**Fall 2023: CS5720 Neural Networks & Deep Learning - ICP-6**

**Assignment-6**

**NAME: SATWIK REDDY MOPURU**

**STUDENT ID:700740060**

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

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

path\_to\_csv = '/content/gdrive/MyDrive/Colab Notebooks/diabetes.csv' 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)

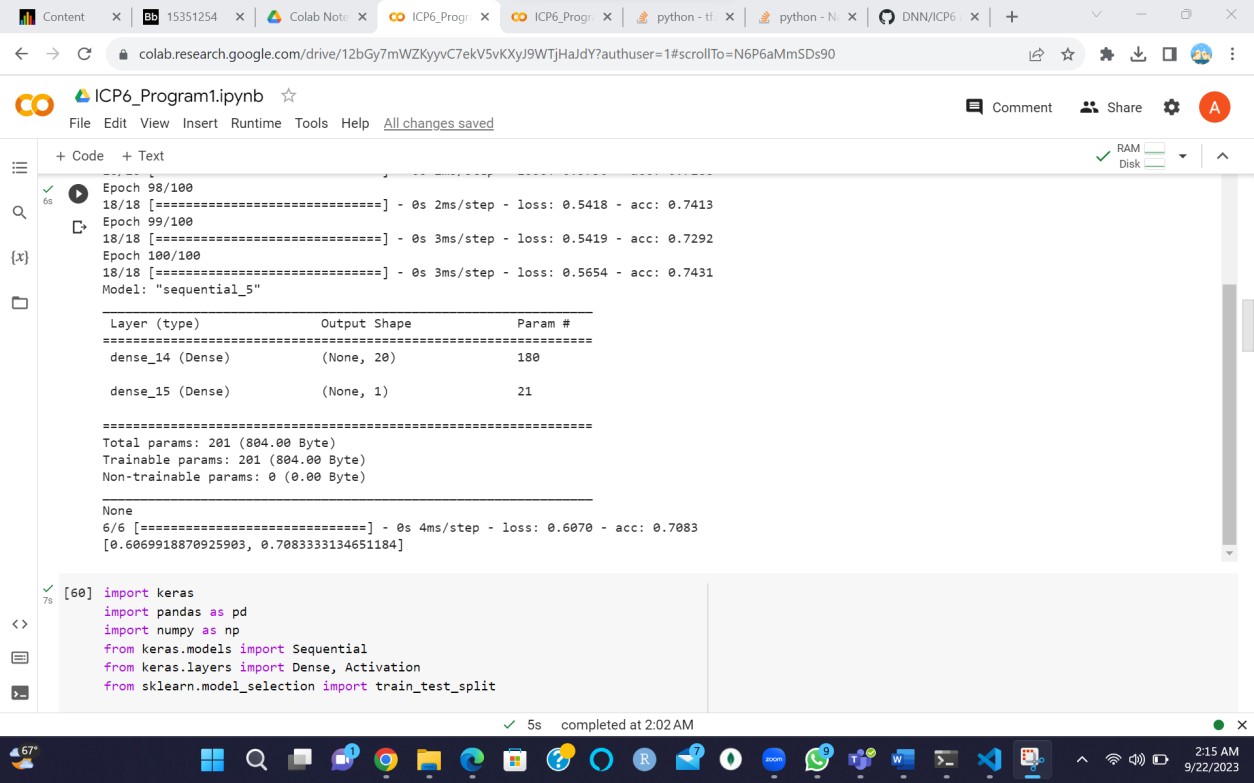
my\_first\_nn = Sequential() # create model my\_first\_nn.add(Dense(20, input\_dim=8, activation='relu')) # hidden layer

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))



The previous code is before the dense layers are added.

import keras import pandas as pd import numpy as np

from keras.models import Sequential

from keras.layers.core import Dense, Activation from sklearn.model\_selection import train\_test\_split

# load dataset

path\_to\_csv = '/content/gdrive/MyDrive/Colab Notebooks/diabetes.csv' dataset = pd.read\_csv(path\_to\_csv, header=None).values

# split dataset into training and test sets

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(dataset[:,0:8], dataset[:,8],

test\_size=0.25,

random\_state=87)

# define the model np.random.seed(155) my\_second\_nn = Sequential()

my\_second\_nn.add(Dense(20, input\_dim=8, activation='relu')) my\_second\_nn.add(Dense(20, input\_dim=8,activation='relu')) my\_second\_nn.add(Dense(20, input\_dim=8,activation='relu'))

my\_second\_nn.add(Dense(1, activation='sigmoid')) my\_second\_nn.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# train the model

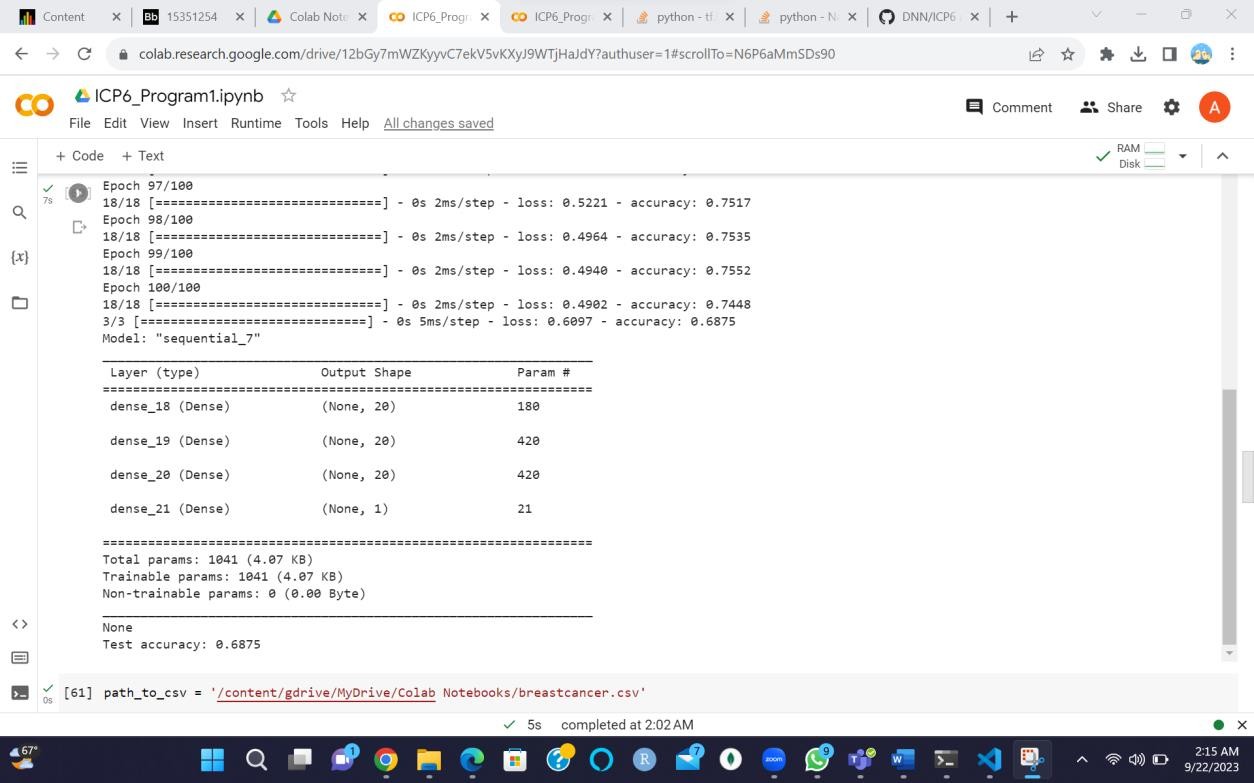
my\_second\_nn\_fitted= my\_second\_nn.fit(X\_train, Y\_train, epochs=100,

initial\_epoch=0)

# evaluate the model on the test set

score = my\_second\_nn.evaluate(X\_test, Y\_test, batch\_size=64) print(my\_second\_nn.summary())

print("Test accuracy:", score[1])



We added two more Dense layers with 20 nodes each in this version, both using the ReLU activation function. We kept the original code's batch size, optimizer, and loss function.

We can see that the accuracy of 2 code increases as we add more dense layers.

1.2,1.3 Change the data source to Breast Cancer dataset \* available in the source code folder and make required changes. Report accuracy of the model. 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 as pd import numpy as np

from sklearn.datasets import load\_breast\_cancer from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler from keras.models import Sequential

from keras.layers import Dense

# Load dataset

data = load\_breast\_cancer()

# Split dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data.data, data.target,

test\_size=0.25,

random\_state=87)

# Normalize data

sc = StandardScaler()

X\_train\_norm = sc.fit\_transform(X\_train) X\_test\_norm = sc.transform(X\_test)

# Create model np.random.seed(155) model = Sequential()

model.add(Dense(20, input\_dim=30, activation='relu')) model.add(Dense(1, activation='sigmoid')) model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

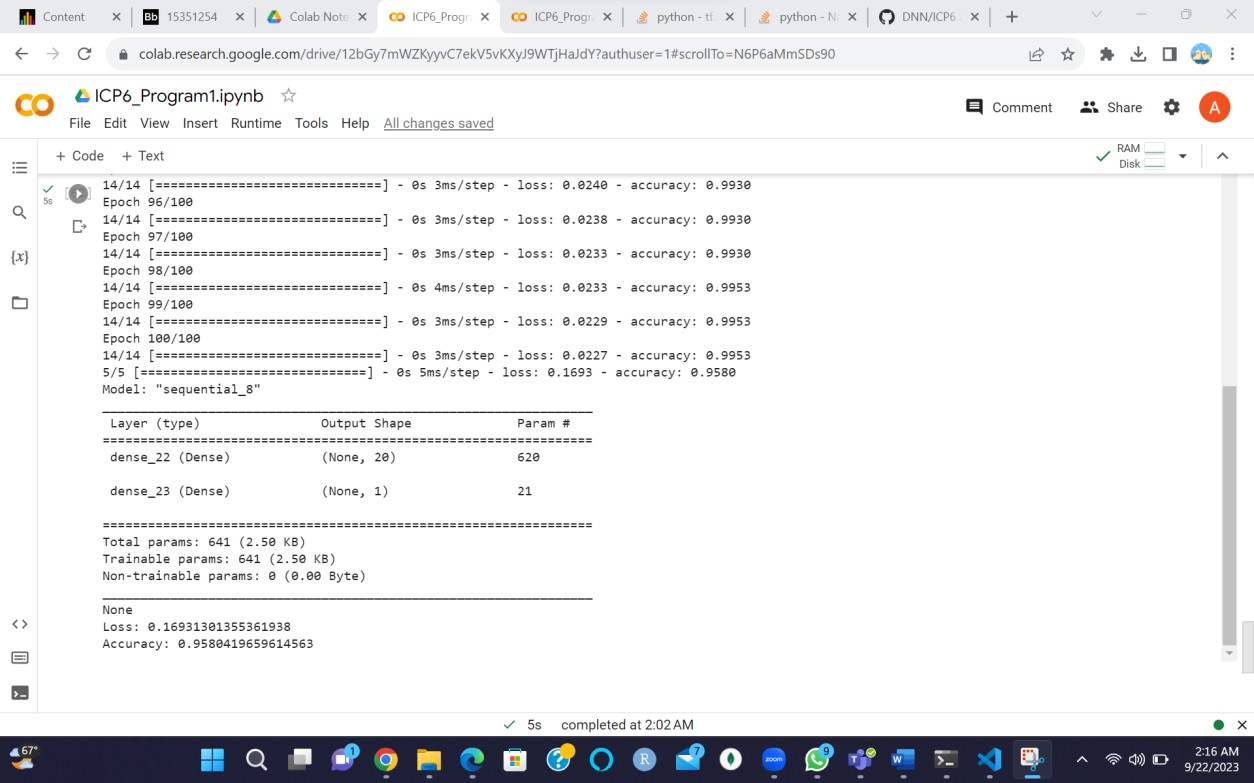
# Train model

model.fit(X\_train\_norm, y\_train, epochs=100, initial\_epoch=0)

# Evaluate model on testing set

loss, accuracy = model.evaluate(X\_test\_norm, y\_test) print(model.summary())

print("Loss:", loss) print("Accuracy:", accuracy)



In the previous code, we used a dataset from the sklearn datasets, normalized the data, and generated the results.

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

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)

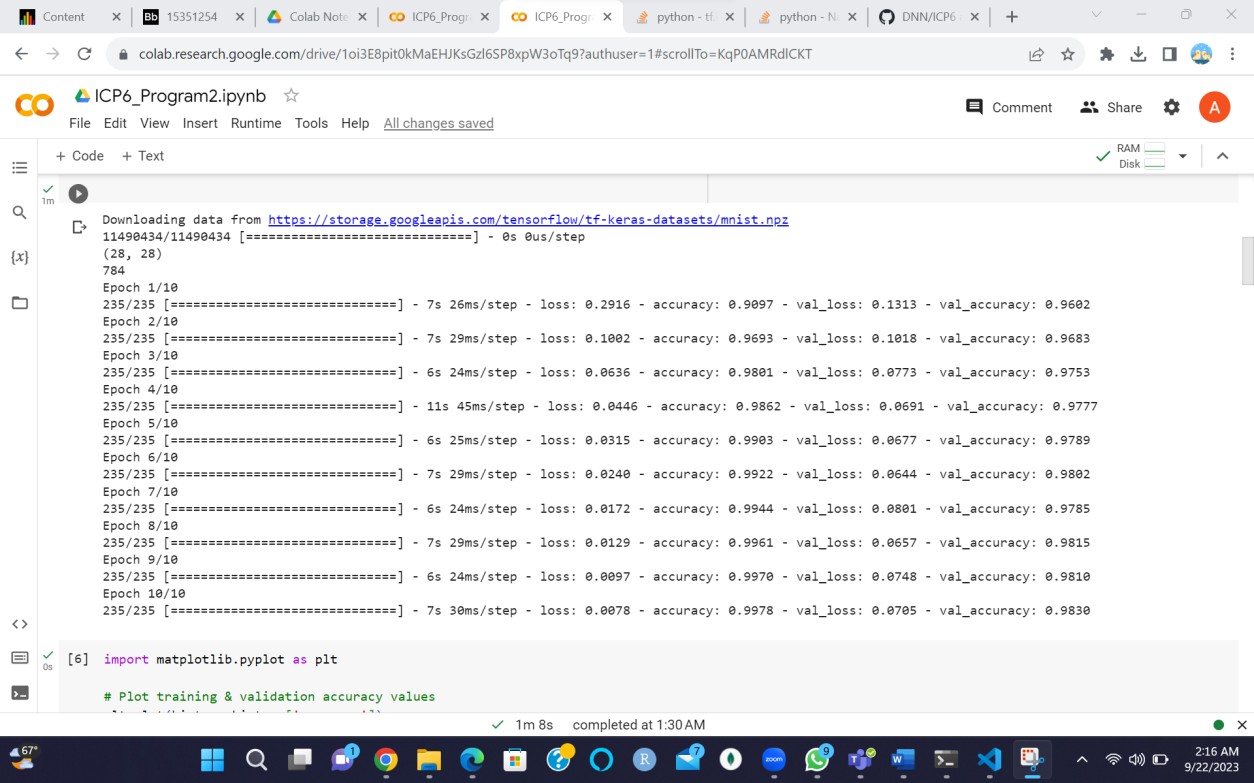
#creating network model = Sequential()

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))

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

import matplotlib.pyplot as plt

# Plot training & validation accuracy values plt.plot(history.history['accuracy']) plt.plot(history.history['val\_accuracy']) plt.title('Model accuracy') plt.ylabel('Accuracy')

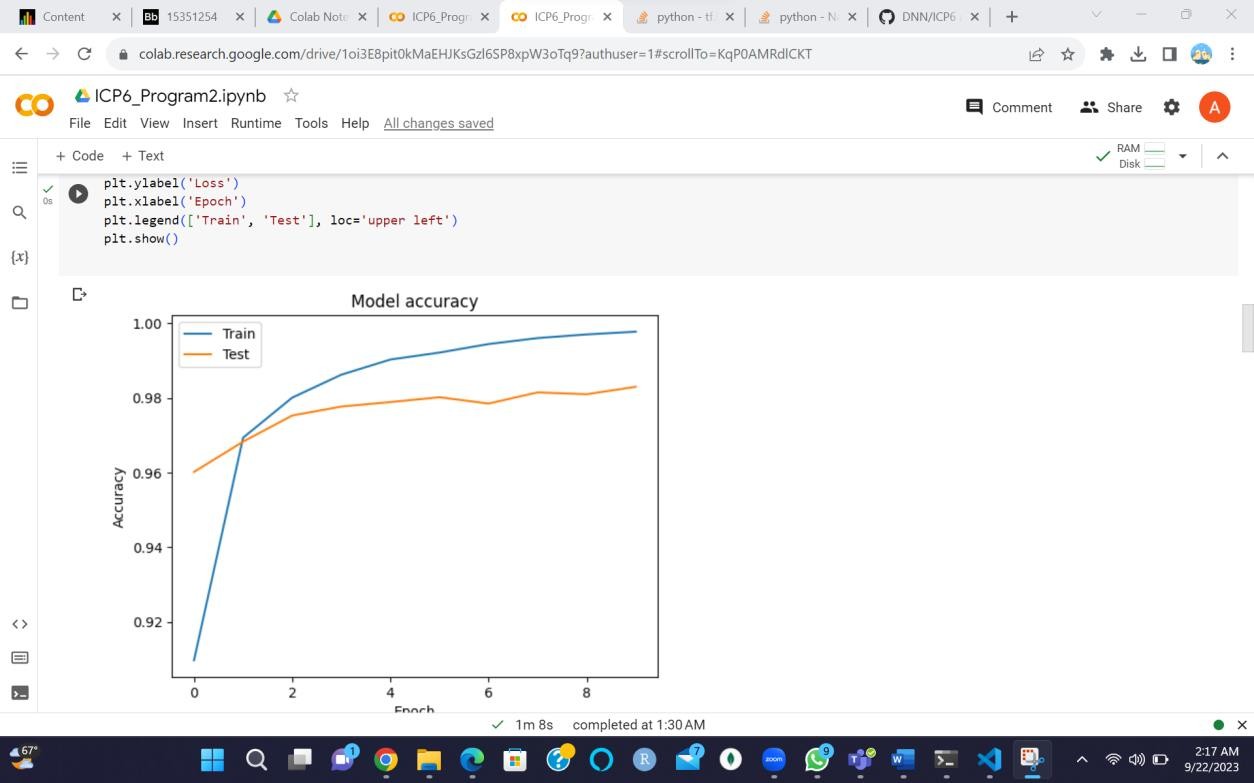
plt.xlabel('Epoch')

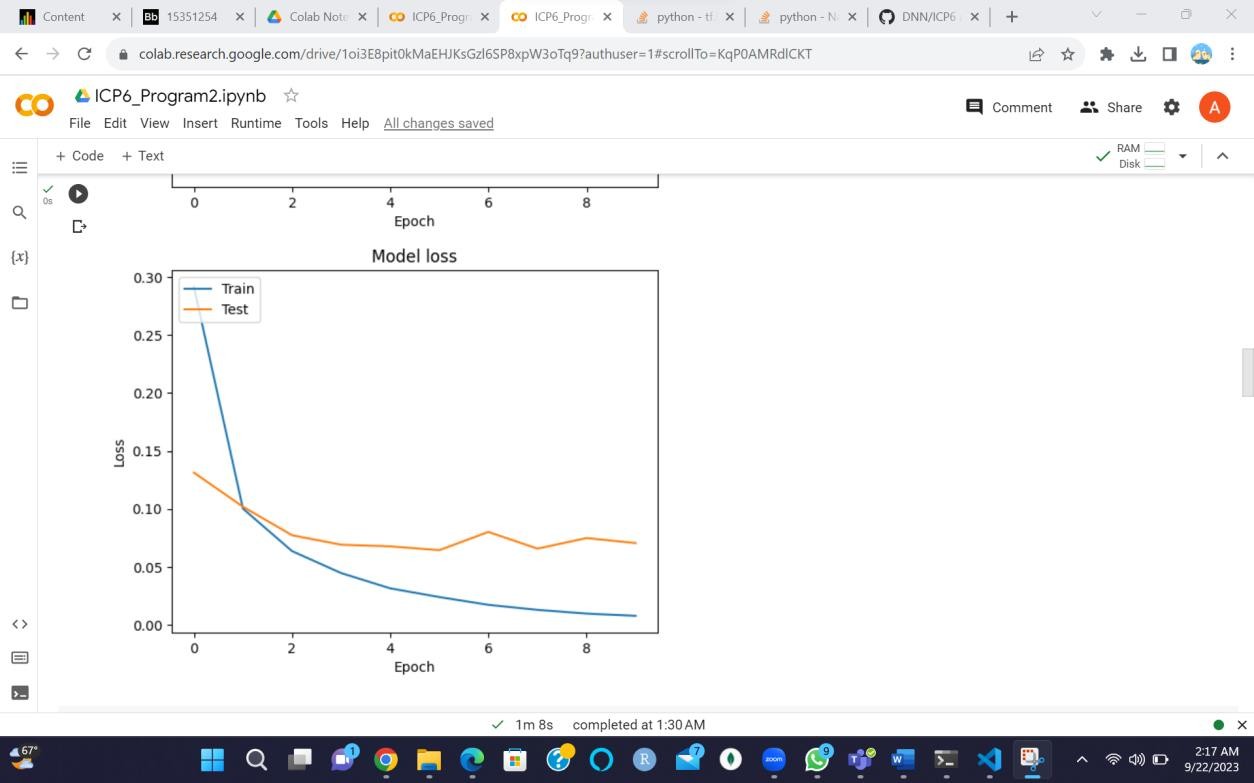
plt.legend(['Train', 'Test'], loc='upper left') plt.show()

# Plot training & validation loss values plt.plot(history.history['loss']) plt.plot(history.history['val\_loss']) plt.title('Model loss') plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left') plt.show()





This code produced two plots: one for accuracy values and one for loss values. The first plot depicts training and validation accuracy curves over epochs, while the second depicts training and validation loss curves over epochs.

1. 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

# select a random image from test data image\_index = 1234

img = test\_images[image\_index]

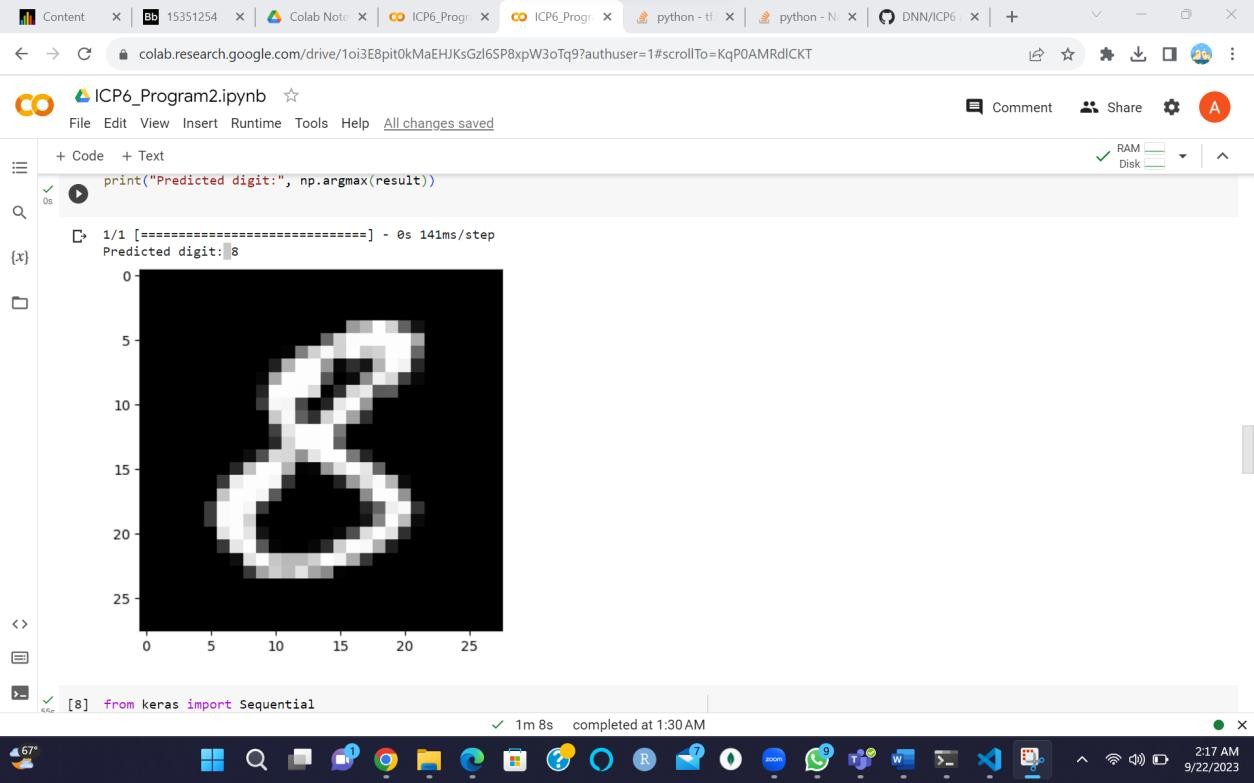
# plot the image plt.imshow(img, cmap='gray')

# reshape image to 1D vector img = img.reshape((1, 784))

# normalize pixel values img = img / 255.0

# predict class of image result = model.predict(img)

print("Predicted digit:", np.argmax(result))



This will plot the image at index 1234 in the test data and then use the trained model to predict the digit in the image.

1. 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()

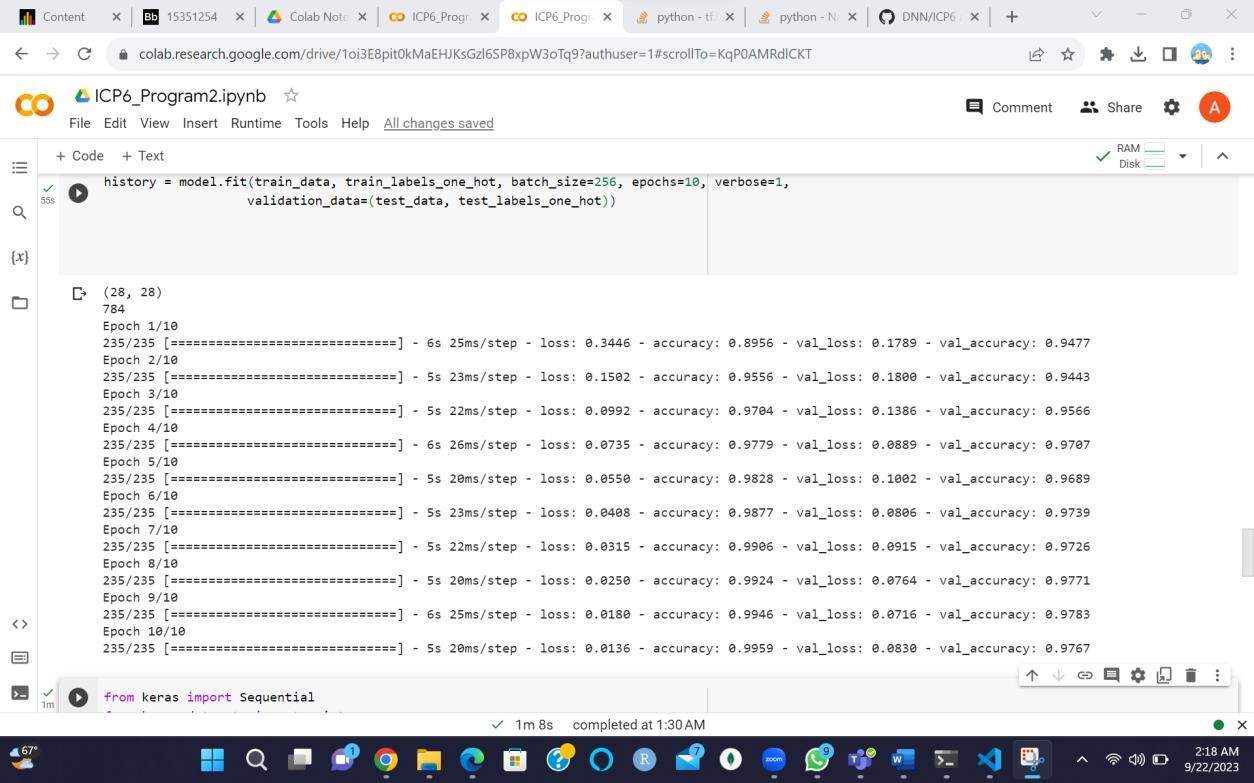
model.add(Dense(512, activation='tanh', input\_shape=(dimData,))) model.add(Dense(256, activation='tanh'))

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))



Here we are using the tanh function since we are using the tanh function the performance and accuracy may slightly vary

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

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)

#creating network model = Sequential()

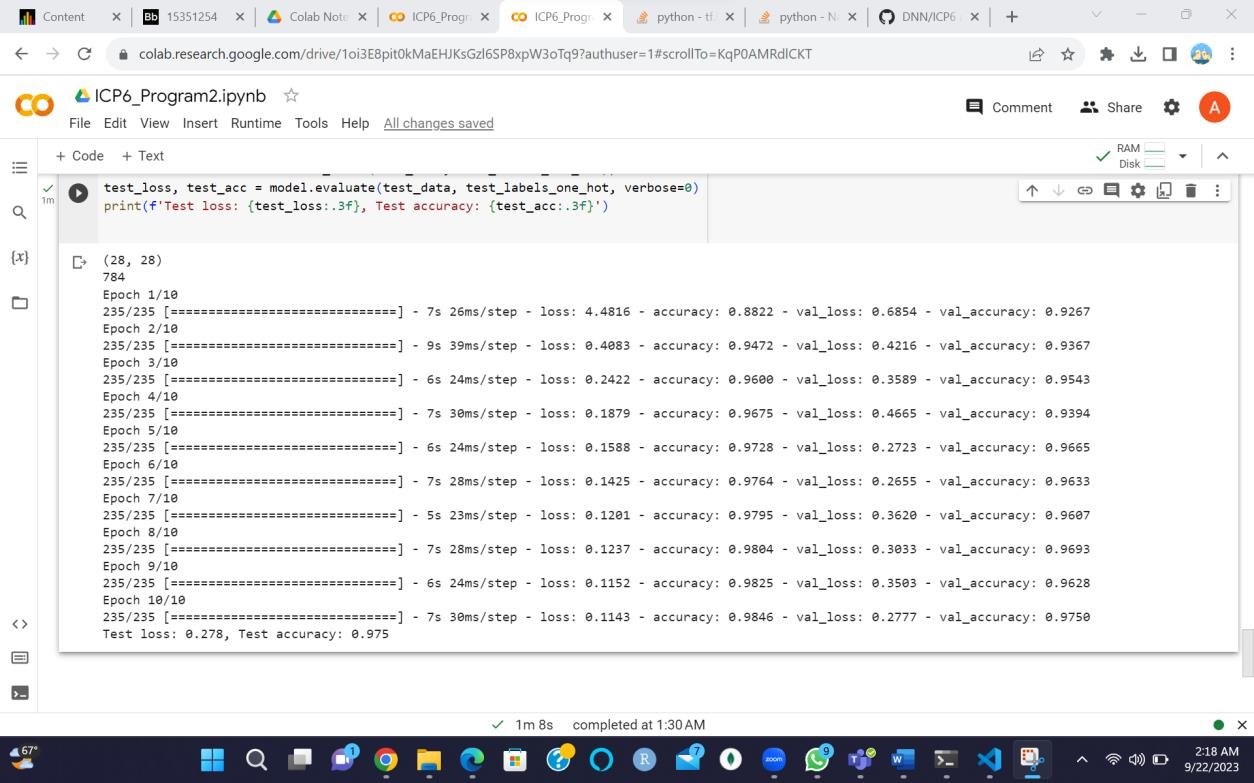
model.add(Dense(512, activation='relu', input\_shape=(dimData,))) model.add(Dense(512, activation='relu'))

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\_loss, test\_acc = model.evaluate(test\_data, test\_labels\_one\_hot, verbose=0)

print(f'Test loss: {test\_loss:.3f}, Test accuracy: {test\_acc:.3f}')

In the step we removed the normalization step by dividing the data by 255.0, as you can see even without normalization, the performance is quite good.

GitHub:-