ASSIGNMENT-6
Name-Chanamolu Pranavya
ID- 700739974
NEURAL NETWORKS & DEEP LEARNING

GitHub link: https://github.com/pxc99740/neural_networks_assignment_6.git

Videolink: https://drive.google.com/drive/folders/131fLkQhJzAgL3m7DyX1f-F-KUG8CaJac

Use Case Description:
Predicting the diabetes disease :

The code is carrying out a basic neural network utilizing the Keras Programming interface. The dataset is stacked utilizing pandas from a CSV document and split into preparing and testing sets utilizing the train_test_split function from scikit-learn.

The neural network has one hidden layer with 20 nodes, an input layer with 8 nodes (relating to the 8 features in the dataset), and a result layer with a single node. The activation function utilized in the hidden layer is ReLU, and the activation function utilized in the hidden layer is sigmoid.

The neural network is compiled using the binary_crossentropy loss function, Adam optimizer, and accuracy as the evaluation metric. The model is then trained on the training set for 100 epochs using the fit method, and the summary of the model is printed using the summary method. Finally, the accuracy of the model is evaluated on the testing set using the evaluate method, and the loss and accuracy are printed.

```
from google.colab import drive
drive.mount('/content/gdrive')
```

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

```
[43] path_to_csv = '/content/gdrive/My Drive/diabetes.csv'
```

1. Using the use case in class

```
[59] import keras
     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(path_to_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
```

```
[59]
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 80/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5609 - acc: 0.7205
Epoch 81/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5643 - acc: 0.7344
Epoch 82/100
18/18 [==============================] - 0s 4ms/step - loss: 0.5620 - acc: 0.7326
Epoch 83/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5509 - acc: 0.7344
Epoch 84/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5660 - acc: 0.7396
Epoch 85/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5671 - acc: 0.7413
Epoch 86/100
18/18 [==============================] - 0s 3ms/step - loss: 0.6017 - acc: 0.6927
Epoch 87/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5886 - acc: 0.7309
Epoch 88/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5635 - acc: 0.7205
Epoch 89/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5649 - acc: 0.7274
Epoch 90/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5483 - acc: 0.7292
Epoch 91/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6072 - acc: 0.7170
Epoch 92/100
```

```
Epoch 93/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5594 - acc: 0.7205
Epoch 94/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5462 - acc: 0.7396
Epoch 95/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5457 - acc: 0.7517
Epoch 96/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5625 - acc: 0.7153
Epoch 97/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5590 - acc: 0.7274
Epoch 98/100
18/18 [==============================] - 0s 4ms/step - loss: 0.5568 - acc: 0.7240
Epoch 99/100
18/18 [==============================] - 0s 4ms/step - loss: 0.5519 - acc: 0.7431
Epoch 100/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5536 - acc: 0.7257
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_26 (Dense)            (None, 20)                180

 dense_27 (Dense)            (None, 1)                 21

=================================================================
Total params: 201
Trainable params: 201
Non-trainable params: 0
_________________________________________________________________
None
6/6 [==============================] - 0s 3ms/step - loss: 0.6981 - acc: 0.6302
[0.6981412768363953, 0.6302083134651184]
```

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

I have now added more dense layers to the code. The new addition to the code is the addition of two new hidden layers with 20 neurons each. The model now has three hidden layers, with the input layer having 8 input features and the output layer having a sigmoid activation function.

The model is compiled using binary cross-entropy loss and the Adam optimizer. The metric used to evaluate the model performance is accuracy.

The training is done using the fit() method, with 100 epochs and verbose set to 0 to suppress the output during training. The evaluation of the model is done using the evaluate() method on the test set.

The summary of the model is printed using the summary() method, which shows the number of trainable parameters and the layer-wise architecture of the model. Finally, the accuracy and loss of the model are printed after evaluation.

The accuracy improved from the first model to the second model. The first model had only one hidden layer with 20 neurons, while the second model had three hidden layers, each with 20 neurons. This allowed the second model to learn more complex features in the data, which resulted in a higher accuracy. Additionally, the second model had more parameters to optimize during training, which likely contributed to the improved accuracy.

Output:

```
my_first_nn = Sequential() # create model
my_first_nn.add(Dense(20, input_dim=8, activation='relu')) # hidden layer with input
my_first_nn.add(Dense(20, activation='relu')) # hidden layer
my_first_nn.add(Dense(20, 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']) # compilation
my_first_nn_fitted = my_first_nn.fit(X_train, Y_train, epochs=100,verbose=0, initial_epoch=0) # Training
print(my_first_nn.summary()) #Summary
print(my_first_nn.evaluate(X_test, Y_test)) #Evaluating
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_28 (Dense)            (None, 20)                180

 dense_29 (Dense)            (None, 20)                420

 dense_30 (Dense)            (None, 20)                420

 dense_31 (Dense)            (None, 1)                 21

=================================================================
Total params: 1,041
Trainable params: 1,041
Non-trainable params: 0
_________________________________________________________________
None
6/6 [==============================] - 0s 3ms/step - loss: 0.5584 - acc: 0.6979
[0.5583901405334473, 0.6979166865348816]
```

Finally, the summary of the model is printed using the summary method, and the accuracy of the model is evaluated on the test data using evaluate method, and the result is printed.

we can see that the accuracy decreased from model 2 to model 3. Model 2 had an accuracy of 0.6979, while model 3 had an accuracy of 0.6823. This could be due to the increased complexity of the model, as model 3 has more parameters (1219) compared to model 2 (1041). It's possible that model 3 is overfitting the training data, resulting in a lower accuracy on the test set.

```
my_first_nn = Sequential() # create model
my_first_nn.add(Dense(20, input_dim=8, activation='relu')) # hidden layer with input
my_first_nn.add(Dense(22, activation='relu')) # hidden layer
my_first_nn.add(Dense(24, 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']) # compilation
my_first_nn_fitted = my_first_nn.fit(X_train, Y_train, epochs=100,verbose=0, initial_epoch=0) # Training
print(my_first_nn.summary()) #Summary
print(my_first_nn.evaluate(X_test, Y_test)) #Evaluating
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_32 (Dense)            (None, 20)                180

 dense_33 (Dense)            (None, 22)                462

 dense_34 (Dense)            (None, 24)                552

 dense_35 (Dense)            (None, 1)                 25

=================================================================
Total params: 1,219
Trainable params: 1,219
Non-trainable params: 0
_________________________________________________________________
None
6/6 [==============================] - 0s 3ms/step - loss: 0.6171 - acc: 0.6823
[0.6171173453330994, 0.6822916865348816]
```

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

I have now changed the previous dataset to a new dataset which is from the breastcancer.csv file. The code to this question is available in the python file named breastcancer.ipynb.

```
[1]  from google.colab import drive
     drive.mount('/content/gdrive')

     Mounted at /content/gdrive
```

```
[34] path_to_csv = '/content/gdrive/My Drive/breastcancer.csv'
```

```
#Importing packages for creating arrays
import numpy as np
import pandas as pd

#Importing packages to convert Categorical data into Numerical
from sklearn.preprocessing import LabelEncoder

#Importing packages for splitting data
from sklearn.model_selection import train_test_split

#Importing packages for keras
from keras.models import Sequential
from keras.layers.core import Dense, Activation
```

```
[36] #Loading the Dataset
     dataset = pd.read_csv(path_to_csv, header=0)
```

```
dataset
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	texture_wors
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	...	17.3
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	...	23.4
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	...	25.5
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	...	26.5
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	...	16.6
...	...	...	...	...	...	...	...	...	...	...	...	...
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	...	26.4
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	...	38.2
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	...	34.1
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	...	39.4
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	...	30.3

569 rows x 33 columns

```python
#converting Categorical data into Numerical Using Label Encoding
le=LabelEncoder()
dataset['diagnosis'] = le.fit_transform(dataset['diagnosis'])
```

```python
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    int64
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
```

```python
#Splitting data into Feature Matrix & Label Matrix
X_train, X_test, Y_train, Y_test = train_test_split(dataset.iloc[:,2:32], dataset.iloc[:,1], test_size=0.25, random_state=87)
```

```python
my_first_nn = Sequential() # create model
my_first_nn.add(Dense(20, input_dim=30, 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']) # compilation
my_first_nn_fitted = my_first_nn.fit(X_train, Y_train, epochs=100,verbose=0, initial_epoch=0) # Training
print(my_first_nn.summary()) #Summary
print(my_first_nn.evaluate(X_test, Y_test)) #Evaluating
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 20)                620

 dense_5 (Dense)             (None, 1)                 21

=================================================================
Total params: 641
Trainable params: 641
Non-trainable params: 0
_________________________________________________________________
None
5/5 [==============================] - 0s 3ms/step - loss: 0.2621 - acc: 0.9161
[0.2620822787284851, 0.9160839319229126]
```

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()

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

```
[42] #importing packages for Normalization
     from sklearn.preprocessing import StandardScaler
```

```
my_first_nn = Sequential() # create model
my_first_nn.add(Dense(20, input_dim=30, 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']) # compilation

sc = StandardScaler() #Create Model
X_train = sc.fit_transform(X_train) #Fit to data, then transform it.
X_test = sc.transform(X_test) # Perform standardization by centering and scaling

my_first_nn_fitted = my_first_nn.fit(X_train, Y_train, epochs=100,verbose=0, initial_epoch=0) # Training
print(my_first_nn.summary()) #Summary
print(my_first_nn.evaluate(X_test, Y_test)) #Evaluating
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 20)                620

 dense_3 (Dense)             (None, 1)                 21

=================================================================
Total params: 641
Trainable params: 641
Non-trainable params: 0
_________________________________________________________________
None
5/5 [==============================] - 0s 3ms/step - loss: 0.1907 - acc: 0.9650
[0.19068476557731628, 0.9650349617004395]
```

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

For the above task I have used Image Classification on the MNIST dataset.

I have first loaded the MNIST dataset, which is a large database of handwritten digits commonly used for image classification tasks as shown below.
The mnist.load_data() function returns two tuples: (train_images, train_labels) and (test_images, test_labels). I have then written code to visualize the image and its corresponding label to get a better understanding of the data.
plt.imshow(train_images[0,:,:],cmap='gray') displays the first image in the training data. The cmap='gray' argument sets the color map to grayscale.plt.title('Ground Truth : {}'.format(train_labels[0])) adds a title to the image with the label of the image. plt.show() shows the image on the screen.

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

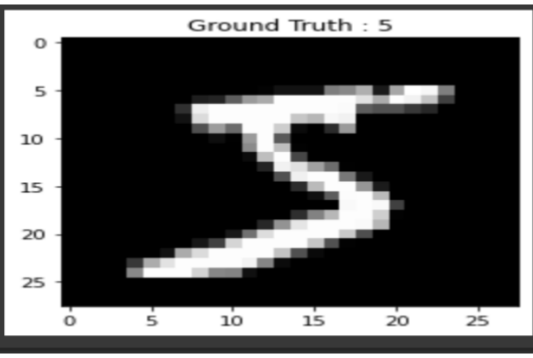
[1]
```python
from keras import Sequential
from keras.datasets import mnist
import numpy as np
from keras.layers import Dense
from keras.utils import to_categorical
```

[2]
```python
(train_images,train_labels),(test_images, test_labels) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

```python
import matplotlib.pyplot as plt

#display the first image in the training data
plt.imshow(train_images[0,:,:],cmap='gray')
plt.title('Ground Truth : {}'.format(train_labels[0]))
plt.show()
```



[4]
```python
train_images.shape[1:]
```

```
(28, 28)
```

```python
#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)
```

```
784
```

[6]
```python
#convert data to float and scale values between 0 and 1
train_data = train_data.astype('float')
test_data = test_data.astype('float')
```

[7]
```python
#scale data
train_data /=255.0
test_data /=255.0
```

[8]
```python
#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)
```

[9]
```python
#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'))
```

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

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

```
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 [==============================] - 5s 17ms/step - loss: 0.2869 - accuracy: 0.9123 - val_loss: 0.1388 - val_accuracy: 0.9533
Epoch 2/10
235/235 [==============================] - 4s 16ms/step - loss: 0.0994 - accuracy: 0.9691 - val_loss: 0.0785 - val_accuracy: 0.9756
Epoch 3/10
235/235 [==============================] - 4s 19ms/step - loss: 0.0626 - accuracy: 0.9807 - val_loss: 0.0713 - val_accuracy: 0.9766
Epoch 4/10
235/235 [==============================] - 4s 17ms/step - loss: 0.0442 - accuracy: 0.9855 - val_loss: 0.0979 - val_accuracy: 0.9711
Epoch 5/10
235/235 [==============================] - 4s 16ms/step - loss: 0.0300 - accuracy: 0.9909 - val_loss: 0.0673 - val_accuracy: 0.9801
Epoch 6/10
235/235 [==============================] - 4s 18ms/step - loss: 0.0231 - accuracy: 0.9926 - val_loss: 0.0650 - val_accuracy: 0.9813
Epoch 7/10
235/235 [==============================] - 4s 16ms/step - loss: 0.0160 - accuracy: 0.9951 - val_loss: 0.0658 - val_accuracy: 0.9809
Epoch 8/10
235/235 [==============================] - 4s 16ms/step - loss: 0.0134 - accuracy: 0.9958 - val_loss: 0.0684 - val_accuracy: 0.9819
Epoch 9/10
235/235 [==============================] - 4s 18ms/step - loss: 0.0100 - accuracy: 0.9969 - val_loss: 0.0752 - val_accuracy: 0.9821
Epoch 10/10
235/235 [==============================] - 4s 16ms/step - loss: 0.0073 - accuracy: 0.9976 - val_loss: 0.0744 - val_accuracy: 0.9819
```

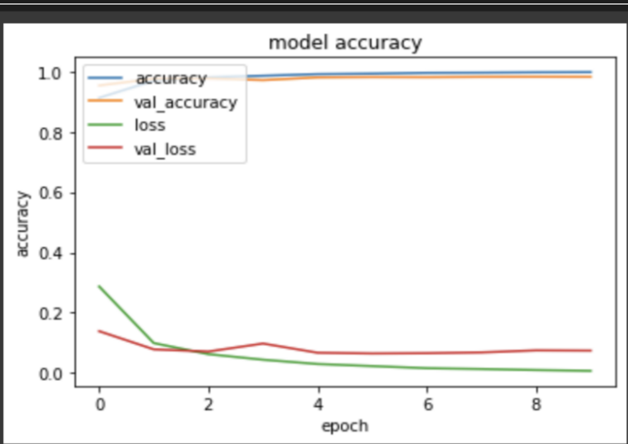
```
[11] [test_loss, test_acc] = model.evaluate(test_data, test_labels_one_hot)
     print("Evaluation result on Test Data : Loss = {}, accuracy = {}".format(test_loss, test_acc))
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0744 - accuracy: 0.9819
Evaluation result on Test Data : Loss = 0.07440487295389175, accuracy = 0.9818999767303467
```

```
[12] history.history.keys()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

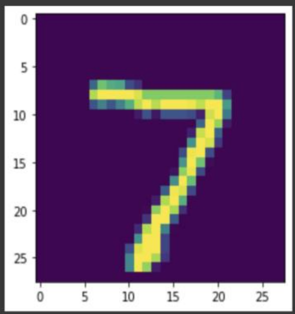
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['accuracy', 'val_accuracy','loss','val_loss'], loc='upper left')
plt.show()
```



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.**

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.

```python
plt.imshow(test_data[0].reshape(28,28));
```




```python
print("predicted label:",model.predict(test_data[0].reshape(1,784)))
```

```
1/1 [==============================] - 0s 83ms/step
predicted label: [[1.7450702e-11 1.5626347e-10 9.0478602e-10 1.1921726e-07 5.5921041e-14
  3.4721317e-13 1.2115700e-15 9.9999964e-01 2.0586902e-12 2.7695830e-07]]
```

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.**

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.

```python
#increasing the number of hidden layers to 4
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(dimData,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(512, activation='relu'))
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))

[test_loss1, test_acc1] = model.evaluate(test_data, test_labels_one_hot)
print("Evaluation result on Test Data with 4 hidden layers: Loss = {}, accuracy = {}".format(test_loss1, test_acc1))
```

```
Epoch 1/10
235/235 [==============================] - 9s 35ms/step - loss: 0.3540 - accuracy: 0.8885 - val_loss: 0.2443 - val_accuracy: 0.9265
Epoch 2/10
235/235 [==============================] - 8s 32ms/step - loss: 0.1075 - accuracy: 0.9673 - val_loss: 0.1842 - val_accuracy: 0.9443
Epoch 3/10
235/235 [==============================] - 7s 29ms/step - loss: 0.0674 - accuracy: 0.9792 - val_loss: 0.1315 - val_accuracy: 0.9622
Epoch 4/10
235/235 [==============================] - 8s 34ms/step - loss: 0.0480 - accuracy: 0.9847 - val_loss: 0.1110 - val_accuracy: 0.9669
Epoch 5/10
235/235 [==============================] - 7s 29ms/step - loss: 0.0350 - accuracy: 0.9891 - val_loss: 0.1143 - val_accuracy: 0.9688
Epoch 6/10
235/235 [==============================] - 7s 31ms/step - loss: 0.0259 - accuracy: 0.9916 - val_loss: 0.0852 - val_accuracy: 0.9784
Epoch 7/10
235/235 [==============================] - 10s 42ms/step - loss: 0.0218 - accuracy: 0.9931 - val_loss: 0.1725 - val_accuracy: 0.9567
Epoch 8/10
235/235 [==============================] - 7s 30ms/step - loss: 0.0177 - accuracy: 0.9942 - val_loss: 0.0722 - val_accuracy: 0.9824
Epoch 9/10
235/235 [==============================] - 7s 31ms/step - loss: 0.0160 - accuracy: 0.9951 - val_loss: 0.0697 - val_accuracy: 0.9828
Epoch 10/10
235/235 [==============================] - 8s 36ms/step - loss: 0.0123 - accuracy: 0.9961 - val_loss: 0.0935 - val_accuracy: 0.9804
313/313 [==============================] - 1s 3ms/step - loss: 0.0935 - accuracy: 0.9804
Evaluation result on Test Data with 4 hidden layers: Loss = 0.09346473962068558, accuracy = 0.980400025844574
```

```python
#increasing the dense in hidden layers
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(dimData,)))
model.add(Dense(612, activation='relu'))
model.add(Dense(712, activation='relu'))
model.add(Dense(812, 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))

[test_loss2, test_acc2] = model.evaluate(test_data, test_labels_one_hot)
print("Evaluation result on Test Data with increase in dense in hidden layers: Loss = {}, accuracy = {}".format(test_loss2, test_acc2))
```

```
Epoch 1/10
235/235 [==============================] - 12s 47ms/step - loss: 0.3537 - accuracy: 0.8866 - val_loss: 0.1845 - val_accuracy: 0.9452
Epoch 2/10
235/235 [==============================] - 10s 42ms/step - loss: 0.1043 - accuracy: 0.9687 - val_loss: 0.1200 - val_accuracy: 0.9630
Epoch 3/10
235/235 [==============================] - 11s 45ms/step - loss: 0.0665 - accuracy: 0.9799 - val_loss: 0.0991 - val_accuracy: 0.9710
Epoch 4/10
235/235 [==============================] - 11s 47ms/step - loss: 0.0485 - accuracy: 0.9850 - val_loss: 0.0751 - val_accuracy: 0.9779
Epoch 5/10
235/235 [==============================] - 10s 43ms/step - loss: 0.0344 - accuracy: 0.9892 - val_loss: 0.0830 - val_accuracy: 0.9762
Epoch 6/10
235/235 [==============================] - 9s 39ms/step - loss: 0.0282 - accuracy: 0.9912 - val_loss: 0.0760 - val_accuracy: 0.9796
Epoch 7/10
235/235 [==============================] - 11s 46ms/step - loss: 0.0198 - accuracy: 0.9934 - val_loss: 0.0710 - val_accuracy: 0.9825
Epoch 8/10
235/235 [==============================] - 10s 43ms/step - loss: 0.0162 - accuracy: 0.9947 - val_loss: 0.1041 - val_accuracy: 0.9752
Epoch 9/10
235/235 [==============================] - 10s 42ms/step - loss: 0.0143 - accuracy: 0.9957 - val_loss: 0.0789 - val_accuracy: 0.9812
Epoch 10/10
235/235 [==============================] - 11s 45ms/step - loss: 0.0120 - accuracy: 0.9963 - val_loss: 0.0803 - val_accuracy: 0.9814
313/313 [==============================] - 1s 3ms/step - loss: 0.0803 - accuracy: 0.9814
Evaluation result on Test Data with increase in dense in hidden layers: Loss = 0.08034028857946396, accuracy = 0.9814000129699707
```

```python
#All hidden layers with tanh activation
model = Sequential()
model.add(Dense(512, activation='tanh', input_shape=(dimData,)))
model.add(Dense(612, activation='tanh'))
model.add(Dense(712, activation='tanh'))
model.add(Dense(812, activation='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))

[test_loss3, test_acc3] = model.evaluate(test_data, test_labels_one_hot)
print("Evaluation result on Test Data with tanh activation: Loss = {}, accuracy = {}".format(test_loss3, test_acc3))
```

```
Epoch 1/10
235/235 [==============================] - 11s 43ms/step - loss: 0.5976 - accuracy: 0.8387 - val_loss: 0.3145 - val_accuracy: 0.8989
Epoch 2/10
235/235 [==============================] - 10s 45ms/step - loss: 0.2167 - accuracy: 0.9323 - val_loss: 0.2426 - val_accuracy: 0.9261
Epoch 3/10
235/235 [==============================] - 10s 43ms/step - loss: 0.1556 - accuracy: 0.9509 - val_loss: 0.3126 - val_accuracy: 0.9064
Epoch 4/10
235/235 [==============================] - 10s 44ms/step - loss: 0.1264 - accuracy: 0.9609 - val_loss: 0.1473 - val_accuracy: 0.9544
Epoch 5/10
235/235 [==============================] - 10s 44ms/step - loss: 0.1084 - accuracy: 0.9651 - val_loss: 0.2393 - val_accuracy: 0.9195
Epoch 6/10
235/235 [==============================] - 9s 40ms/step - loss: 0.0977 - accuracy: 0.9687 - val_loss: 0.1798 - val_accuracy: 0.9424
Epoch 7/10
235/235 [==============================] - 10s 42ms/step - loss: 0.0880 - accuracy: 0.9720 - val_loss: 0.1765 - val_accuracy: 0.9445
Epoch 8/10
235/235 [==============================] - 11s 46ms/step - loss: 0.0806 - accuracy: 0.9736 - val_loss: 0.1343 - val_accuracy: 0.9587
Epoch 9/10
235/235 [==============================] - 10s 42ms/step - loss: 0.0753 - accuracy: 0.9755 - val_loss: 0.1591 - val_accuracy: 0.9528
Epoch 10/10
235/235 [==============================] - 10s 43ms/step - loss: 0.0706 - accuracy: 0.9774 - val_loss: 0.1125 - val_accuracy: 0.9664
313/313 [==============================] - 1s 4ms/step - loss: 0.1125 - accuracy: 0.9664
Evaluation result on Test Data with tanh activation: Loss = 0.11245911568403244, accuracy = 0.9664000272750854
```

```python
#All hidden layers with sigmoid activation
model = Sequential()
model.add(Dense(512, activation='sigmoid', input_shape=(dimData,)))
model.add(Dense(612, activation='sigmoid'))
model.add(Dense(712, activation='sigmoid'))
model.add(Dense(812, activation='sigmoid'))
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))

[test_loss4, test_acc4] = model.evaluate(test_data, test_labels_one_hot)
print("Evaluation result on Test Data with sigmoid activation: Loss = {}, accuracy = {}".format(test_loss4, test_acc4))
```

```
Epoch 1/10
235/235 [==============================] - 11s 43ms/step - loss: 1.0092 - accuracy: 0.6530 - val_loss: 0.4195 - val_accuracy: 0.8709
Epoch 2/10
235/235 [==============================] - 10s 42ms/step - loss: 0.3233 - accuracy: 0.9012 - val_loss: 0.2781 - val_accuracy: 0.9092
Epoch 3/10
235/235 [==============================] - 10s 43ms/step - loss: 0.2285 - accuracy: 0.9300 - val_loss: 0.1991 - val_accuracy: 0.9365
Epoch 4/10
235/235 [==============================] - 9s 40ms/step - loss: 0.1905 - accuracy: 0.9413 - val_loss: 0.5257 - val_accuracy: 0.8392
Epoch 5/10
235/235 [==============================] - 10s 42ms/step - loss: 0.1644 - accuracy: 0.9494 - val_loss: 0.1717 - val_accuracy: 0.9483
Epoch 6/10
235/235 [==============================] - 10s 42ms/step - loss: 0.1516 - accuracy: 0.9528 - val_loss: 0.2163 - val_accuracy: 0.9355
Epoch 7/10
235/235 [==============================] - 10s 42ms/step - loss: 0.1390 - accuracy: 0.9572 - val_loss: 0.1455 - val_accuracy: 0.9531
Epoch 8/10
235/235 [==============================] - 10s 43ms/step - loss: 0.1299 - accuracy: 0.9595 - val_loss: 0.1440 - val_accuracy: 0.9561
Epoch 9/10
235/235 [==============================] - 11s 45ms/step - loss: 0.1187 - accuracy: 0.9630 - val_loss: 0.1443 - val_accuracy: 0.9566
Epoch 10/10
235/235 [==============================] - 10s 41ms/step - loss: 0.1117 - accuracy: 0.9659 - val_loss: 0.1305 - val_accuracy: 0.9596
313/313 [==============================] - 1s 3ms/step - loss: 0.1305 - accuracy: 0.9596
Evaluation result on Test Data with sigmoid activation: Loss = 0.13049034774303436, accuracy = 0.9595999717712402
```

```python
print("Evaluation result on Test Data with 2 hidden layers: Loss = {}, accuracy = {}".format(test_loss, test_acc))
print("Evaluation result on Test Data with 4 hidden layers: Loss = {}, accuracy = {}".format(test_loss1, test_acc1))
print("Evaluation result on Test Data with increase in dense in hidden layers: Loss = {}, accuracy = {}".format(test_loss2, test_acc2))
print("Evaluation result on Test Data with tanh activation: Loss = {}, accuracy = {}".format(test_loss3, test_acc3))
print("Evaluation result on Test Data with sigmoid activation: Loss = {}, accuracy = {}".format(test_loss4, test_acc4))
```

```
Evaluation result on Test Data with 2 hidden layers: Loss = 0.07440487295389175, accuracy = 0.9818999767303467
Evaluation result on Test Data with 4 hidden layers: Loss = 0.09346473962068558, accuracy = 0.980400025844574
Evaluation result on Test Data with increase in dense in hidden layers: Loss = 0.08034028857946396, accuracy = 0.9814000129699707
Evaluation result on Test Data with tanh activation: Loss = 0.11245911568403244, accuracy = 0.9664000272750854
Evaluation result on Test Data with sigmoid activation: Loss = 0.13049034774303436, accuracy = 0.9595999717712402
```

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

4. 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')

#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)
```

```python
#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))

[test_loss5, test_acc5] = model.evaluate(test_data, test_labels_one_hot)
print("Evaluation result on Test Data without scaling: Loss = {}, accuracy = {}".format(test_loss5, test_acc5))
```

```
(28, 28)
Epoch 1/10
235/235 [==============================] - 5s 19ms/step - loss: 5.9833 - accuracy: 0.8760 - val_loss: 0.7219 - val_accuracy: 0.9238
Epoch 2/10
235/235 [==============================] - 4s 16ms/step - loss: 0.4283 - accuracy: 0.9434 - val_loss: 0.3116 - val_accuracy: 0.9529
Epoch 3/10
235/235 [==============================] - 4s 16ms/step - loss: 0.2436 - accuracy: 0.9584 - val_loss: 0.3553 - val_accuracy: 0.9397
Epoch 4/10
235/235 [==============================] - 4s 18ms/step - loss: 0.2062 - accuracy: 0.9651 - val_loss: 0.5098 - val_accuracy: 0.9419
Epoch 5/10
235/235 [==============================] - 4s 16ms/step - loss: 0.1623 - accuracy: 0.9725 - val_loss: 0.3150 - val_accuracy: 0.9513
Epoch 6/10
235/235 [==============================] - 4s 16ms/step - loss: 0.1517 - accuracy: 0.9761 - val_loss: 0.2748 - val_accuracy: 0.9651
Epoch 7/10
235/235 [==============================] - 4s 18ms/step - loss: 0.1359 - accuracy: 0.9784 - val_loss: 0.2715 - val_accuracy: 0.9681
Epoch 8/10
235/235 [==============================] - 4s 16ms/step - loss: 0.1251 - accuracy: 0.9809 - val_loss: 0.3734 - val_accuracy: 0.9654
Epoch 9/10
235/235 [==============================] - 4s 16ms/step - loss: 0.1241 - accuracy: 0.9824 - val_loss: 0.3609 - val_accuracy: 0.9685
Epoch 10/10
235/235 [==============================] - 4s 18ms/step - loss: 0.1149 - accuracy: 0.9839 - val_loss: 0.3727 - val_accuracy: 0.9711
313/313 [==============================] - 1s 2ms/step - loss: 0.3727 - accuracy: 0.9711
Evaluation result on Test Data without scaling: Loss = 0.37273916602134705, accuracy = 0.9710999727249146
```

```
print("Evaluation result on Test Data with 2 hidden layers: Loss = {}, accuracy = {}".format(test_loss, test_acc))
print("Evaluation result on Test Data with 4 hidden layers: Loss = {}, accuracy = {}".format(test_loss1, test_acc1))
print("Evaluation result on Test Data with increase in dense in hidden layers: Loss = {}, accuracy = {}".format(test_loss2, test_acc2))
print("Evaluation result on Test Data with tanh activation: Loss = {}, accuracy = {}".format(test_loss3, test_acc3))
print("Evaluation result on Test Data with sigmoid activation: Loss = {}, accuracy = {}".format(test_loss4, test_acc4))
print("Evaluation result on Test Data without scaling: Loss = {}, accuracy = {}".format(test_loss5, test_acc5))
```

```
Evaluation result on Test Data with 2 hidden layers: Loss = 0.07440487295389175, accuracy = 0.9818999767303467
Evaluation result on Test Data with 4 hidden layers: Loss = 0.09346473962068558, accuracy = 0.980400025844574
Evaluation result on Test Data with increase in dense in hidden layers: Loss = 0.08034028857946396, accuracy = 0.9814000129699707
Evaluation result on Test Data with tanh activation: Loss = 0.11245911568403244, accuracy = 0.9664000272750854
Evaluation result on Test Data with sigmoid activation: Loss = 0.13049034774303436, accuracy = 0.9595999717712402
Evaluation result on Test Data without scaling: Loss = 0.37273916602134705, accuracy = 0.9710999727249146
```