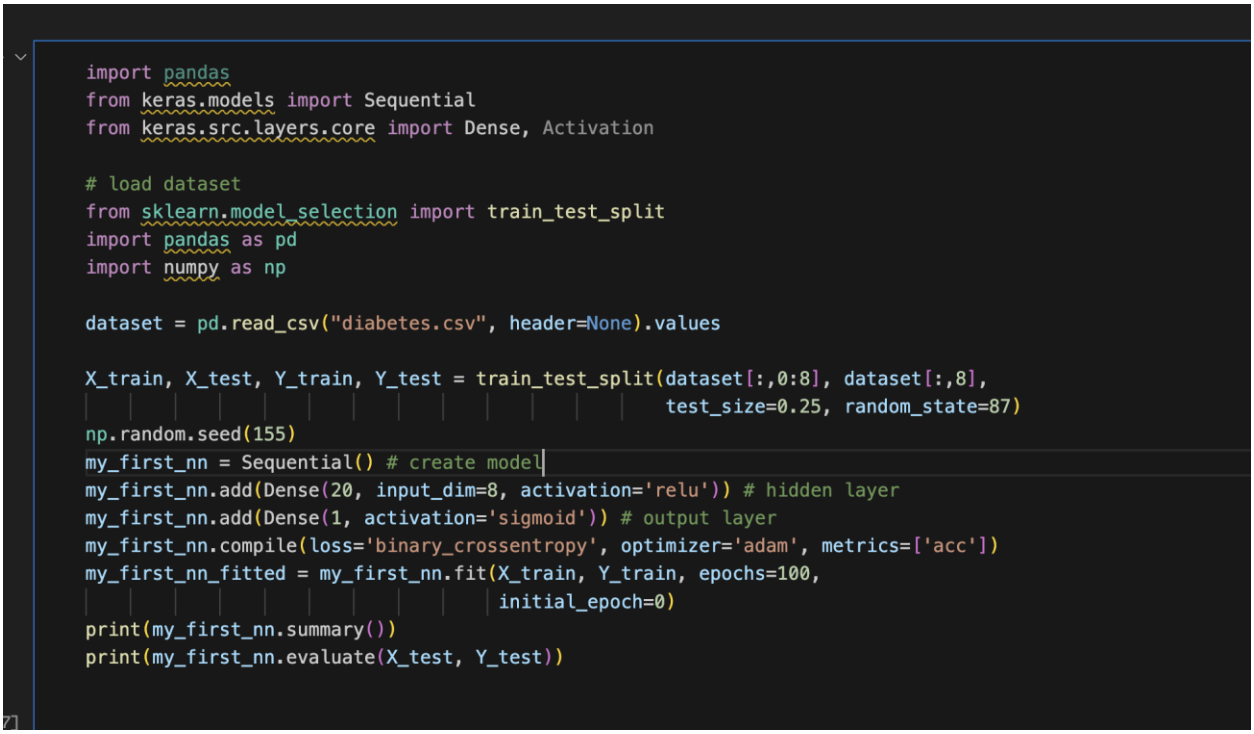
Neural Networks & Deep Learning: ICP6

Name: Lalitha Sowjanya Kamuju
ID: 700747213

1. Use the use case in the class: a. Add more Dense layers to the existing code and check how the accuracy changes.

2.



```
import pandas
from keras.models import Sequential
from keras.src.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("diabetes.csv", header=None).values

X_train, X_test, Y_train, Y_test = train_test_split(dataset[:,0:8], dataset[:,8],
                                                    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
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))
```

```
Epoch 1/100
18/18 [==============================] - 1s 2ms/step - loss: 17.2049 - acc: 0.3490
Epoch 2/100
18/18 [==============================] - 0s 2ms/step - loss: 7.0138 - acc: 0.4583
Epoch 3/100
18/18 [==============================] - 0s 2ms/step - loss: 4.8536 - acc: 0.4948
Epoch 4/100
18/18 [==============================] - 0s 2ms/step - loss: 3.7394 - acc: 0.5174
Epoch 5/100
18/18 [==============================] - 0s 2ms/step - loss: 3.2509 - acc: 0.5347
Epoch 6/100
18/18 [==============================] - 0s 2ms/step - loss: 2.8500 - acc: 0.5365
Epoch 7/100
18/18 [==============================] - 0s 2ms/step - loss: 2.3641 - acc: 0.5434
Epoch 8/100
18/18 [==============================] - 0s 2ms/step - loss: 1.9412 - acc: 0.5347
Epoch 9/100
18/18 [==============================] - 0s 2ms/step - loss: 1.6240 - acc: 0.5503
Epoch 10/100
18/18 [==============================] - 0s 2ms/step - loss: 1.3940 - acc: 0.5642
Epoch 11/100
18/18 [==============================] - 0s 2ms/step - loss: 1.2197 - acc: 0.5833
Epoch 12/100
18/18 [==============================] - 0s 2ms/step - loss: 1.1154 - acc: 0.5868
Epoch 13/100
...
_________________________________________________________________
None
6/6 [==============================] - 0s 4ms/step - loss: 0.6950 - acc: 0.6198
[0.695040762424469, 0.6197916865348816]
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

ADDING MORE DENSE LAYERS AND CHECKING THE OUTPUT

```
import pandas
from keras.models import Sequential
from keras.src.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("diabetes.csv", header=None).values

X_train, X_test, Y_train, Y_test = train_test_split(dataset[:,0:8], dataset[:,8],
                                                    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'))
my_first_nn.add(Dense(10, activation='relu'))  # Additional hidden layer
my_first_nn.add(Dense(5, activation='relu'))   # Additional hidden layer
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))
```

```
Epoch 1/100
18/18 [==============================] - 1s 2ms/step - loss: 5.0296 - acc: 0.6597
Epoch 2/100
18/18 [==============================] - 0s 2ms/step - loss: 1.7448 - acc: 0.4514
Epoch 3/100
18/18 [==============================] - 0s 2ms/step - loss: 0.9936 - acc: 0.3993
Epoch 4/100
18/18 [==============================] - 0s 2ms/step - loss: 0.7435 - acc: 0.4410
Epoch 5/100
18/18 [==============================] - 0s 2ms/step - loss: 0.7087 - acc: 0.6111
Epoch 6/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6921 - acc: 0.6302
Epoch 7/100
18/18 [==============================] - 0s 3ms/step - loss: 0.6782 - acc: 0.6337
Epoch 8/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6717 - acc: 0.6337
Epoch 9/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6608 - acc: 0.6424
Epoch 10/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6587 - acc: 0.6458
Epoch 11/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6559 - acc: 0.6580
Epoch 12/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6516 - acc: 0.6615
Epoch 13/100
```

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.src.layers.core import Dense, Activation

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

dataset = pd.read_csv("breastcancer.csv")
dataset.drop(['id'],axis=1,inplace = True)
del dataset['Unnamed: 32']

X = dataset.iloc[:, 2:].values
Y = dataset.iloc[:, 1].values
label = LabelEncoder()
Y = label.fit_transform(Y)

X_train, X_test, Y_train, Y_test = train_test_split(X,Y,
                                   test_size=0.25)
np.random.seed(155)
my_first_nn = Sequential() # create model
my_first_nn.add(Dense(20, input_dim=29, activation='relu')) # hidden layer
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))
```

```
Epoch 1/100
14/14 [==============================] - 1s 2ms/step - loss: -95749.1484 - acc: 0.0023
Epoch 2/100
14/14 [==============================] - 0s 2ms/step - loss: -130526.5000 - acc: 0.0023
Epoch 3/100
14/14 [==============================] - 0s 2ms/step - loss: -167976.8281 - acc: 0.0023
Epoch 4/100
14/14 [==============================] - 0s 3ms/step - loss: -209383.2500 - acc: 0.0023
Epoch 5/100
14/14 [==============================] - 0s 2ms/step - loss: -253128.2969 - acc: 0.0023
Epoch 6/100
14/14 [==============================] - 0s 2ms/step - loss: -299703.5625 - acc: 0.0023
Epoch 7/100
14/14 [==============================] - 0s 2ms/step - loss: -351225.3750 - acc: 0.0023
Epoch 8/100
14/14 [==============================] - 0s 2ms/step - loss: -407727.9688 - acc: 0.0023
Epoch 9/100
14/14 [==============================] - 0s 2ms/step - loss: -466729.3125 - acc: 0.0023
Epoch 10/100
14/14 [==============================] - 0s 2ms/step - loss: -533177.7500 - acc: 0.0023
Epoch 11/100
14/14 [==============================] - 0s 2ms/step - loss: -602043.6875 - acc: 0.0023
Epoch 12/100
14/14 [==============================] - 0s 2ms/step - loss: -676769.0625 - acc: 0.0023
Epoch 13/100
...
```

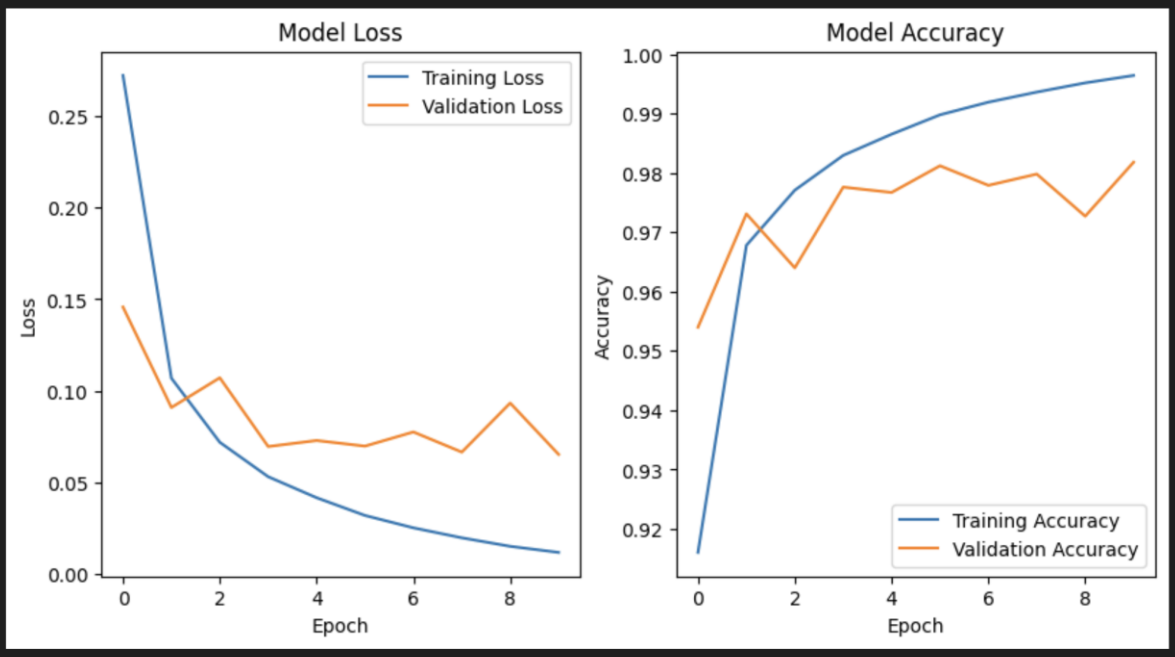
3. Normalize the data before feeding the data to the model and check how the normalization change your accuracy (code given below). from sklearn.preprocessing import StandardScaler sc = StandardScaler()

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

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

dataset = pd.read_csv("breastcancer.csv")
dataset.drop(['id'],axis=1,inplace = True)
del dataset['Unnamed: 32']

X = dataset.iloc[:, 2:].values
Y = dataset.iloc[:, 1].values
label = LabelEncoder()
Y = label.fit_transform(Y)

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

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

np.random.seed(155)
my_first_nn = Sequential() # create model
my_first_nn.add(Dense(20, input_dim=29, activation='relu')) # hidden layer
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))
```

```
Epoch 1/100
14/14 [==============================] - 3s 4ms/step - loss: -250.8924 - acc: 0.0023
Epoch 2/100
14/14 [==============================] - 0s 3ms/step - loss: -386.3432 - acc: 0.0023
Epoch 3/100
14/14 [==============================] - 0s 3ms/step - loss: -538.6116 - acc: 0.0023
Epoch 4/100
14/14 [==============================] - 0s 2ms/step - loss: -705.1451 - acc: 0.0023
Epoch 5/100
14/14 [==============================] - 0s 2ms/step - loss: -884.5459 - acc: 0.0023
Epoch 6/100
14/14 [==============================] - 0s 3ms/step - loss: -1086.6575 - acc: 0.0023
Epoch 7/100
14/14 [==============================] - 0s 2ms/step - loss: -1315.6779 - acc: 0.0023
Epoch 8/100
14/14 [==============================] - 0s 2ms/step - loss: -1562.3336 - acc: 0.0023
Epoch 9/100
14/14 [==============================] - 0s 2ms/step - loss: -1830.2288 - acc: 0.0023
Epoch 10/100
14/14 [==============================] - 0s 2ms/step - loss: -2125.2886 - acc: 0.0023
Epoch 11/100
14/14 [==============================] - 0s 2ms/step - loss: -2451.5872 - acc: 0.0023
Epoch 12/100
14/14 [==============================] - 0s 3ms/step - loss: -2795.5876 - acc: 0.0023
Epoch 13/100
...
_________________________________________________________________
None
5/5 [==============================] - 0s 4ms/step - loss: -141545.1250 - acc: 0.0000e+00
[-141545.125, 0.0]
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

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

(train_images,train_labels),(test_images, test_labels) = mnist.load_data()

print(train_images.shape[1:])
#process the data
#1. convert each image of shape 28*28 to 784 dimensional which will be fed to the network as a single feature
dimData = np.prod(train_images.shape[1:])
print(dimData)
train_data = train_images.reshape(train_images.shape[0],dimData)
test_data = test_images.reshape(test_images.shape[0],dimData)

#convert data to float and scale values between 0 and 1
train_data = train_data.astype('float')
test_data = test_data.astype('float')
#scale data
train_data /=255.0
test_data /=255.0
#change the labels frominteger to one-hot encoding. to_categorical is doing the same thing as LabelEncoder()
train_labels_one_hot = to_categorical(train_labels)
test_labels_one_hot = to_categorical(test_labels)

#creating network
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(dimData,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))

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



```
Epoch 1/10
235/235 [==============================] - 9s 33ms/step - loss: 0.2783 - accuracy: 0.9140 - val_loss: 0.1208 - val_accuracy: 0.9632
Epoch 2/10
235/235 [==============================] - 8s 35ms/step - loss: 0.1062 - accuracy: 0.9672 - val_loss: 0.0926 - val_accuracy: 0.9707
Epoch 3/10
235/235 [==============================] - 14s 58ms/step - loss: 0.0712 - accuracy: 0.9774 - val_loss: 0.0845 - val_accuracy: 0.9732
Epoch 4/10
235/235 [==============================] - 7s 31ms/step - loss: 0.0535 - accuracy: 0.9831 - val_loss: 0.0822 - val_accuracy: 0.9743
Epoch 5/10
235/235 [==============================] - 11s 47ms/step - loss: 0.0408 - accuracy: 0.9871 - val_loss: 0.0946 - val_accuracy: 0.9713
Epoch 6/10
235/235 [==============================] - 6s 26ms/step - loss: 0.0309 - accuracy: 0.9898 - val_loss: 0.0777 - val_accuracy: 0.9781
Epoch 7/10
235/235 [==============================] - 7s 31ms/step - loss: 0.0252 - accuracy: 0.9915 - val_loss: 0.0754 - val_accuracy: 0.9761
Epoch 8/10
235/235 [==============================] - 6s 26ms/step - loss: 0.0197 - accuracy: 0.9934 - val_loss: 0.0836 - val_accuracy: 0.9769
Epoch 9/10
235/235 [==============================] - 10s 43ms/step - loss: 0.0150 - accuracy: 0.9950 - val_loss: 0.0793 - val_accuracy: 0.9784
Epoch 10/10
235/235 [==============================] - 6s 24ms/step - loss: 0.0131 - accuracy: 0.9958 - val_loss: 0.0727 - val_accuracy: 0.9795
```

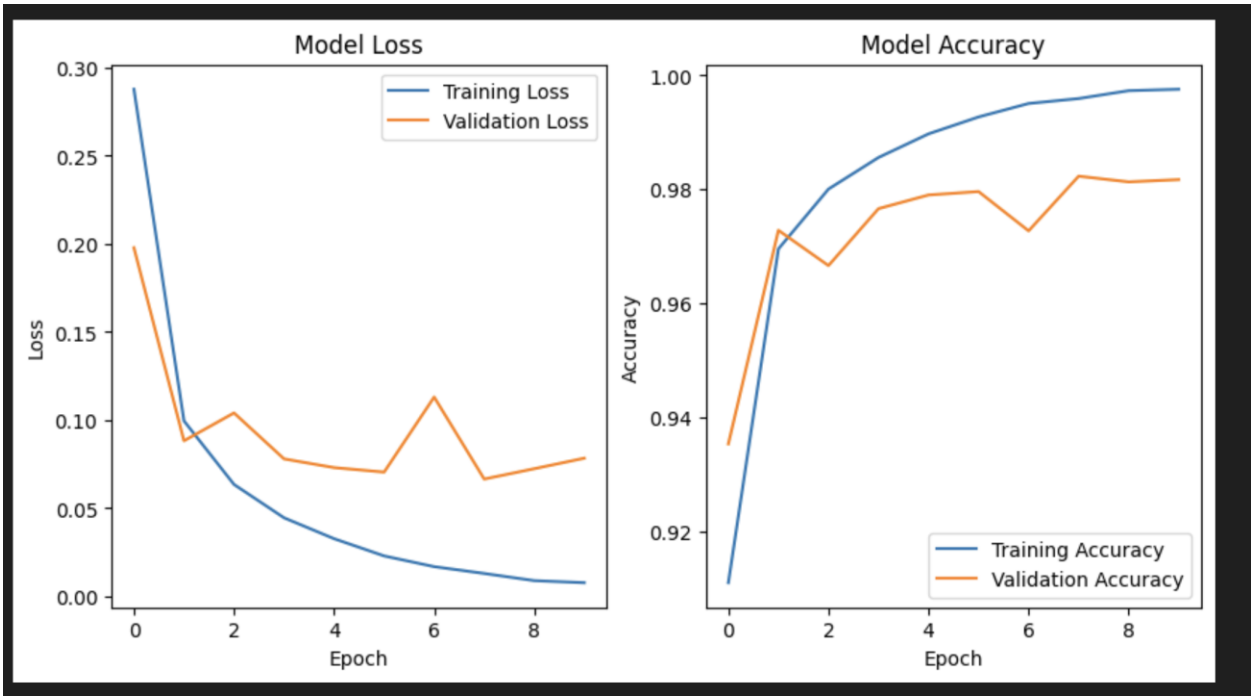
Plotting the loss and accuracy for both training data and validation data using the history object in the source code:

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

```
from keras import Sequential
from keras.datasets import mnist
import numpy as np
from keras.layers import Dense
from keras.utils import to_categorical

(train_images,train_labels),(test_images, test_labels) = mnist.load_data()

print(train_images.shape[1:])
#process the data
#1. convert each image of shape 28*28 to 784 dimensional which will be fed to the network as a single feature
dimData = np.prod(train_images.shape[1:])
print(dimData)
train_data = train_images.reshape(train_images.shape[0],dimData)
test_data = test_images.reshape(test_images.shape[0],dimData)

#convert data to float and scale values between 0 and 1
train_data = train_data.astype('float')
test_data = test_data.astype('float')
#scale data
train_data /=255.0
test_data /=255.0
#change the labels frominteger to one-hot encoding. to_categorical is doing the same thing as LabelEncoder()
train_labels_one_hot = to_categorical(train_labels)
test_labels_one_hot = to_categorical(test_labels)

#creating network
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(dimData,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))

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

```python
import matplotlib.pyplot as plt

# Plot training & validation loss values
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# Plot training & validation accuracy values
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()


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

```
(28, 28)
784
Epoch 1/10
235/235 [==============================] - 9s 35ms/step - loss: 0.2878 - accuracy: 0.9109 - val_loss: 0.1978 - val_accuracy: 0.9353
Epoch 2/10
235/235 [==============================] - 6s 26ms/step - loss: 0.0996 - accuracy: 0.9695 - val_loss: 0.0883 - val_accuracy: 0.9728
Epoch 3/10
235/235 [==============================] - 8s 33ms/step - loss: 0.0635 - accuracy: 0.9800 - val_loss: 0.1041 - val_accuracy: 0.9666
Epoch 4/10
235/235 [==============================] - 6s 28ms/step - loss: 0.0446 - accuracy: 0.9856 - val_loss: 0.0780 - val_accuracy: 0.9766
Epoch 5/10
235/235 [==============================] - 8s 33ms/step - loss: 0.0328 - accuracy: 0.9897 - val_loss: 0.0731 - val_accuracy: 0.9790
Epoch 6/10
235/235 [==============================] - 7s 28ms/step - loss: 0.0230 - accuracy: 0.9927 - val_loss: 0.0706 - val_accuracy: 0.9796
Epoch 7/10
235/235 [==============================] - 8s 33ms/step - loss: 0.0169 - accuracy: 0.9951 - val_loss: 0.1131 - val_accuracy: 0.9727
Epoch 8/10
235/235 [==============================] - 10s 41ms/step - loss: 0.0130 - accuracy: 0.9959 - val_loss: 0.0666 - val_accuracy: 0.9823
Epoch 9/10
235/235 [==============================] - 13s 55ms/step - loss: 0.0090 - accuracy: 0.9973 - val_loss: 0.0725 - val_accuracy: 0.9813
Epoch 10/10
235/235 [==============================] - 11s 45ms/step - loss: 0.0078 - accuracy: 0.9976 - val_loss: 0.0784 - val_accuracy: 0.9817
```

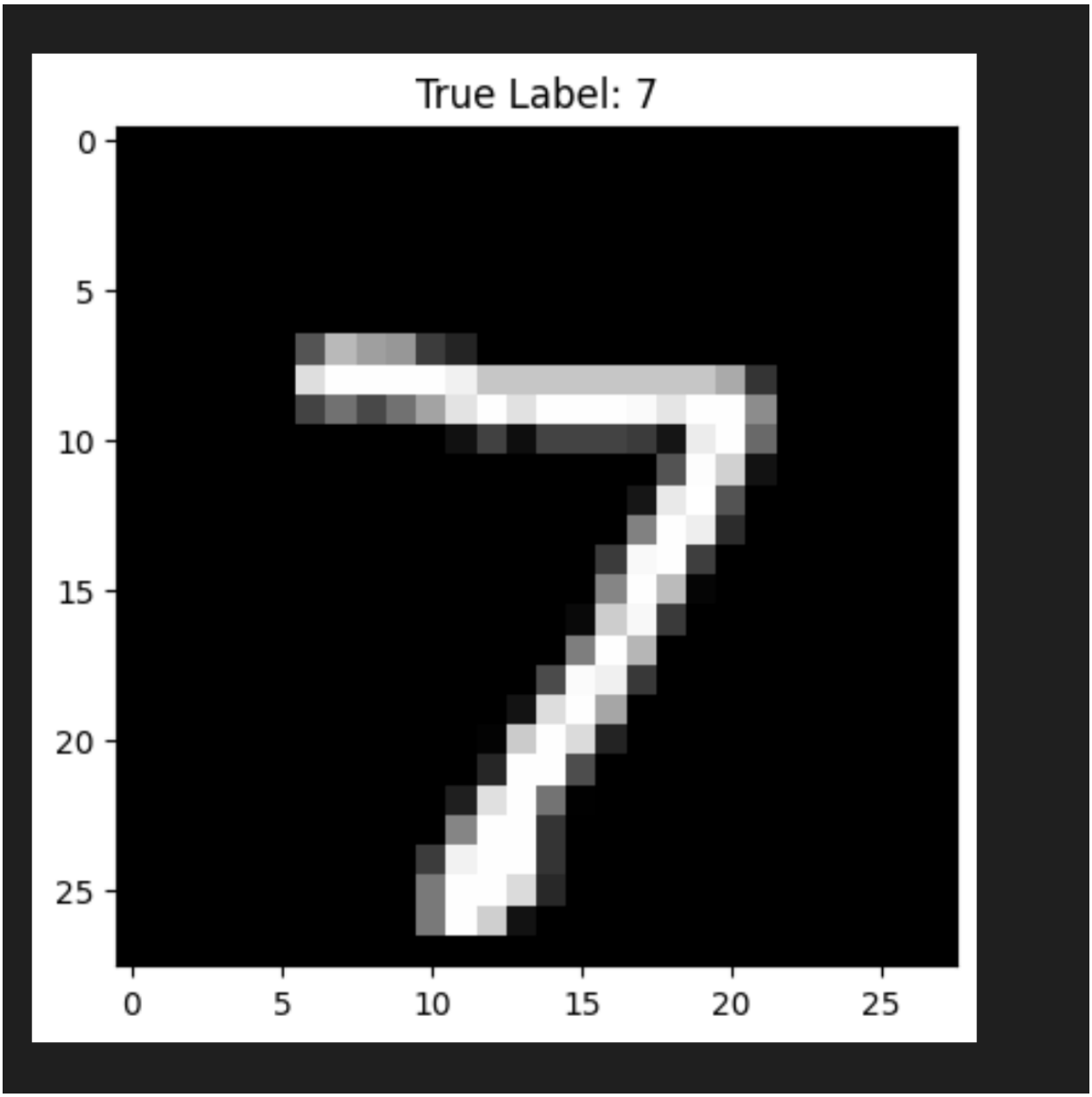


5. Plot one of the images in the test data, and then do inferencing to check what is the prediction of the model on that single image.

```
from keras import Sequential
from keras.datasets import mnist
import numpy as np
from keras.layers import Dense
from keras.utils import to_categorical

(train_images,train_labels),(test_images, test_labels) = mnist.load_data()

print(train_images.shape[1:])
#process the data
#1. convert each image of shape 28*28 to 784 dimensional which will be fed to the network as a single feature
dimData = np.prod(train_images.shape[1:])
print(dimData)
train_data = train_images.reshape(train_images.shape[0],dimData)
test_data = test_images.reshape(test_images.shape[0],dimData)

#convert data to float and scale values between 0 and 1
train_data = train_data.astype('float')
test_data = test_data.astype('float')
#scale data
train_data /=255.0
test_data /=255.0
#change the labels frominteger to one-hot encoding. to_categorical is doing the same thing as LabelEncoder()
train_labels_one_hot = to_categorical(train_labels)
test_labels_one_hot = to_categorical(test_labels)

#creating network
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(dimData,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))

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

```python
import matplotlib.pyplot as plt

# Plot one image from the test data
plt.figure()
plt.imshow(test_images[0], cmap='gray')
plt.title(f"True Label: {test_labels[0]}")

# Perform inference on the single image
image = test_data[0].reshape(1, dimData)
prediction = model.predict(image)
predicted_label = np.argmax(prediction)

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

```
(28, 28)
784
Epoch 1/10
235/235 [==============================] - 7s 29ms/step - loss: 0.2928 - accuracy: 0.9098 - val_loss: 0.1700 - val_accuracy: 0.9449
Epoch 2/10
235/235 [==============================] - 8s 33ms/step - loss: 0.1004 - accuracy: 0.9694 - val_loss: 0.1193 - val_accuracy: 0.9610
Epoch 3/10
235/235 [==============================] - 7s 28ms/step - loss: 0.0622 - accuracy: 0.9803 - val_loss: 0.0717 - val_accuracy: 0.9777
Epoch 4/10
235/235 [==============================] - 8s 33ms/step - loss: 0.0433 - accuracy: 0.9863 - val_loss: 0.0875 - val_accuracy: 0.9748
Epoch 5/10
235/235 [==============================] - 7s 31ms/step - loss: 0.0312 - accuracy: 0.9898 - val_loss: 0.0852 - val_accuracy: 0.9756
Epoch 6/10
235/235 [==============================] - 11s 46ms/step - loss: 0.0220 - accuracy: 0.9930 - val_loss: 0.0677 - val_accuracy: 0.9814
Epoch 7/10
235/235 [==============================] - 10s 44ms/step - loss: 0.0169 - accuracy: 0.9949 - val_loss: 0.0874 - val_accuracy: 0.9775
Epoch 8/10
235/235 [==============================] - 9s 38ms/step - loss: 0.0133 - accuracy: 0.9958 - val_loss: 0.0705 - val_accuracy: 0.9820
Epoch 9/10
235/235 [==============================] - 7s 30ms/step - loss: 0.0097 - accuracy: 0.9972 - val_loss: 0.0684 - val_accuracy: 0.9828
Epoch 10/10
235/235 [==============================] - 8s 33ms/step - loss: 0.0074 - accuracy: 0.9977 - val_loss: 0.0778 - val_accuracy: 0.9794
```



6. 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

```
from keras import Sequential
from keras.datasets import mnist
import numpy as np
from keras.layers import Dense
from keras.utils import to_categorical

(train_images,train_labels),(test_images, test_labels) = mnist.load_data()

print(train_images.shape[1:])
#process the data
#1. convert each image of shape 28*28 to 784 dimensional which will be fed to the network as a single feature
dimData = np.prod(train_images.shape[1:])
print(dimData)
train_data = train_images.reshape(train_images.shape[0],dimData)
test_data = test_images.reshape(test_images.shape[0],dimData)

#convert data to float and scale values between 0 and 1
train_data = train_data.astype('float')
test_data = test_data.astype('float')
#scale data
train_data /=255.0
test_data /=255.0
#change the labels frominteger to one-hot encoding. to_categorical is doing the same thing as LabelEncoder()
train_labels_one_hot = to_categorical(train_labels)
test_labels_one_hot = to_categorical(test_labels)

#creating network
model = Sequential()
model.add(Dense(512, activation='tanh', input_shape=(dimData,)))  # Change activation to 'tanh'
model.add(Dense(256, activation='tanh'))  # Add another hidden layer with 'tanh'
model.add(Dense(128, activation='tanh'))  # Add one more hidden layer with 'tanh'
model.add(Dense(10, activation='softmax'))

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

```
(28, 28)
784
Epoch 1/10
235/235 [==============================] - 7s 28ms/step - loss: 0.3479 - accuracy: 0.8952 - val_loss: 0.1964 - val_accuracy: 0.9414
Epoch 2/10
235/235 [==============================] - 11s 49ms/step - loss: 0.1493 - accuracy: 0.9557 - val_loss: 0.1710 - val_accuracy: 0.9439
Epoch 3/10
235/235 [==============================] - 7s 29ms/step - loss: 0.0986 - accuracy: 0.9700 - val_loss: 0.1051 - val_accuracy: 0.9671
Epoch 4/10
235/235 [==============================] - 6s 24ms/step - loss: 0.0723 - accuracy: 0.9773 - val_loss: 0.0997 - val_accuracy: 0.9690
Epoch 5/10
235/235 [==============================] - 7s 28ms/step - loss: 0.0513 - accuracy: 0.9839 - val_loss: 0.1598 - val_accuracy: 0.9478
Epoch 6/10
235/235 [==============================] - 6s 24ms/step - loss: 0.0399 - accuracy: 0.9879 - val_loss: 0.0734 - val_accuracy: 0.9768
Epoch 7/10
235/235 [==============================] - 7s 28ms/step - loss: 0.0290 - accuracy: 0.9918 - val_loss: 0.0848 - val_accuracy: 0.9735
Epoch 8/10
235/235 [==============================] - 5s 23ms/step - loss: 0.0227 - accuracy: 0.9932 - val_loss: 0.0920 - val_accuracy: 0.9725
Epoch 9/10
235/235 [==============================] - 7s 28ms/step - loss: 0.0171 - accuracy: 0.9950 - val_loss: 0.0671 - val_accuracy: 0.9806
Epoch 10/10
235/235 [==============================] - 5s 23ms/step - loss: 0.0120 - accuracy: 0.9966 - val_loss: 0.0889 - val_accuracy: 0.9765
```

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

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

(train_images,train_labels),(test_images, test_labels) = mnist.load_data()

print(train_images.shape[1:])
#process the data
#1. convert each image of shape 28*28 to 784 dimensional which will be fed to the network as a single feature
dimData = np.prod(train_images.shape[1:])
print(dimData)
train_data = train_images.reshape(train_images.shape[0],dimData)
test_data = test_images.reshape(test_images.shape[0],dimData)

#convert data to float and scale values between 0 and 1
train_data = train_data.astype('float')
test_data = test_data.astype('float')
#scale data
#train_data /=255.0
#test_data /=255.0
#change the labels frominteger to one-hot encoding. to_categorical is doing the same thing as LabelEncoder()
train_labels_one_hot = to_categorical(train_labels)
test_labels_one_hot = to_categorical(test_labels)

#creating network
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(dimData,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))

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

```
(28, 28)
784
Epoch 1/10
235/235 [==============================] - 10s 40ms/step - loss: 5.6614 - accuracy: 0.8777 - val_loss: 0.7007 - val_accuracy: 0.9321
Epoch 2/10
235/235 [==============================] - 8s 34ms/step - loss: 0.4260 - accuracy: 0.9466 - val_loss: 0.4194 - val_accuracy: 0.9417
Epoch 3/10
235/235 [==============================] - 9s 36ms/step - loss: 0.2534 - accuracy: 0.9596 - val_loss: 0.3492 - val_accuracy: 0.9488
Epoch 4/10
235/235 [==============================] - 9s 37ms/step - loss: 0.2018 - accuracy: 0.9676 - val_loss: 0.5652 - val_accuracy: 0.9292
Epoch 5/10
235/235 [==============================] - 9s 36ms/step - loss: 0.1605 - accuracy: 0.9740 - val_loss: 0.2677 - val_accuracy: 0.9634
Epoch 6/10
235/235 [==============================] - 7s 32ms/step - loss: 0.1411 - accuracy: 0.9765 - val_loss: 0.3155 - val_accuracy: 0.9633
Epoch 7/10
235/235 [==============================] - 7s 29ms/step - loss: 0.1408 - accuracy: 0.9785 - val_loss: 0.2825 - val_accuracy: 0.9697
Epoch 8/10
235/235 [==============================] - 7s 31ms/step - loss: 0.1209 - accuracy: 0.9826 - val_loss: 0.3198 - val_accuracy: 0.9706
Epoch 9/10
235/235 [==============================] - 7s 30ms/step - loss: 0.1142 - accuracy: 0.9835 - val_loss: 0.3881 - val_accuracy: 0.9673
Epoch 10/10
235/235 [==============================] - 7s 28ms/step - loss: 0.1169 - accuracy: 0.9851 - val_loss: 0.3564 - val_accuracy: 0.9695
```

GitHub Link : https://github.com/sowjanya-kamuju/Assignment6

Video Link :
https://vimeo.com/manage/videos/915443585/07e5722c79?studio_recording=true&record_session_id=2446b8b8-220b-487c-8d48-bfc7f09b0dbb