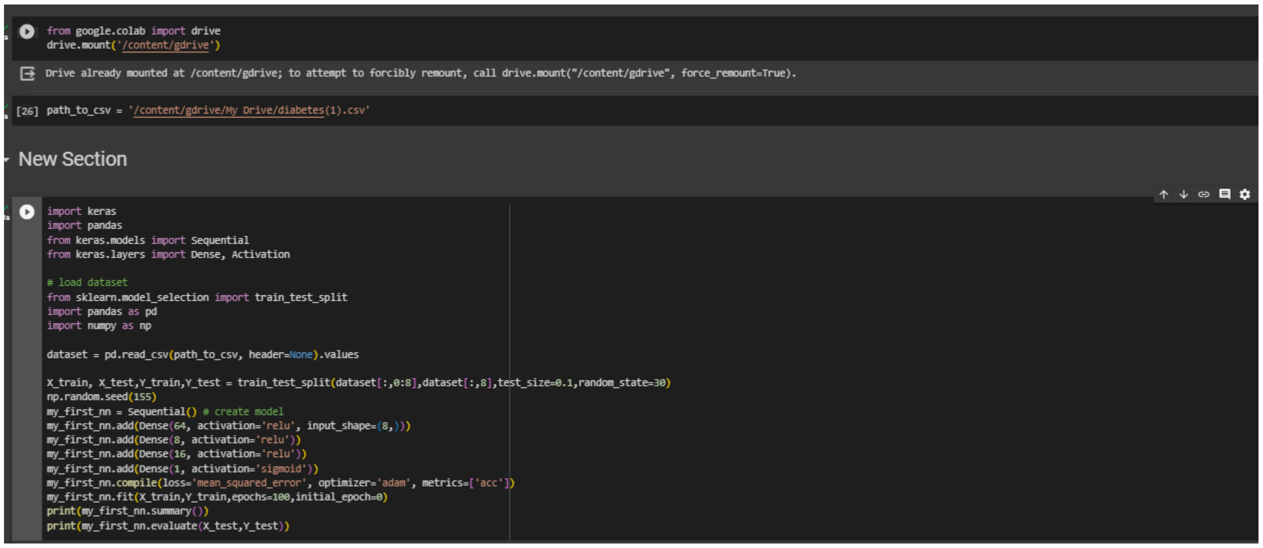
ICP-6 REPORT



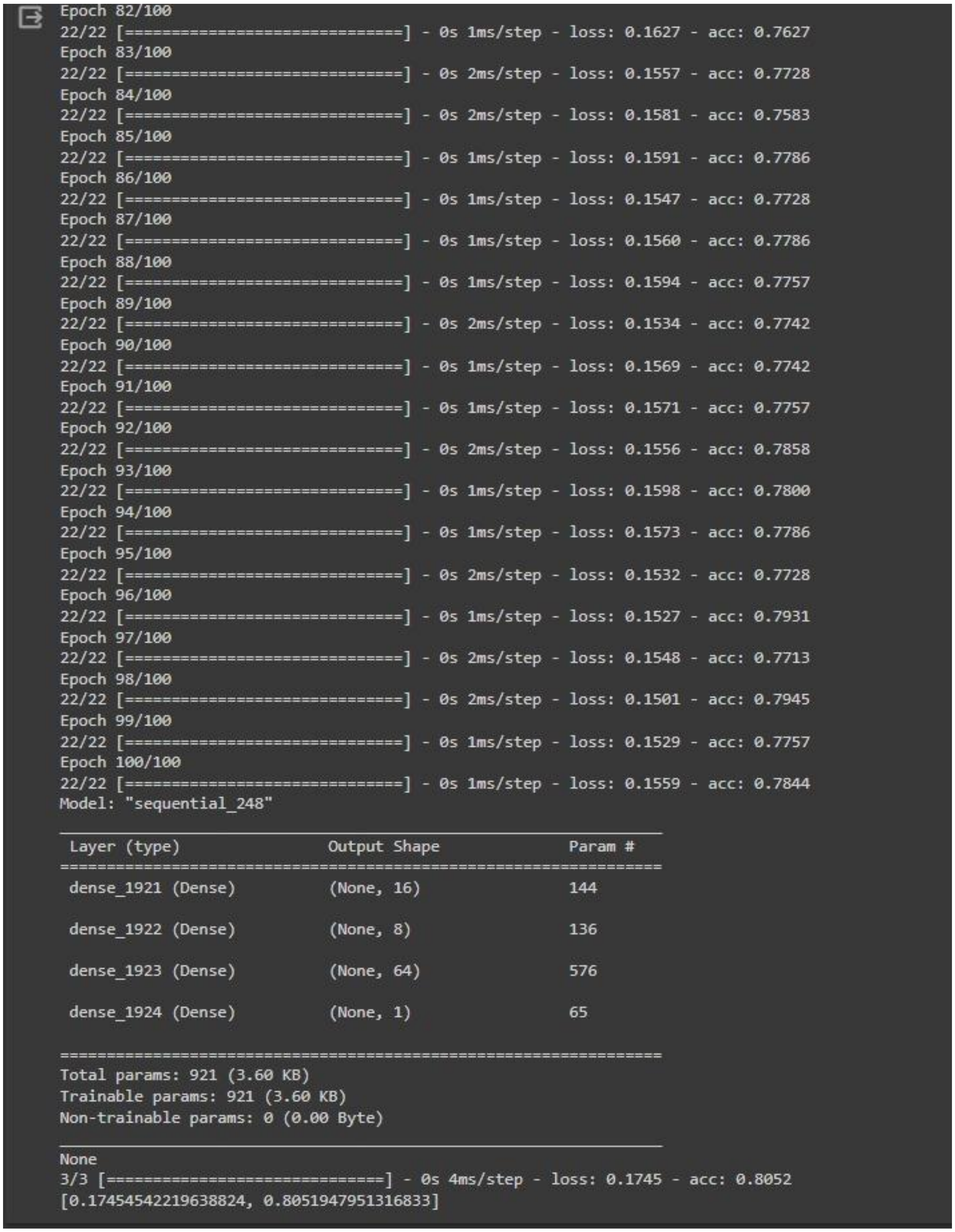
```
from google.colab import drive
drive.mount('/content/gdrive')
```

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

```
[26] path_to_csv = '/content/gdrive/My Drive/diabetes(1).csv'
```

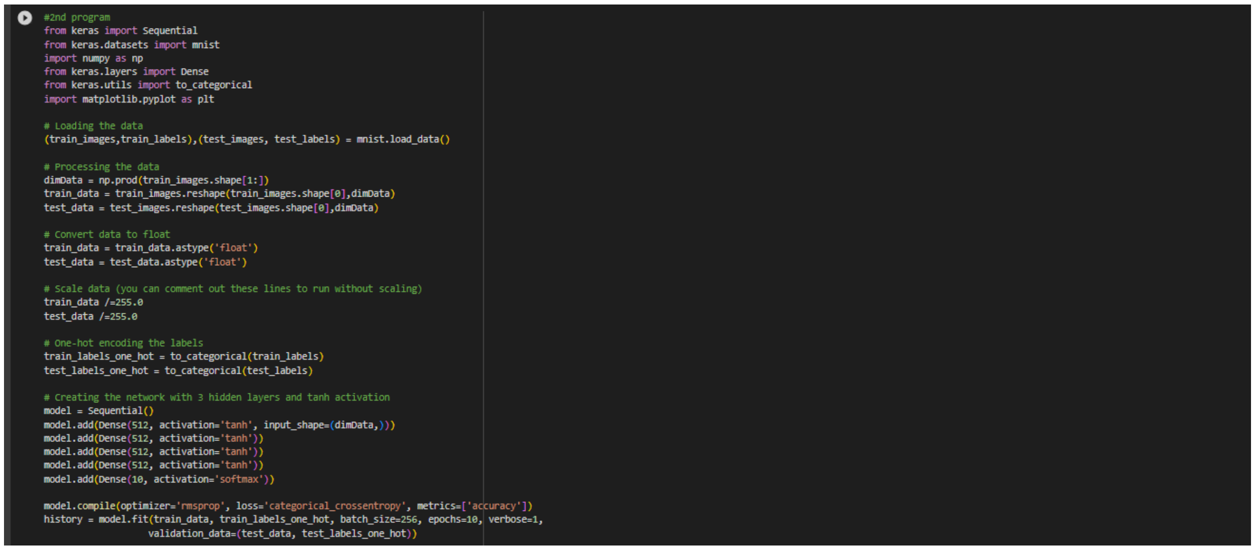
New Section

```
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(64, activation='relu', input_shape=(8,)))
my_first_nn.add(Dense(8, activation='relu'))
my_first_nn.add(Dense(16, 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))
```

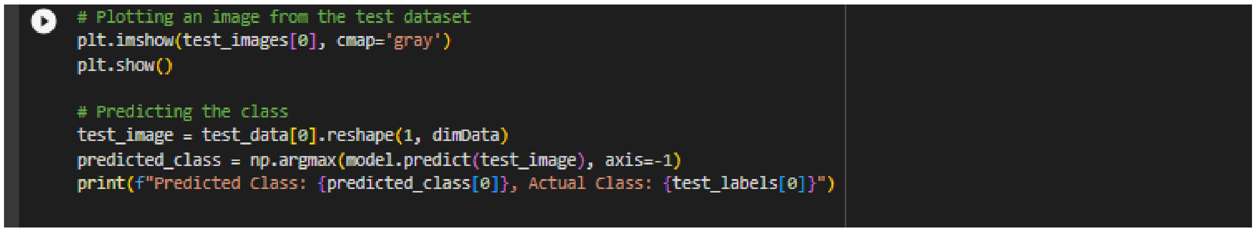


```
Epoch 82/100
22/22 [==============================] - 0s 1ms/step - loss: 0.1627 - acc: 0.7627
Epoch 83/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1557 - acc: 0.7728
Epoch 84/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1581 - acc: 0.7583
Epoch 85/100
22/22 [==============================] - 0s 1ms/step - loss: 0.1591 - acc: 0.7786
Epoch 86/100
22/22 [==============================] - 0s 1ms/step - loss: 0.1547 - acc: 0.7728
Epoch 87/100
22/22 [==============================] - 0s 1ms/step - loss: 0.1560 - acc: 0.7786
Epoch 88/100
22/22 [==============================] - 0s 1ms/step - loss: 0.1594 - acc: 0.7757
Epoch 89/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1534 - acc: 0.7742
Epoch 90/100
22/22 [==============================] - 0s 1ms/step - loss: 0.1569 - acc: 0.7742
Epoch 91/100
22/22 [==============================] - 0s 1ms/step - loss: 0.1571 - acc: 0.7757
Epoch 92/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1556 - acc: 0.7858
Epoch 93/100
22/22 [==============================] - 0s 1ms/step - loss: 0.1598 - acc: 0.7800
Epoch 94/100
22/22 [==============================] - 0s 1ms/step - loss: 0.1573 - acc: 0.7786
Epoch 95/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1532 - acc: 0.7728
Epoch 96/100
22/22 [==============================] - 0s 1ms/step - loss: 0.1527 - acc: 0.7931
Epoch 97/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1548 - acc: 0.7713
Epoch 98/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1501 - acc: 0.7945
Epoch 99/100
22/22 [==============================] - 0s 1ms/step - loss: 0.1529 - acc: 0.7757
Epoch 100/100
22/22 [==============================] - 0s 1ms/step - loss: 0.1559 - acc: 0.7844
Model: "sequential_248"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1921 (Dense)          (None, 16)                144

 dense_1922 (Dense)          (None, 8)                 136

 dense_1923 (Dense)          (None, 64)                576

 dense_1924 (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: 0.1745 - acc: 0.8052
[0.17454542219638824, 0.8051947951316833]
```

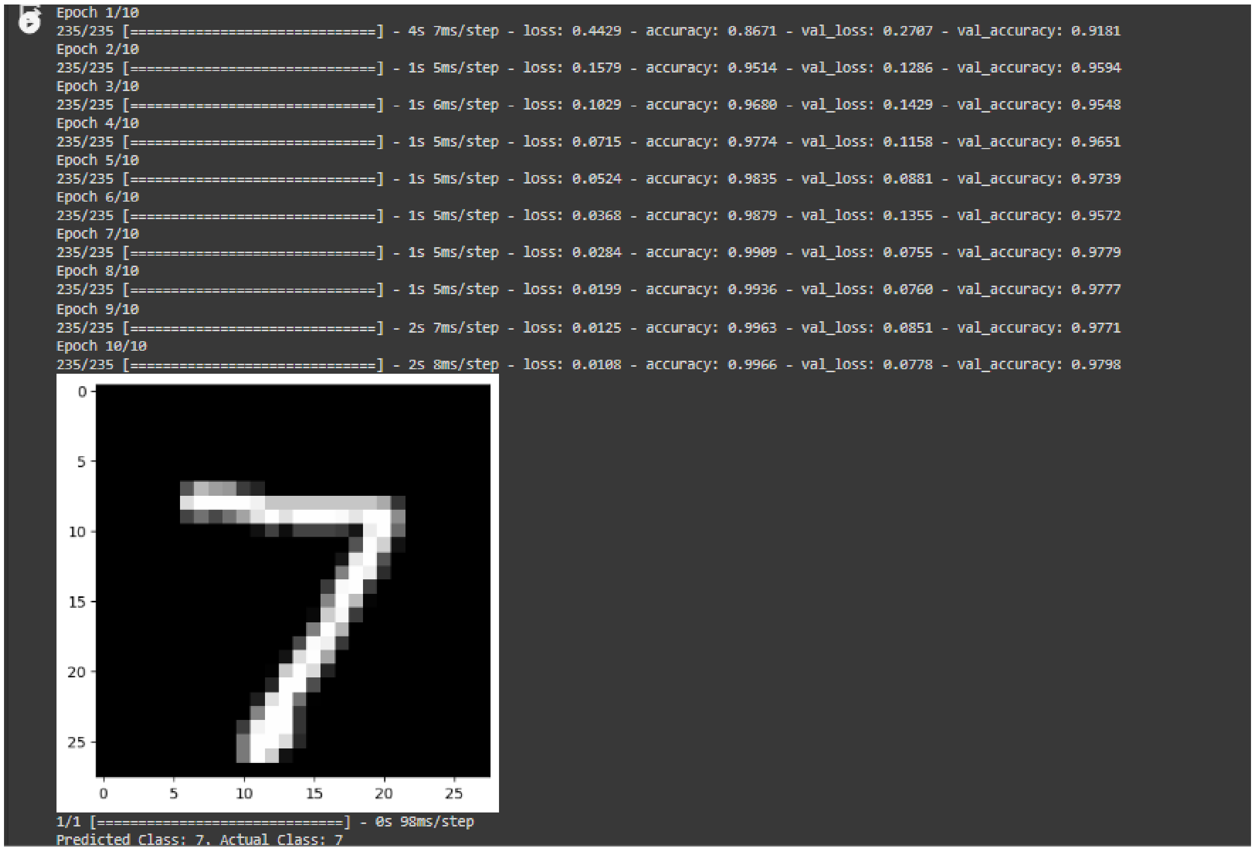
SECOND CODE:



```python
#2nd program
from keras import Sequential
from keras.datasets import mnist
import numpy as np
from keras.layers import Dense
from keras.utils import to_categorical
import matplotlib.pyplot as plt

# Loading the data
(train_images,train_labels),(test_images, test_labels) = mnist.load_data()

# Processing the data
dimData = np.prod(train_images.shape[1:])
train_data = train_images.reshape(train_images.shape[0],dimData)
test_data = test_images.reshape(test_images.shape[0],dimData)

# Convert data to float
train_data = train_data.astype('float')
test_data = test_data.astype('float')

# Scale data (you can comment out these lines to run without scaling)
train_data /=255.0
test_data /=255.0

# One-hot encoding the labels
train_labels_one_hot = to_categorical(train_labels)
test_labels_one_hot = to_categorical(test_labels)

# Creating the network with 3 hidden layers and tanh activation
model = Sequential()
model.add(Dense(512, activation='tanh', input_shape=(dimData,)))
model.add(Dense(512, activation='tanh'))
model.add(Dense(512, activation='tanh'))
model.add(Dense(512, 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))
```



```python
# Plotting an image from the test dataset
plt.imshow(test_images[0], cmap='gray')
plt.show()

# Predicting the class
test_image = test_data[0].reshape(1, dimData)
predicted_class = np.argmax(model.predict(test_image), axis=-1)
print(f"Predicted Class: {predicted_class[0]}, Actual Class: {test_labels[0]}")
```



```
Epoch 1/10
235/235 [==============================] - 4s 7ms/step - loss: 0.4429 - accuracy: 0.8671 - val_loss: 0.2707 - val_accuracy: 0.9181
Epoch 2/10
235/235 [==============================] - 1s 5ms/step - loss: 0.1579 - accuracy: 0.9514 - val_loss: 0.1286 - val_accuracy: 0.9594
Epoch 3/10
235/235 [==============================] - 1s 6ms/step - loss: 0.1029 - accuracy: 0.9680 - val_loss: 0.1429 - val_accuracy: 0.9548
Epoch 4/10
235/235 [==============================] - 1s 5ms/step - loss: 0.0715 - accuracy: 0.9774 - val_loss: 0.1158 - val_accuracy: 0.9651
Epoch 5/10
235/235 [==============================] - 1s 5ms/step - loss: 0.0524 - accuracy: 0.9835 - val_loss: 0.0881 - val_accuracy: 0.9739
Epoch 6/10
235/235 [==============================] - 1s 5ms/step - loss: 0.0368 - accuracy: 0.9879 - val_loss: 0.1355 - val_accuracy: 0.9572
Epoch 7/10
235/235 [==============================] - 1s 5ms/step - loss: 0.0284 - accuracy: 0.9909 - val_loss: 0.0755 - val_accuracy: 0.9779
Epoch 8/10
235/235 [==============================] - 1s 5ms/step - loss: 0.0199 - accuracy: 0.9936 - val_loss: 0.0760 - val_accuracy: 0.9777
Epoch 9/10
235/235 [==============================] - 2s 7ms/step - loss: 0.0125 - accuracy: 0.9963 - val_loss: 0.0851 - val_accuracy: 0.9771
Epoch 10/10
235/235 [==============================] - 2s 8ms/step - loss: 0.0108 - accuracy: 0.9966 - val_loss: 0.0778 - val_accuracy: 0.9798
```

```
1/1 [==============================] - 0s 98ms/step
Predicted Class: 7, Actual Class: 7
```

GITHUB REPO LINK:- https://github.com/sxk7912/Bigdata

YOUTUBE LINK:- https://youtu.be/K6fbQ2NzttQ