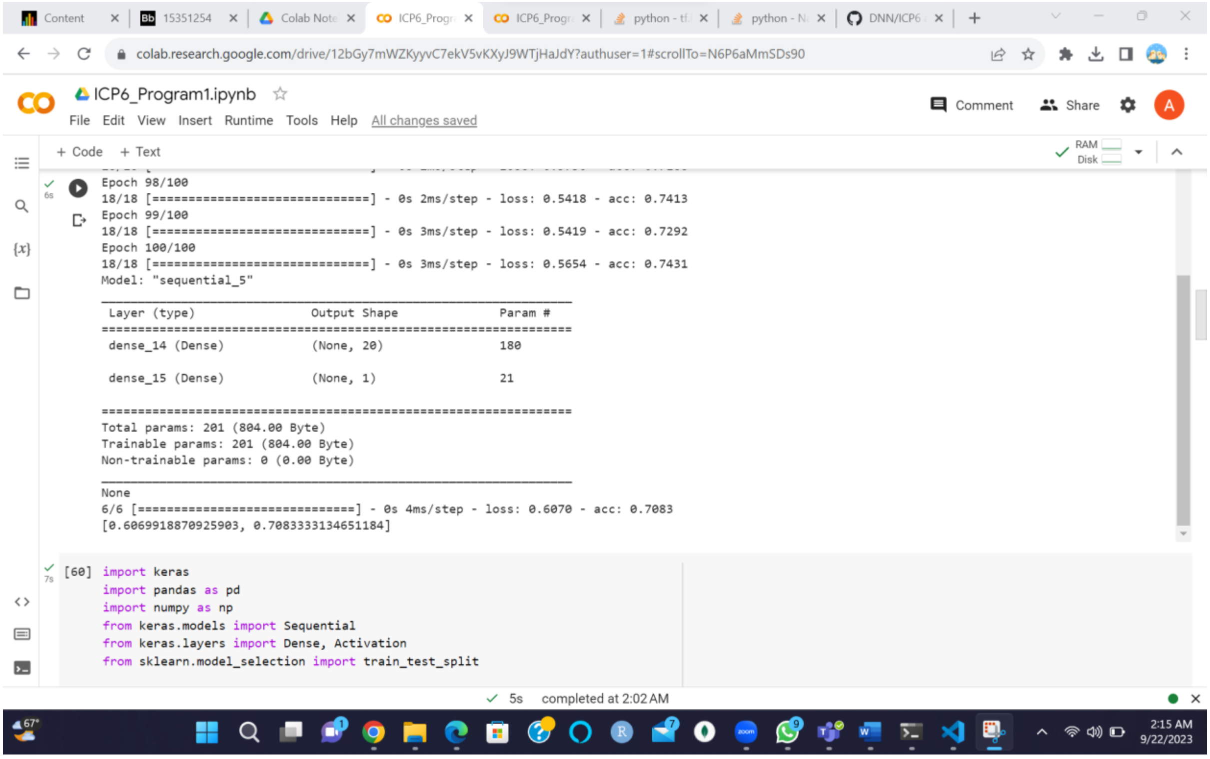
ICP- 6
Ushakiran Yadav Avula
700746114

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

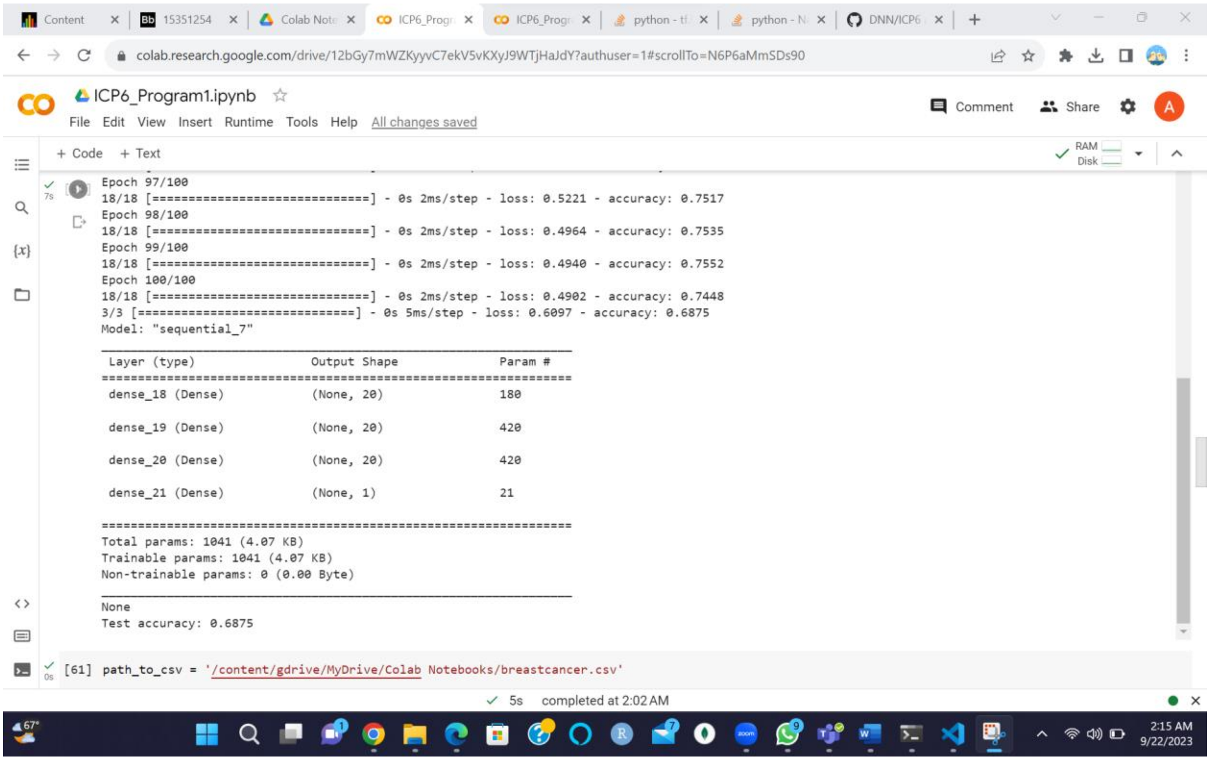
```
from google.colab import drive
drive.mount('/content/gdrive')
path_to_csv = '/content/gdrive/MyDrive/Colab Notebooks/diabetes.csv'
import keras
import pandas
from keras.models import Sequential
from keras.layers.core import Dense, Activation

# 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'))
```

```python
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])
```

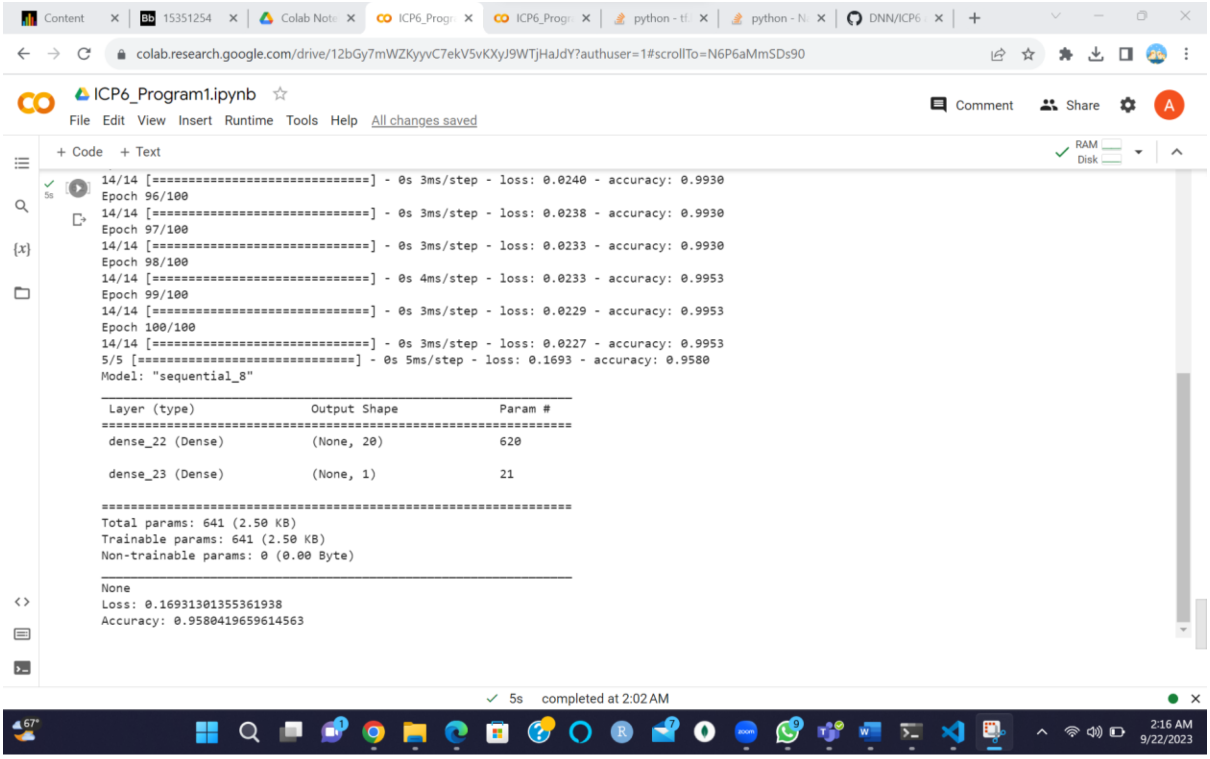


We added two more Dense layers with 20 nodes each in this version, both using the ReLU activation function. We kept the original code's batch size, optimizer, and loss function.

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()

```python
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)
```

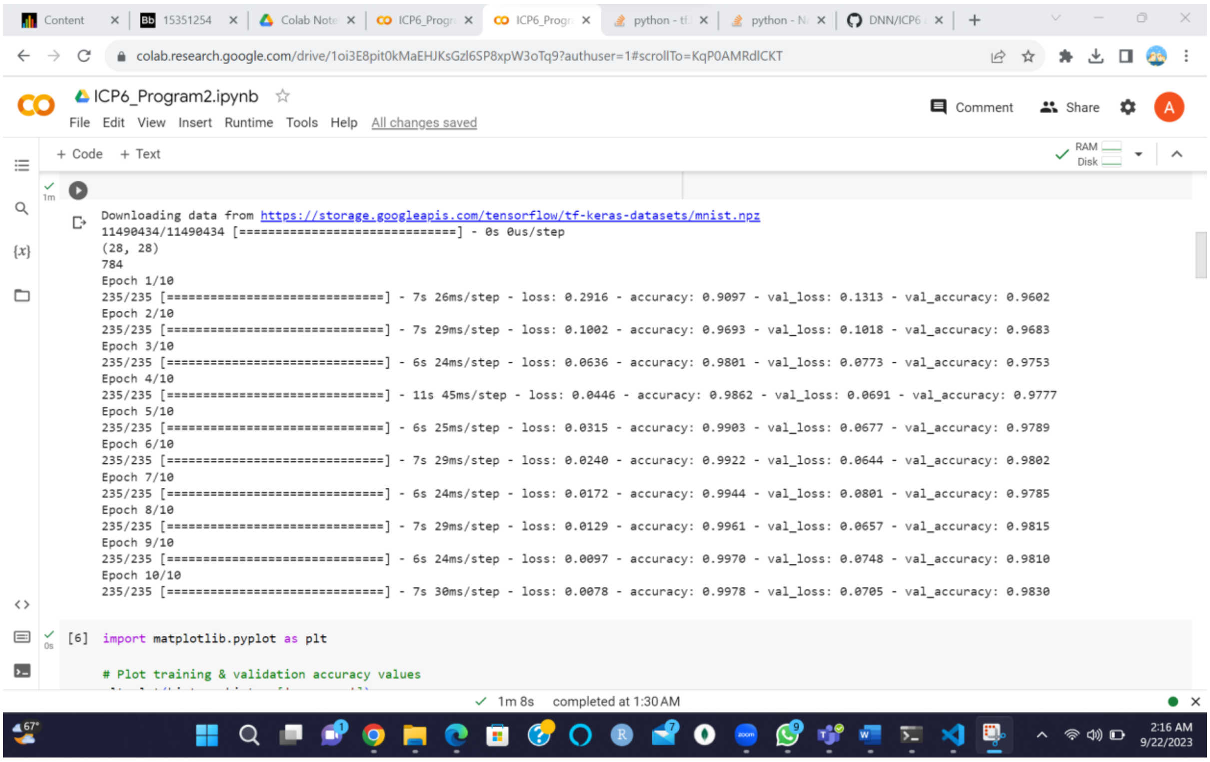


In the previous code, we used a dataset from the sklearn datasets, normalized the data, and generated the results.

2.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)
```

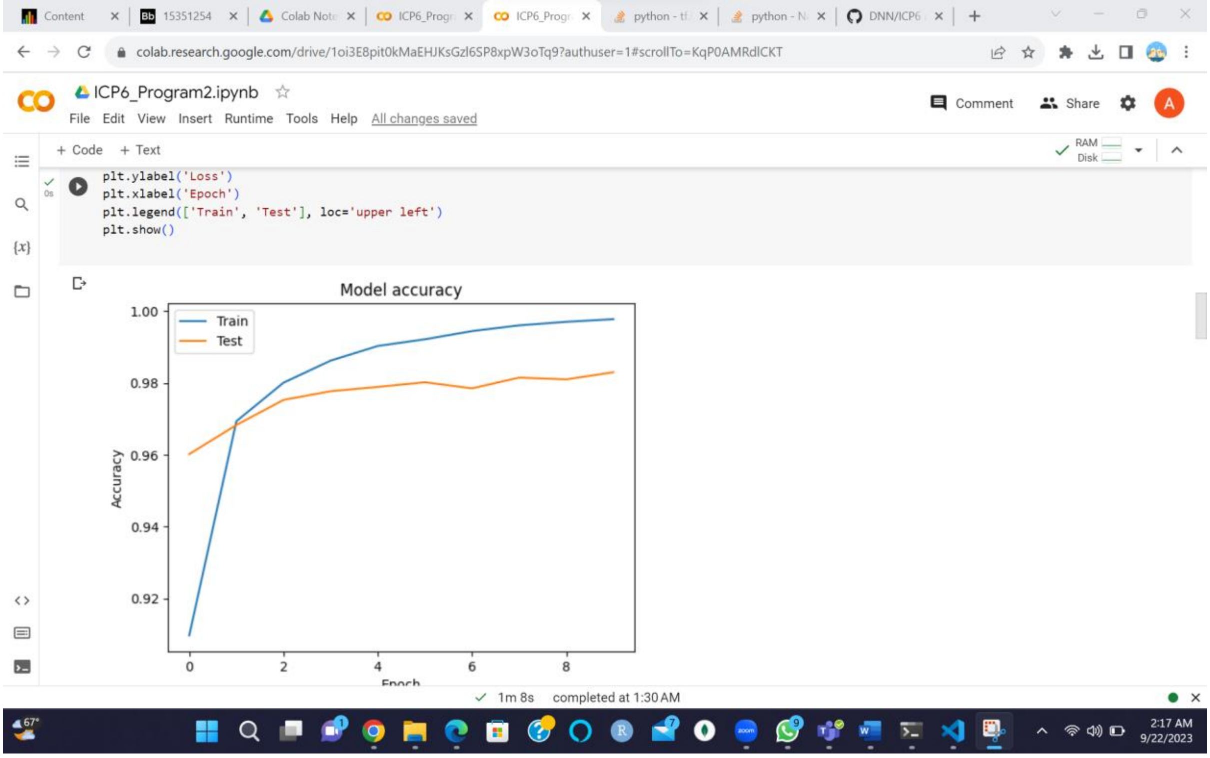
```
#convert data to float and scale values between 0 and 1
train_data = train_data.astype('float')
test_data = test_data.astype('float')
#scale data
train_data /=255.0
test_data /=255.0
#change the labels frominteger to one-hot encoding. to_categorical is
doing the same thing as LabelEncoder()
train_labels_one_hot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

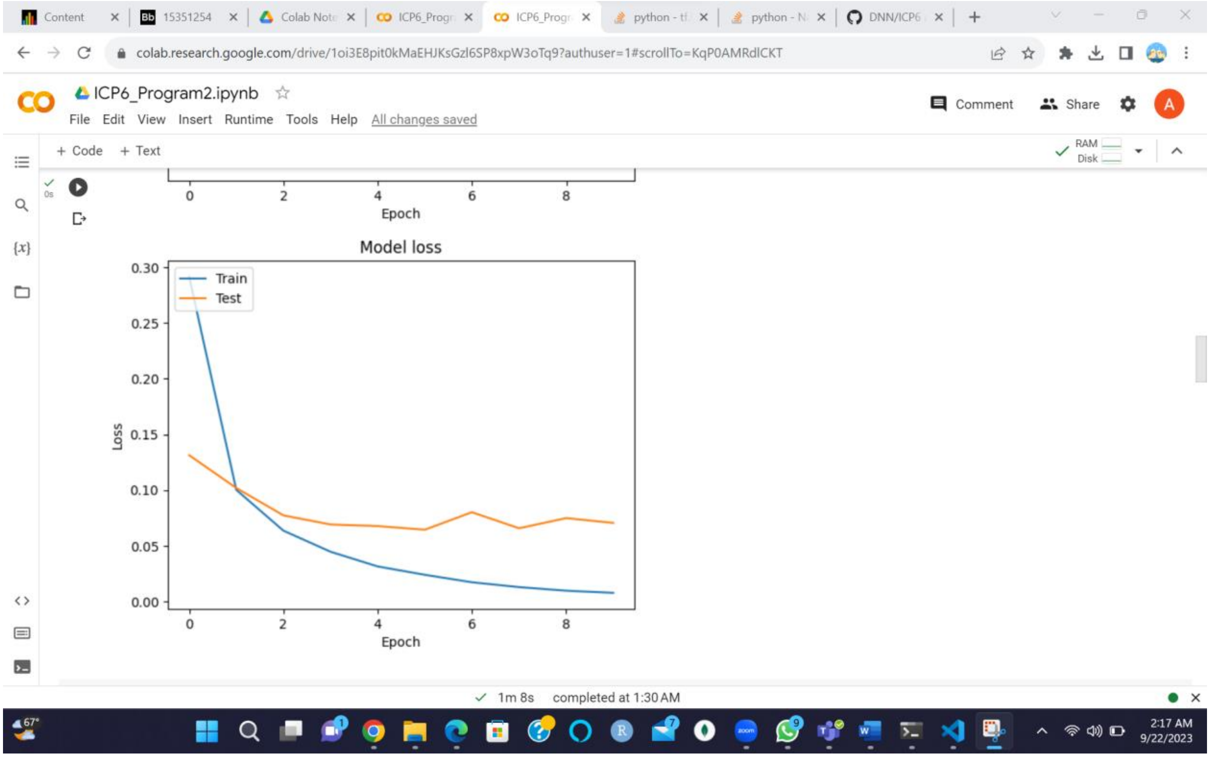


1. Plot the loss and accuracy for both training data and validation data using the history object in the source code.

```
import matplotlib.pyplot as plt
```

```python
# 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()
```

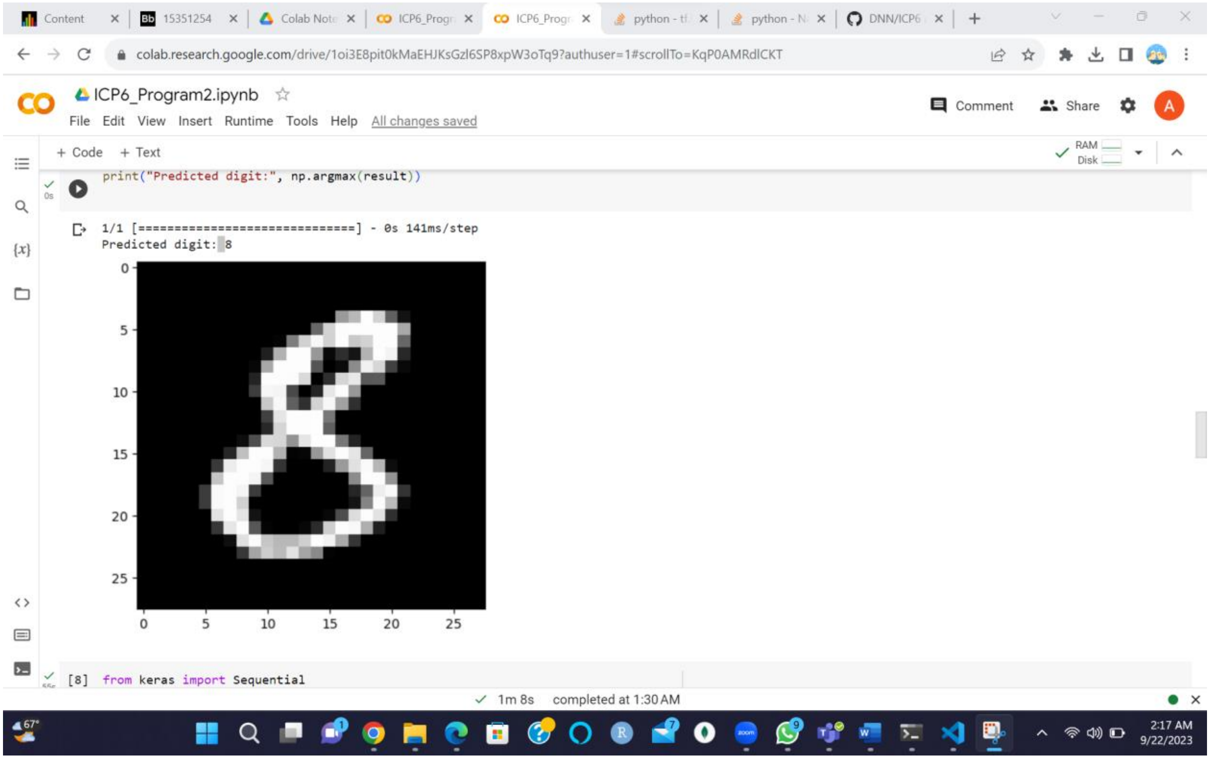




This code produced two plots: one for accuracy values and one for loss values. The first plot depicts training and validation accuracy curves over epochs, while the second depicts training and validation loss curves over epochs.

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.

```
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))
```

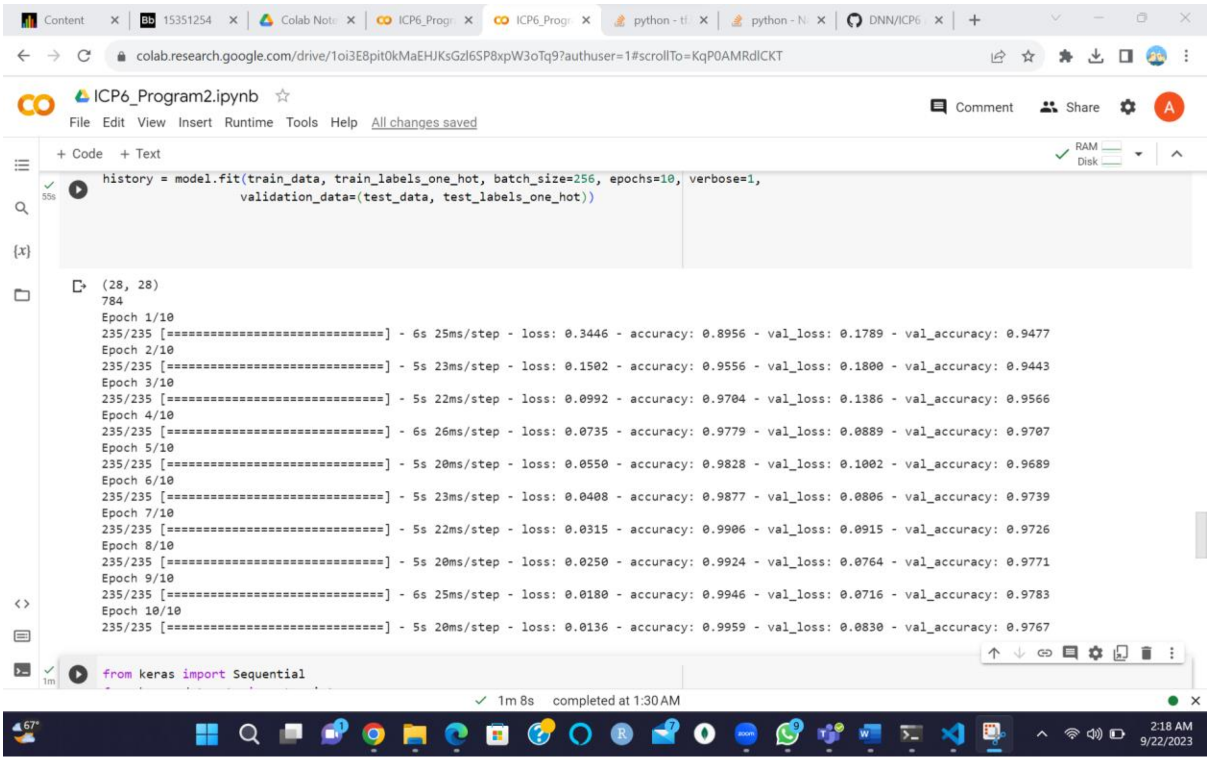


This will plot the image at index 1234 in the test data and then use the trained model to predict the digit in the image.

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.

```
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))
```

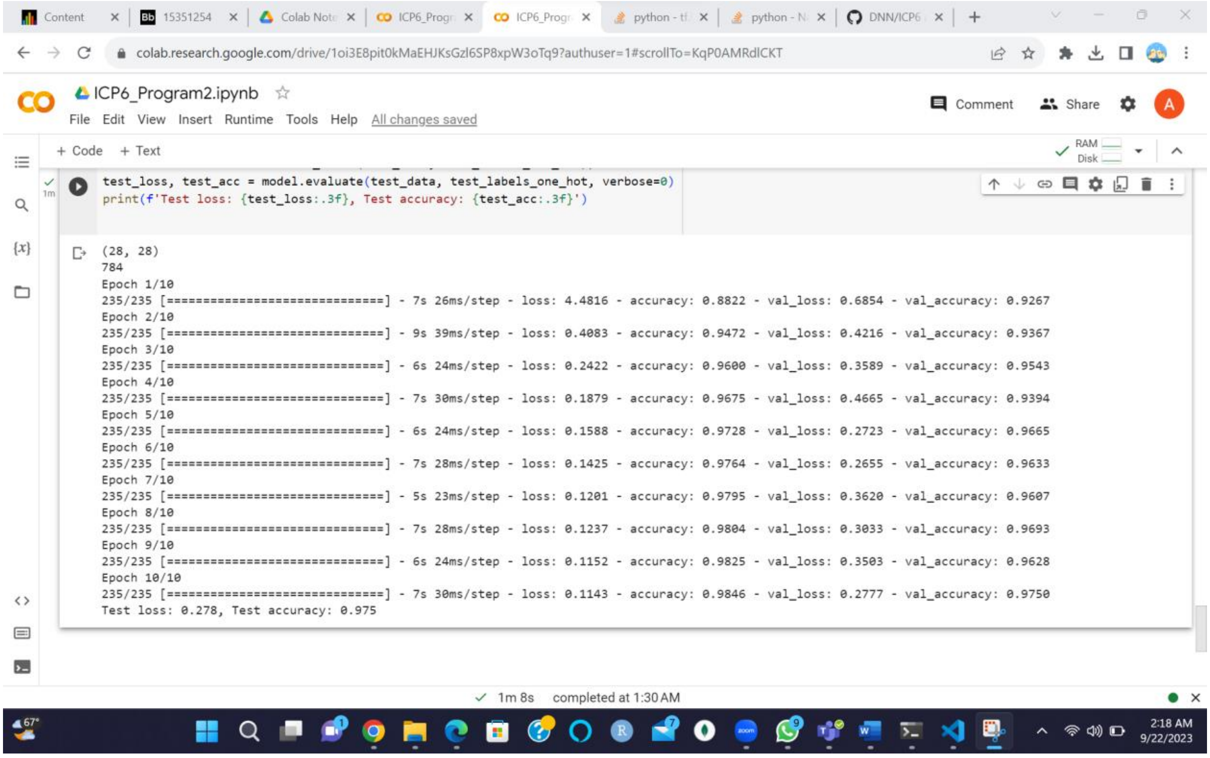


Here we are using the tanh function since we are using the tanh function the performance and accuracy may slightly vary

4. 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:
https://github.com/ushakiranyadav/ICP-6