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

    Mounted at /content/gdrive
```

```
[4] path_to_csv = '/content/gdrive/MyDrive/diabetes(1).csv'
```

```
# 1) Use the use case in the class: Add more Dense layers to the existing code and check how the accuracy changes.
import keras
import pandas
from keras.models import Sequential
from keras.layers 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=0.1, random_state=30)
np.random.seed(155)
my_first_nn = Sequential() # create model
my_first_nn.add(Dense(16, activation='relu', input_shape=(8,)))
my_first_nn.add(Dense(8, activation='relu'))
my_first_nn.add(Dense(64, activation='relu'))
my_first_nn.add(Dense(1, activation='sigmoid'))
my_first_nn.compile(loss='mean_squared_error', optimizer='adam', metrics=['acc'])
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 82/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1668 - acc: 0.7685
Epoch 83/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1673 - acc: 0.7569
Epoch 84/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1616 - acc: 0.7598
Epoch 85/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1636 - acc: 0.7525
Epoch 86/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1618 - acc: 0.7656
Epoch 87/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1673 - acc: 0.7699
Epoch 88/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1722 - acc: 0.7496
Epoch 89/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1686 - acc: 0.7467
Epoch 90/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1625 - acc: 0.7641
Epoch 91/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1634 - acc: 0.7598
Epoch 92/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1605 - acc: 0.7815
Epoch 93/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1605 - acc: 0.7670
Epoch 94/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1598 - acc: 0.7771
Epoch 95/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1594 - acc: 0.7670
Epoch 96/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1582 - acc: 0.7800
Epoch 97/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1637 - acc: 0.7583
Epoch 98/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1576 - acc: 0.7757
Epoch 99/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1591 - acc: 0.7685
Epoch 100/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1584 - acc: 0.7713
Model: "sequential"
```

```
________________________________________________________________
 Layer (type)                Output Shape              Param #
================================================================
 dense (Dense)               (None, 16)                144

 dense_1 (Dense)             (None, 8)                 136

 dense_2 (Dense)             (None, 64)                576

 dense_3 (Dense)             (None, 1)                 65

================================================================
Total params: 921 (3.60 KB)
Trainable params: 921 (3.60 KB)
Non-trainable params: 0 (0.00 Byte)
________________________________________________________________
None
3/3 [==============================] - 0s 4ms/step - loss: 0.1715 - acc: 0.7143
[0.17146751284599304, 0.7142857313156128]
```

```python
import keras
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,)))
model.add(Dense(256, activation='tanh'))
model.add(Dense(128, 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))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
(28, 28)
784
Epoch 1/10
235/235 [==============================] - 6s 22ms/step - loss: 0.3332 - accuracy: 0.8980 - val_loss: 0.2159 - val_accuracy: 0.9344
Epoch 2/10
235/235 [==============================] - 6s 26ms/step - loss: 0.1476 - accuracy: 0.9557 - val_loss: 0.1847 - val_accuracy: 0.9428
Epoch 3/10
235/235 [==============================] - 5s 19ms/step - loss: 0.0981 - accuracy: 0.9709 - val_loss: 0.1033 - val_accuracy: 0.9681
Epoch 4/10
235/235 [==============================] - 5s 23ms/step - loss: 0.0707 - accuracy: 0.9786 - val_loss: 0.1083 - val_accuracy: 0.9639
Epoch 5/10
235/235 [==============================] - 8s 32ms/step - loss: 0.0522 - accuracy: 0.9840 - val_loss: 0.0738 - val_accuracy: 0.9757
Epoch 6/10
235/235 [==============================] - 5s 21ms/step - loss: 0.0399 - accuracy: 0.9880 - val_loss: 0.0745 - val_accuracy: 0.9761
Epoch 7/10
235/235 [==============================] - 7s 30ms/step - loss: 0.0296 - accuracy: 0.9912 - val_loss: 0.0853 - val_accuracy: 0.9731
Epoch 8/10
235/235 [==============================] - 5s 21ms/step - loss: 0.0223 - accuracy: 0.9933 - val_loss: 0.0786 - val_accuracy: 0.9765
Epoch 9/10
235/235 [==============================] - 5s 19ms/step - loss: 0.0168 - accuracy: 0.9952 - val_loss: 0.0703 - val_accuracy: 0.9781
Epoch 10/10
235/235 [==============================] - 8s 34ms/step - loss: 0.0118 - accuracy: 0.9966 - val_loss: 0.0683 - val_accuracy: 0.9801
```

```python
# 2.2) Run the same code without scaling the images and check the performance?
import keras
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 (no scaling)
train_data = train_data.astype('float')
test_data = test_data.astype('float')

# Change the labels from integer 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(128, 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))
```

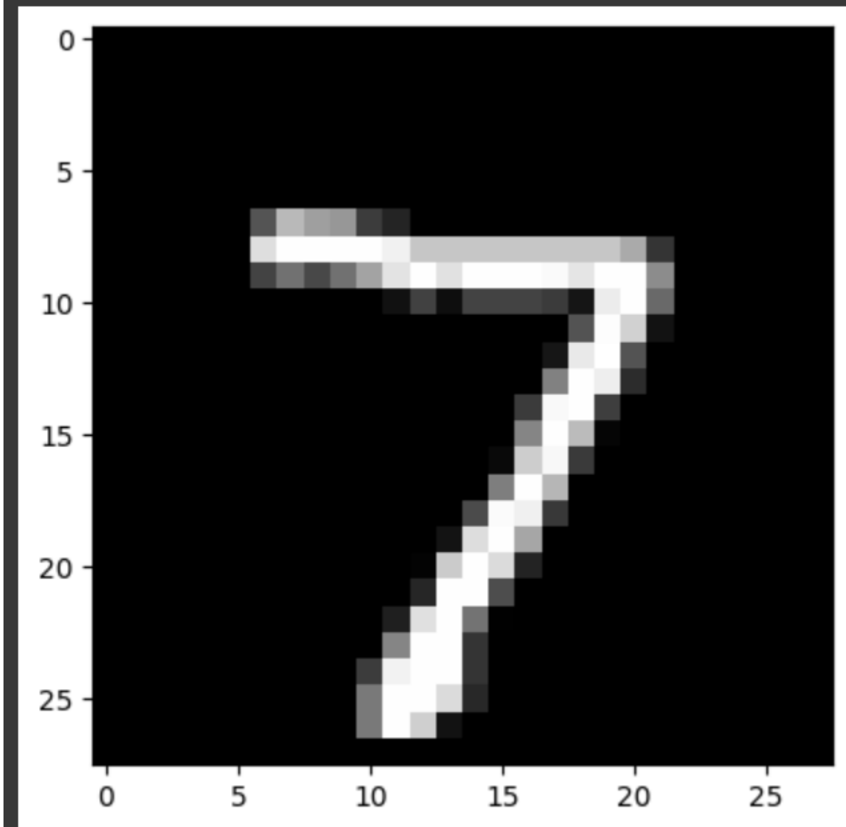
```
(28, 28)
784
Epoch 1/10
235/235 [==============================] - 7s 25ms/step - loss: 0.4886 - accuracy: 0.8663 - val_loss: 0.2099 - val_accuracy: 0.9396
Epoch 2/10
235/235 [==============================] - 8s 32ms/step - loss: 0.1724 - accuracy: 0.9499 - val_loss: 0.1357 - val_accuracy: 0.9591
Epoch 3/10
235/235 [==============================] - 6s 26ms/step - loss: 0.1347 - accuracy: 0.9601 - val_loss: 0.1183 - val_accuracy: 0.9634
Epoch 4/10
235/235 [==============================] - 7s 29ms/step - loss: 0.1145 - accuracy: 0.9659 - val_loss: 0.1180 - val_accuracy: 0.9645
Epoch 5/10
235/235 [==============================] - 7s 29ms/step - loss: 0.1004 - accuracy: 0.9700 - val_loss: 0.1256 - val_accuracy: 0.9617
Epoch 6/10
235/235 [==============================] - 6s 26ms/step - loss: 0.0954 - accuracy: 0.9710 - val_loss: 0.1091 - val_accuracy: 0.9658
Epoch 7/10
235/235 [==============================] - 7s 31ms/step - loss: 0.0872 - accuracy: 0.9735 - val_loss: 0.1032 - val_accuracy: 0.9673
Epoch 8/10
235/235 [==============================] - 6s 25ms/step - loss: 0.0817 - accuracy: 0.9753 - val_loss: 0.1262 - val_accuracy: 0.9640
Epoch 9/10
235/235 [==============================] - 7s 31ms/step - loss: 0.0784 - accuracy: 0.9770 - val_loss: 0.1028 - val_accuracy: 0.9675
Epoch 10/10
235/235 [==============================] - 6s 25ms/step - loss: 0.0720 - accuracy: 0.9784 - val_loss: 0.0938 - val_accuracy: 0.9724
```

```python
# 2.3) 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.
import matplotlib.pyplot as plt

# Choose an index to plot a test image
index = 0

# Plot the selected test image
plt.imshow(test_images[index], cmap='gray')
plt.show()

# Make a prediction on the selected test image
image = test_data[index].reshape(1, -1)
prediction = model.predict(image)
predicted_class = np.argmax(prediction)
print(f"Predicted Class: {predicted_class}")
```




```
1/1 [==============================] - 0s 186ms/step
Predicted Class: 7
```

Github repo: https://github.com/dheerukarra/BigDataAnalytics/tree/main/ICP_6

Youtube link: https://youtu.be/9rinhGlsCUE