the test data, and then do inferencing to check what is the prediction of the model on that single image.

Github Link: https://github.com/DivyaReddy1999/NNDL_Assignment2.git

ICP2_NNDL

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

# Load the MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Preprocess the data and normalize the pixel values to [0, 1]
X_train = X_train.astype('float32') / 255
X_test = X_test.astype('float32') / 255

# Flatten the images from 28x28 to 784-dimensional vectors
X_train = X_train.reshape((len(X_train), 28 * 28))
X_test = X_test.reshape((len(X_test), 28 * 28))

# One-hot encode the target labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Create the model
model = Sequential()
model.add(Dense(128, activation='relu', input_shape=(784,)))
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

Github Link: https://github.com/DivyaReddy1999/NNDL_Assignment2.git

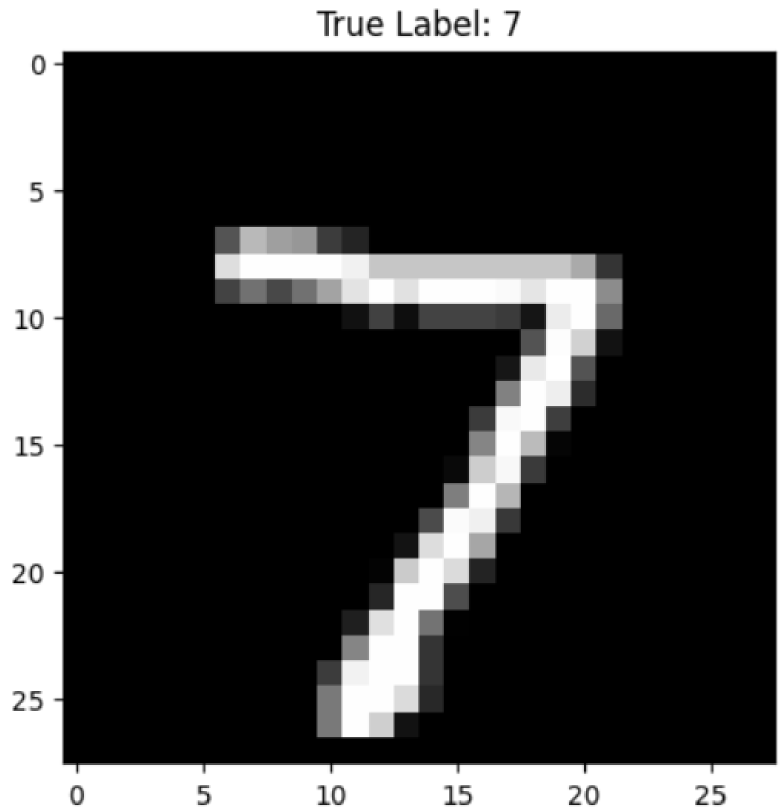
ICP2_NNDL

```python
# Compile the model
model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=128, validation_split=0.2)

# Plot one of the images from the test data
image_idx = 0  # Change this index to see different images
plt.imshow(X_test[image_idx].reshape(28, 28), cmap='gray')
plt.title(f"True Label: {np.argmax(y_test[image_idx])}")
plt.show()

# Make predictions on the single image
single_image = X_test[image_idx].reshape(1, 784)  # Reshape to 1x784 for prediction
prediction = model.predict(single_image)
predicted_label = np.argmax(prediction)

print(f"Predicted Label: {predicted_label}")
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/10
48000/48000 [==============================] - 1s 24us/sample - loss: 0.3751 - accuracy: 0.8949 - val_loss: 0.1789 - val_accuracy: 0.9506
Epoch 2/10
48000/48000 [==============================] - 1s 21us/sample - loss: 0.1557 - accuracy: 0.9543 - val_loss: 0.1359 - val_accuracy: 0.9603
Epoch 3/10
48000/48000 [==============================] - 1s 21us/sample - loss: 0.1130 - accuracy: 0.9662 - val_loss: 0.1209 - val_accuracy: 0.9655
Epoch 4/10
48000/48000 [==============================] - 1s 30us/sample - loss: 0.0861 - accuracy: 0.9747 - val_loss: 0.1142 - val_accuracy: 0.9679
Epoch 5/10
48000/48000 [==============================] - 1s 30us/sample - loss: 0.0693 - accuracy: 0.9793 - val_loss: 0.1043 - val_accuracy: 0.9705
Epoch 6/10
48000/48000 [==============================] - 1s 29us/sample - loss: 0.0554 - accuracy: 0.9839 - val_loss: 0.0939 - val_accuracy: 0.9729
Epoch 7/10
48000/48000 [==============================] - 1s 22us/sample - loss: 0.0462 - accuracy: 0.9864 - val_loss: 0.1060 - val_accuracy: 0.9716
Epoch 8/10
48000/48000 [==============================] - 1s 22us/sample - loss: 0.0401 - accuracy: 0.9878 - val_loss: 0.0908 - val_accuracy: 0.9743
Epoch 9/10
48000/48000 [==============================] - 1s 21us/sample - loss: 0.0289 - accuracy: 0.9916 - val_loss: 0.0918 - val_accuracy: 0.9752
Epoch 10/10
48000/48000 [==============================] - 1s 22us/sample - loss: 0.0238 - accuracy: 0.9929 - val_loss: 0.1039 - val_accuracy: 0.9734
```

Github Link: https://github.com/DivyaReddy1999/NNDL_Assignment2.git

ICP2_NNDL




```
Predicted Label: 7
```

3: We had used 2 hidden layers and Relu activation. Try to change the number of hidden layer and the activation to tanh or sigmoid and see what happens.

Github Link: https://github.com/DivyaReddy1999/NNDL_Assignment2.git

ICP2_NNDL

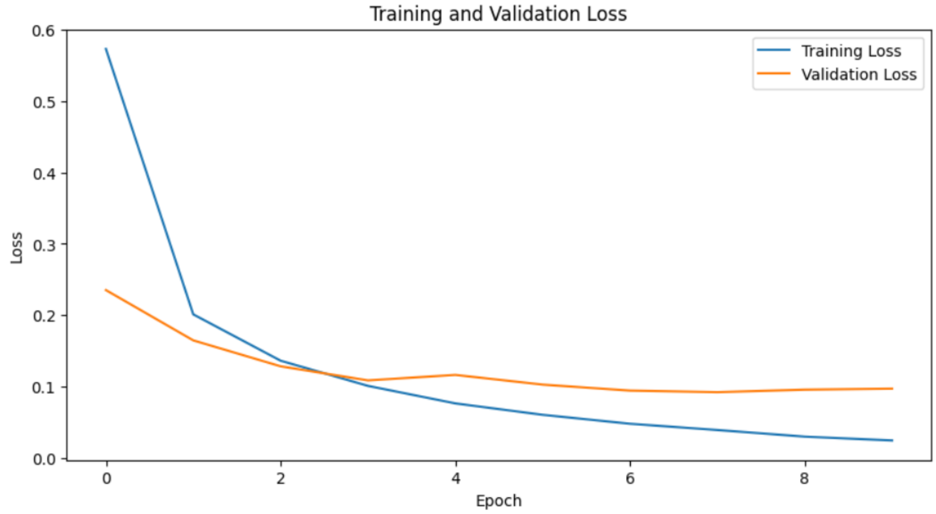
```python
# Load the MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Preprocess the data and normalize the pixel values to [0, 1]
X_train = X_train.astype('float32') / 255
X_test = X_test.astype('float32') / 255

# Flatten the images from 28x28 to 784-dimensional vectors
X_train = X_train.reshape((len(X_train), 28 * 28))
X_test = X_test.reshape((len(X_test), 28 * 28))

# One-hot encode the target labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Create a new model with 3 hidden layers and different activation functions
model = Sequential()
model.add(Dense(128, activation='tanh', input_shape=(784,)))  # First hidden layer with tanh activation
model.add(Dense(64, activation='sigmoid'))  # Second hidden layer with sigmoid activation
model.add(Dense(32, activation='relu'))  # Third hidden layer with relu activation
model.add(Dense(10, activation='softmax'))  # Output layer with softmax activation

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

# Train the model and store the training history
history = model.fit(X_train, y_train, epochs=10, batch_size=128, validation_split=0.2)
```

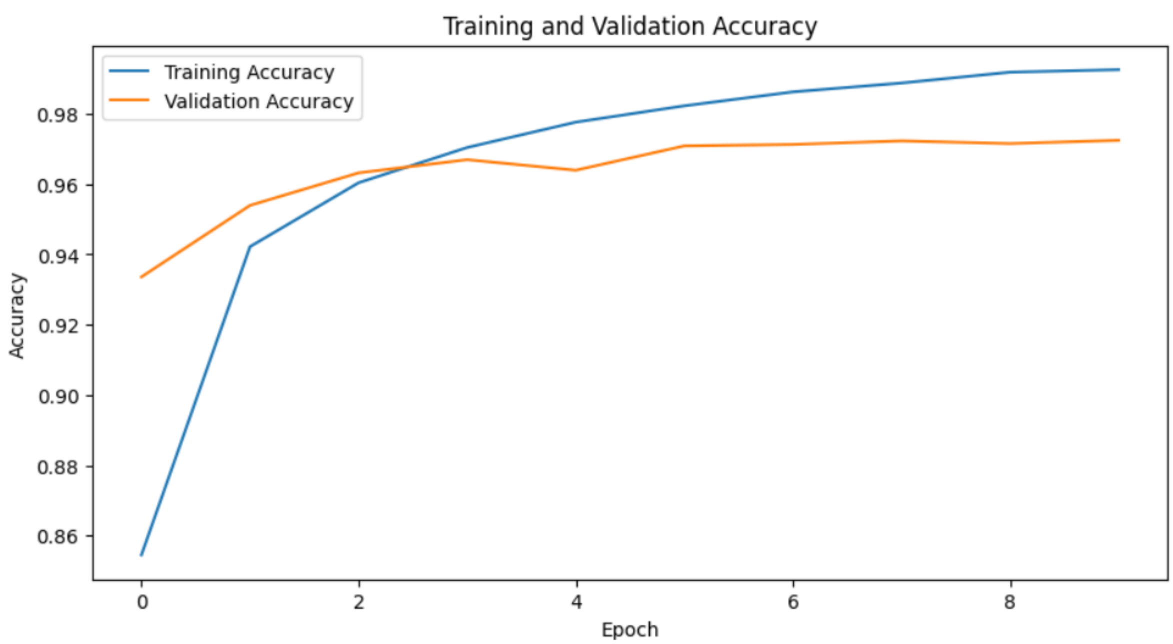
```python
# Compile the model
model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model and store the training history
history = model.fit(X_train, y_train, epochs=10, batch_size=128, validation_split=0.2)

# Plot the training and validation loss
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')
plt.show()

# Plot the training and validation accuracy
plt.figure(figsize=(10, 5))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')
plt.show()
```

Github Link: https://github.com/DivyaReddy1999/NNDL_Assignment2.git

ICP2_NNDL

```
48000/48000 [==============================] - 1s 24us/sample - loss: 0.0303 - accuracy: 0.9918 - val_loss: 0.0960 - val_accuracy: 0.9715
Epoch 10/10
48000/48000 [==============================] - 2s 31us/sample - loss: 0.0249 - accuracy: 0.9925 - val_loss: 0.0974 - val_accuracy: 0.9724
```





4. Run the same code without scaling the images and check the performance?

Github Link: https://github.com/DivyaReddy1999/NNDL_Assignment2.git

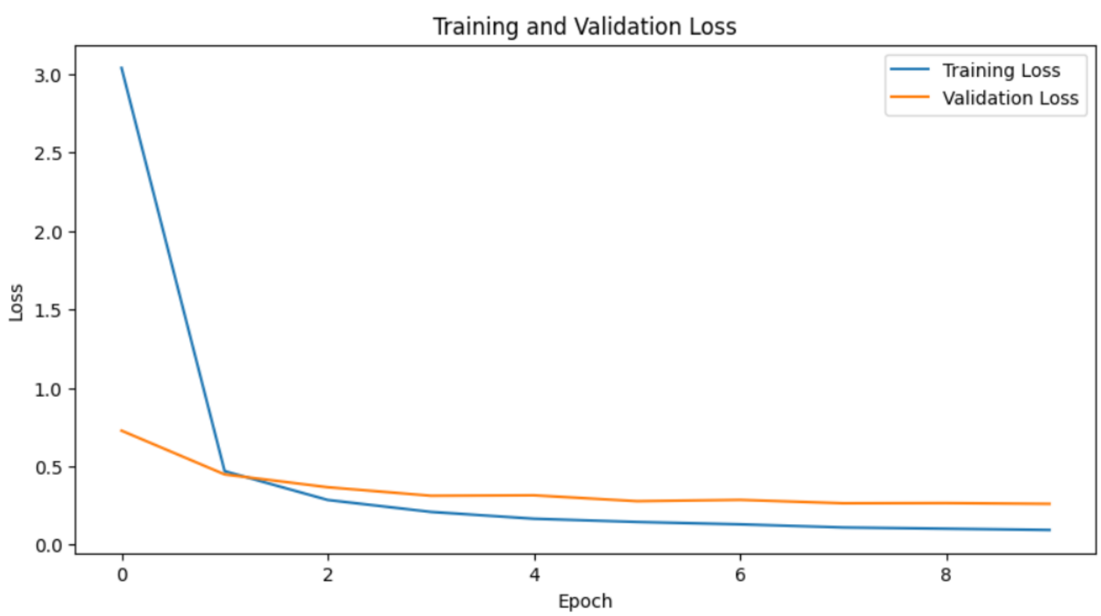
ICP2_NNDL

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

# Load the MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Flatten the images from 28x28 to 784-dimensional vectors
X_train = X_train.reshape((len(X_train), 28 * 28))
X_test = X_test.reshape((len(X_test), 28 * 28))

# One-hot encode the target labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Create the model
model = Sequential()
model.add(Dense(128, activation='relu', input_shape=(784,)))
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))

# Compile the model
model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
```

Github Link: https://github.com/DivyaReddy1999/NNDL_Assignment2.git

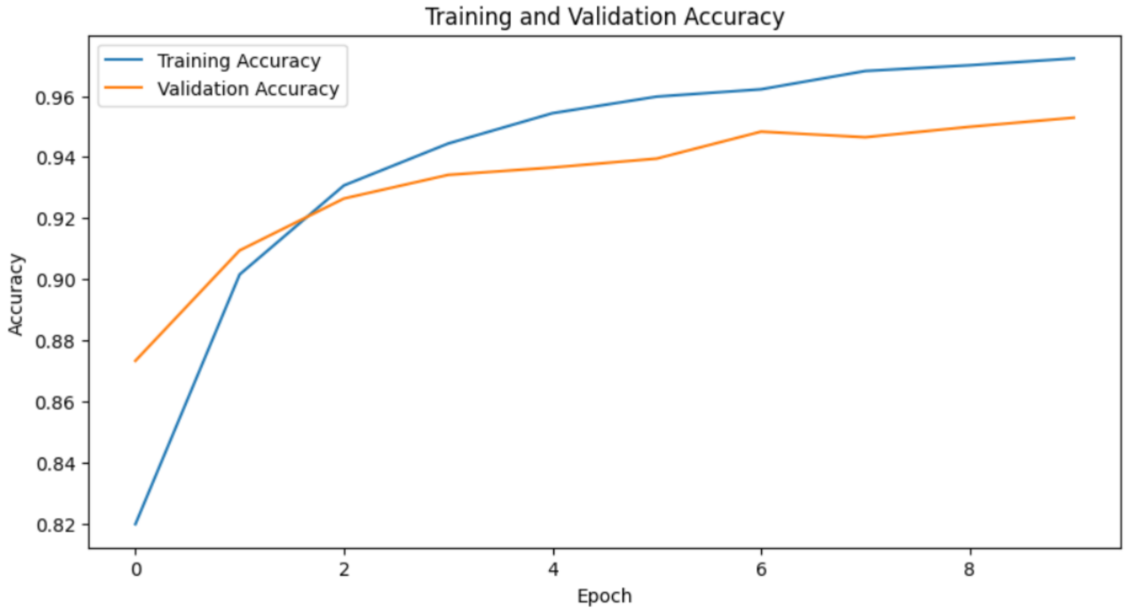
ICP2_NNDL

```
# Compile the model
model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model and store the training history
history = model.fit(X_train, y_train, epochs=10, batch_size=128, validation_split=0.2)

# Plot the training and validation loss
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')
plt.show()

# Plot the training and validation accuracy
plt.figure(figsize=(10, 5))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')
plt.show()
```



Github Link: https://github.com/DivyaReddy1999/NNDL_Assignment2.git

ICP2_NNDL



Github Link: https://github.com/DivyaReddy1999/NNDL_Assignment2.git