

NEURAL NETWORK & DEEP LEARNING

ASSIGNMENT 6

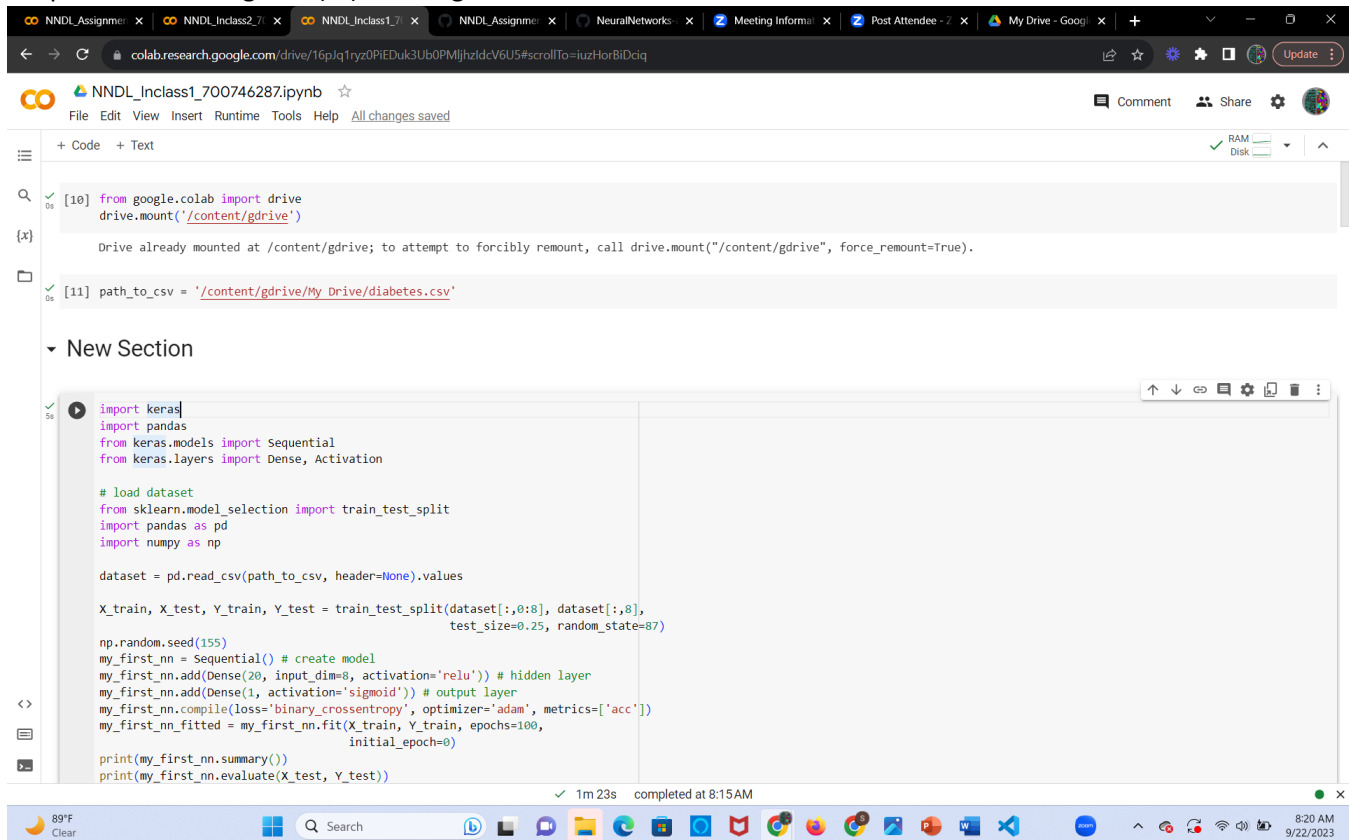
Name : SAI SNUSHA NAKKA

Student ID : 700746287

Git hub Link: https://github.com/NSnusha/NNDL_Assignment6

Video link: https://drive.google.com/file/d/1vAPMQuRB_fKzScIQ1fcu6ngx9wm-EO6R/view?usp=sharing

In class programming: 1. Use the use case in the class: a. Add more Dense layers to the existing code and check how the accuracy changes. 2. Change the data source to Breast Cancer dataset * available in the source code folder and make required changes. Report accuracy of the model. 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() Breast Cancer dataset is designated to predict if a patient has Malignant (M) or Benign = B cancer



```
[10] 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).

[11] path_to_csv = '/content/gdrive/My Drive/diabetes.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=0.25, 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))
```

```
colab.research.google.com/drive/16p1q1ryz0PIEDuk3Ub0PMIjhZdcV6U5#scrollTo=ch2yYnM0DsZ7

NNDL_Inclass1_700746287.ipynb
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

[12] 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=0.25, 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))

Epoch 80/100
18/18 [=====] - 0s 1ms/step - loss: 0.5493 - acc: 0.7292
Epoch 81/100
18/18 [=====] - 0s 1ms/step - loss: 0.5435 - acc: 0.7483
Epoch 82/100
18/18 [=====] - 0s 1ms/step - loss: 0.5443 - acc: 0.7344
Epoch 83/100
18/18 [=====] - 0s 1ms/step - loss: 0.5749 - acc: 0.7205
Epoch 84/100
18/18 [=====] - 0s 1ms/step - loss: 0.5580 - acc: 0.7222
Epoch 85/100
18/18 [=====] - 0s 1ms/step - loss: 0.5701 - acc: 0.7500
Epoch 86/100
18/18 [=====] - 0s 1ms/step - loss: 0.5662 - acc: 0.7153
Epoch 87/100
18/18 [=====] - 0s 2ms/step - loss: 0.5685 - acc: 0.7083
Epoch 88/100
18/18 [=====] - 0s 1ms/step - loss: 0.5740 - acc: 0.7101
Epoch 89/100
18/18 [=====] - 0s 2ms/step - loss: 0.5605 - acc: 0.7517
Epoch 90/100
18/18 [=====] - 0s 1ms/step - loss: 0.5470 - acc: 0.7465
Epoch 91/100
18/18 [=====] - 0s 2ms/step - loss: 0.5558 - acc: 0.7222
Epoch 92/100
18/18 [=====] - 0s 1ms/step - loss: 0.5724 - acc: 0.7274
Epoch 93/100
18/18 [=====] - 0s 1ms/step - loss: 0.5625 - acc: 0.7274
Epoch 94/100
18/18 [=====] - 0s 1ms/step - loss: 0.5712 - acc: 0.7014
Epoch 95/100
18/18 [=====] - 0s 2ms/step - loss: 0.5708 - acc: 0.7170
Epoch 96/100
18/18 [=====] - 0s 1ms/step - loss: 0.5502 - acc: 0.7222
Epoch 97/100
18/18 [=====] - 0s 1ms/step - loss: 0.5470 - acc: 0.7309
Epoch 98/100
18/18 [=====] - 0s 1ms/step - loss: 0.5446 - acc: 0.7274
Epoch 99/100
18/18 [=====] - 0s 1ms/step - loss: 0.5543 - acc: 0.7240
Epoch 100/100
18/18 [=====] - 0s 1ms/step - loss: 0.5657 - acc: 0.7305
Model: "sequential_4"
Layer (type) Output Shape Param #
-----
dense_10 (Dense) (None, 20) 180
dense_11 (Dense) (None, 1) 21
-----
Total params: 201 (804.00 Byte)
Trainable params: 201 (804.00 Byte)
Non-trainable params: 0 (0.00 Byte)
None
6/6 [=====] - 0s 2ms/step - loss: 0.6187 - acc: 0.7135
[0.6186739802360535, 0.7135416865348816]
```

```
NNDL_Inclass1_700746287.ipynb
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

[12] Epoch 82/100
18/18 [=====] - 0s 1ms/step - loss: 0.5443 - acc: 0.7344
Epoch 83/100
18/18 [=====] - 0s 1ms/step - loss: 0.5749 - acc: 0.7205
Epoch 84/100
18/18 [=====] - 0s 1ms/step - loss: 0.5580 - acc: 0.7222
Epoch 85/100
18/18 [=====] - 0s 1ms/step - loss: 0.5701 - acc: 0.7500
Epoch 86/100
18/18 [=====] - 0s 1ms/step - loss: 0.5662 - acc: 0.7153
Epoch 87/100
18/18 [=====] - 0s 2ms/step - loss: 0.5685 - acc: 0.7083
Epoch 88/100
18/18 [=====] - 0s 1ms/step - loss: 0.5740 - acc: 0.7101
Epoch 89/100
18/18 [=====] - 0s 2ms/step - loss: 0.5605 - acc: 0.7517
Epoch 90/100
18/18 [=====] - 0s 1ms/step - loss: 0.5470 - acc: 0.7465
Epoch 91/100
18/18 [=====] - 0s 2ms/step - loss: 0.5558 - acc: 0.7222
Epoch 92/100
18/18 [=====] - 0s 1ms/step - loss: 0.5724 - acc: 0.7274
Epoch 93/100
18/18 [=====] - 0s 1ms/step - loss: 0.5625 - acc: 0.7274
Epoch 94/100
18/18 [=====] - 0s 1ms/step - loss: 0.5712 - acc: 0.7014
Epoch 95/100
18/18 [=====] - 0s 2ms/step - loss: 0.5708 - acc: 0.7170
Epoch 96/100
18/18 [=====] - 0s 1ms/step - loss: 0.5502 - acc: 0.7222
Epoch 97/100
18/18 [=====] - 0s 1ms/step - loss: 0.5470 - acc: 0.7309
Epoch 98/100
18/18 [=====] - 0s 1ms/step - loss: 0.5446 - acc: 0.7274
Epoch 99/100
18/18 [=====] - 0s 1ms/step - loss: 0.5543 - acc: 0.7240
Epoch 100/100
18/18 [=====] - 0s 1ms/step - loss: 0.5657 - acc: 0.7305
Model: "sequential_4"
Layer (type) Output Shape Param #
-----
dense_10 (Dense) (None, 20) 180
dense_11 (Dense) (None, 1) 21
-----
Total params: 201 (804.00 Byte)
Trainable params: 201 (804.00 Byte)
Non-trainable params: 0 (0.00 Byte)
None
6/6 [=====] - 0s 2ms/step - loss: 0.6187 - acc: 0.7135
[0.6186739802360535, 0.7135416865348816]
```

colab.research.google.com/drive/16p1q1ryz0PIEDuk3Ub0PMIjhZdcV6U5#scrollTo=ch2yYnM0Dsz7

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

[13]

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 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(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 8us/step
(28, 28)
784
Epoch 1/10
235/235 [=====] - 7s 27ms/step - loss: 0.2922 - accuracy: 0.9101 - val_loss: 0.1490 - val_accuracy: 0.9538
Epoch 2/10
235/235 [=====] - 4s 17ms/step - loss: 0.1006 - accuracy: 0.9680 - val_loss: 0.0892 - val_accuracy: 0.9712
Epoch 3/10
235/235 [=====] - 4s 17ms/step - loss: 0.0643 - accuracy: 0.9798 - val_loss: 0.0775 - val_accuracy: 0.9758
Epoch 4/10
235/235 [=====] - 5s 19ms/step - loss: 0.0442 - accuracy: 0.9865 - val_loss: 0.0702 - val_accuracy: 0.9784
Epoch 5/10
235/235 [=====] - 4s 18ms/step - loss: 0.0322 - accuracy: 0.9898 - val_loss: 0.0741 - val_accuracy: 0.9771
Epoch 6/10
235/235 [=====] - 4s 17ms/step - loss: 0.0226 - accuracy: 0.9929 - val_loss: 0.0678 - val_accuracy: 0.9793
Epoch 7/10

1m 23s completed at 8:15 AM

colab.research.google.com/drive/16p1q1ryz0PIEDuk3Ub0PMIjhZdcV6U5#scrollTo=ch2yYnM0Dsz7

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

[5]

import keras
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers 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])

Epoch 02/100
18/18 [=====] - 0s 2ms/step - loss: 0.4574 - accuracy: 0.7899
Epoch 03/100
18/18 [=====] - 0s 1ms/step - loss: 0.4588 - accuracy: 0.7917
Epoch 04/100
18/18 [=====] - 0s 1ms/step - loss: 0.4418 - accuracy: 0.8090
Epoch 05/100
18/18 [=====] - 0s 1ms/step - loss: 0.4730 - accuracy: 0.7656
Epoch 06/100
18/18 [=====] - 0s 2ms/step - loss: 0.4700 - accuracy: 0.7917
Epoch 07/100
18/18 [=====] - 0s 1ms/step - loss: 0.4366 - accuracy: 0.7917
Epoch 08/100
18/18 [=====] - 0s 1ms/step - loss: 0.4427 - accuracy: 0.8021
Epoch 09/100
18/18 [=====] - 0s 1ms/step - loss: 0.4364 - accuracy: 0.8090
Epoch 90/100
18/18 [=====] - 0s 2ms/step - loss: 0.4646 - accuracy: 0.7865

1m 23s completed at 8:15 AM

Colab interface showing a Jupyter Notebook titled "NNDL_Inclass1_700746287.ipynb". The notebook displays the output of a Keras model training process. The output shows the progress of training over 100 epochs, with loss and accuracy metrics. The final test accuracy is 0.703125.

```
Epoch 83/100 - 0s 1ms/step - loss: 0.4588 - accuracy: 0.7917
Epoch 84/100
Epoch 85/100 - 0s 1ms/step - loss: 0.4418 - accuracy: 0.8090
Epoch 86/100 - 0s 1ms/step - loss: 0.4730 - accuracy: 0.7656
Epoch 87/100 - 0s 2ms/step - loss: 0.4700 - accuracy: 0.7917
Epoch 88/100 - 0s 1ms/step - loss: 0.4366 - accuracy: 0.7917
Epoch 89/100 - 0s 1ms/step - loss: 0.4427 - accuracy: 0.8021
Epoch 90/100 - 0s 1ms/step - loss: 0.4364 - accuracy: 0.8090
Epoch 91/100 - 0s 2ms/step - loss: 0.4646 - accuracy: 0.7865
Epoch 92/100 - 0s 1ms/step - loss: 0.4716 - accuracy: 0.7969
Epoch 93/100 - 0s 1ms/step - loss: 0.4441 - accuracy: 0.8056
Epoch 94/100 - 0s 1ms/step - loss: 0.4399 - accuracy: 0.8038
Epoch 95/100 - 0s 1ms/step - loss: 0.4344 - accuracy: 0.8073
Epoch 96/100 - 0s 2ms/step - loss: 0.4432 - accuracy: 0.7951
Epoch 97/100 - 0s 1ms/step - loss: 0.4695 - accuracy: 0.7812
Epoch 98/100 - 0s 1ms/step - loss: 0.4805 - accuracy: 0.7830
Epoch 99/100 - 0s 1ms/step - loss: 0.4596 - accuracy: 0.7899
Epoch 100/100 - 0s 2ms/step - loss: 0.4439 - accuracy: 0.8073
3/3 [=====] - 0s 1ms/step - loss: 0.4518 - accuracy: 0.7899
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
dense_2 (Dense)	(None, 20)	180
dense_3 (Dense)	(None, 20)	420
dense_4 (Dense)	(None, 20)	420
dense_5 (Dense)	(None, 1)	21

Total params: 1041 (4.07 KB)
Trainable params: 1041 (4.07 KB)
Non-trainable params: 0 (0.00 Byte)

Test accuracy: 0.703125

Colab interface showing a Jupyter Notebook titled "NNDL_Inclass1_700746287.ipynb". The notebook displays the code for loading and training a Keras model on breast cancer data. The code includes imports for pandas, numpy, sklearn, and keras, followed by data loading, preprocessing, model creation, training, and evaluation.

```
path_to_csv = '/content/gdrive/MyDrive/breastcancer.csv'
```

```
[7] path_to_csv = '/content/gdrive/MyDrive/breastcancer.csv'
```

```
[8] 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)

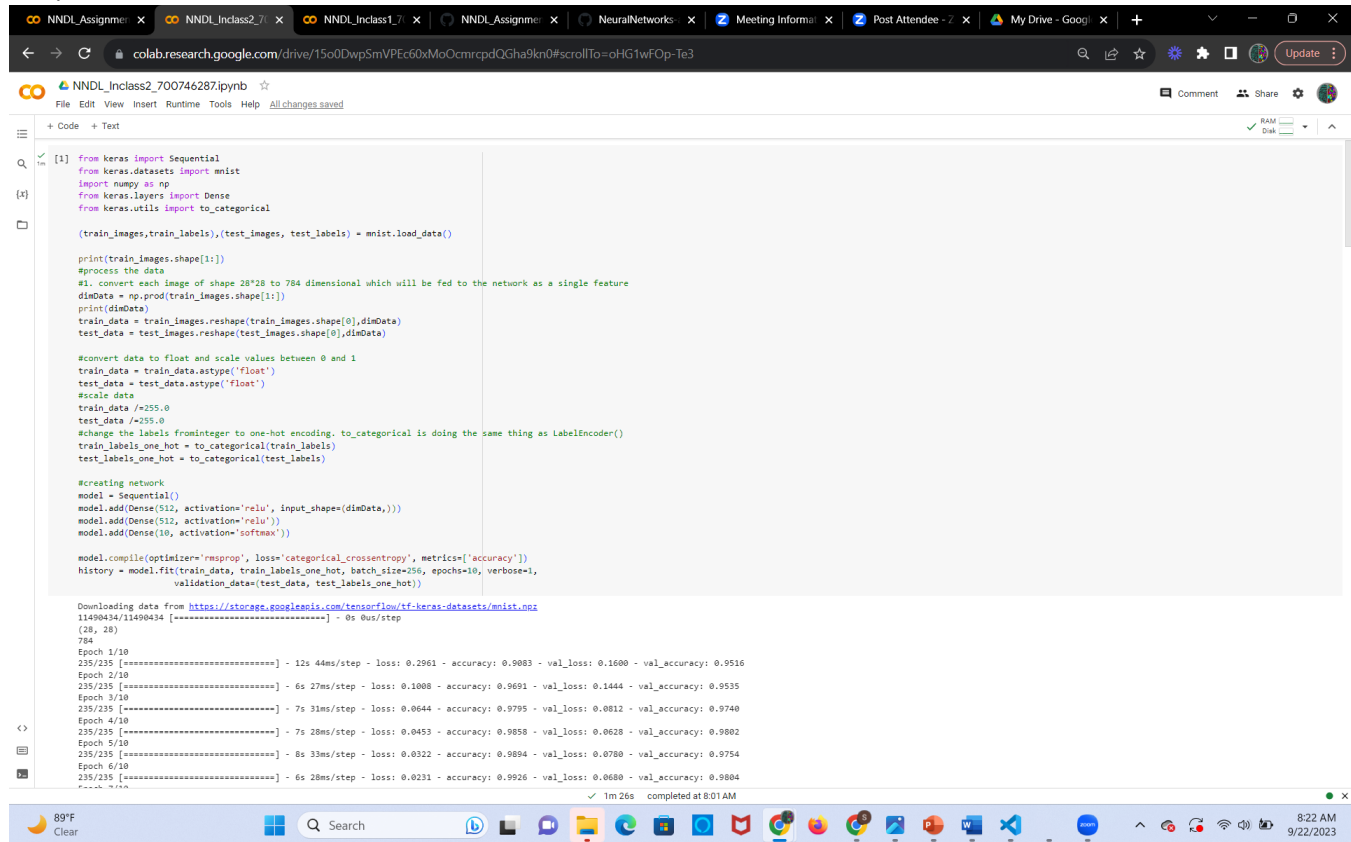
# Create model
np.random.seed(155)
model = Sequential()
model.add(Dense(20, input_dim=30, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# 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)
```

```
14/14 [=====] - 0s 3ms/step - loss: 0.0395 - accuracy: 0.9906
Epoch 53/100
14/14 [=====] - 0s 1ms/step - loss: 0.0389 - accuracy: 0.9906
Epoch 54/100
14/14 [=====] - 0s 1ms/step - loss: 0.0383 - accuracy: 0.9930
Epoch 55/100 - 0s 2ms/step - loss: 0.0379 - accuracy: 0.9906
Epoch 56/100 - 0s 2ms/step - loss: 0.0373 - accuracy: 0.9906
Epoch 57/100 - 0s 1ms/step - loss: 0.0365 - accuracy: 0.9930
Epoch 58/100
```

2. In class programming: Use Image Classification on the hand written digits data set (mnist) 1. Plot the loss and accuracy for both training data and validation data using the history object in the source code. 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. 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. 4. Run the same code without scaling the images and check the performance?



```
[1]: 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()

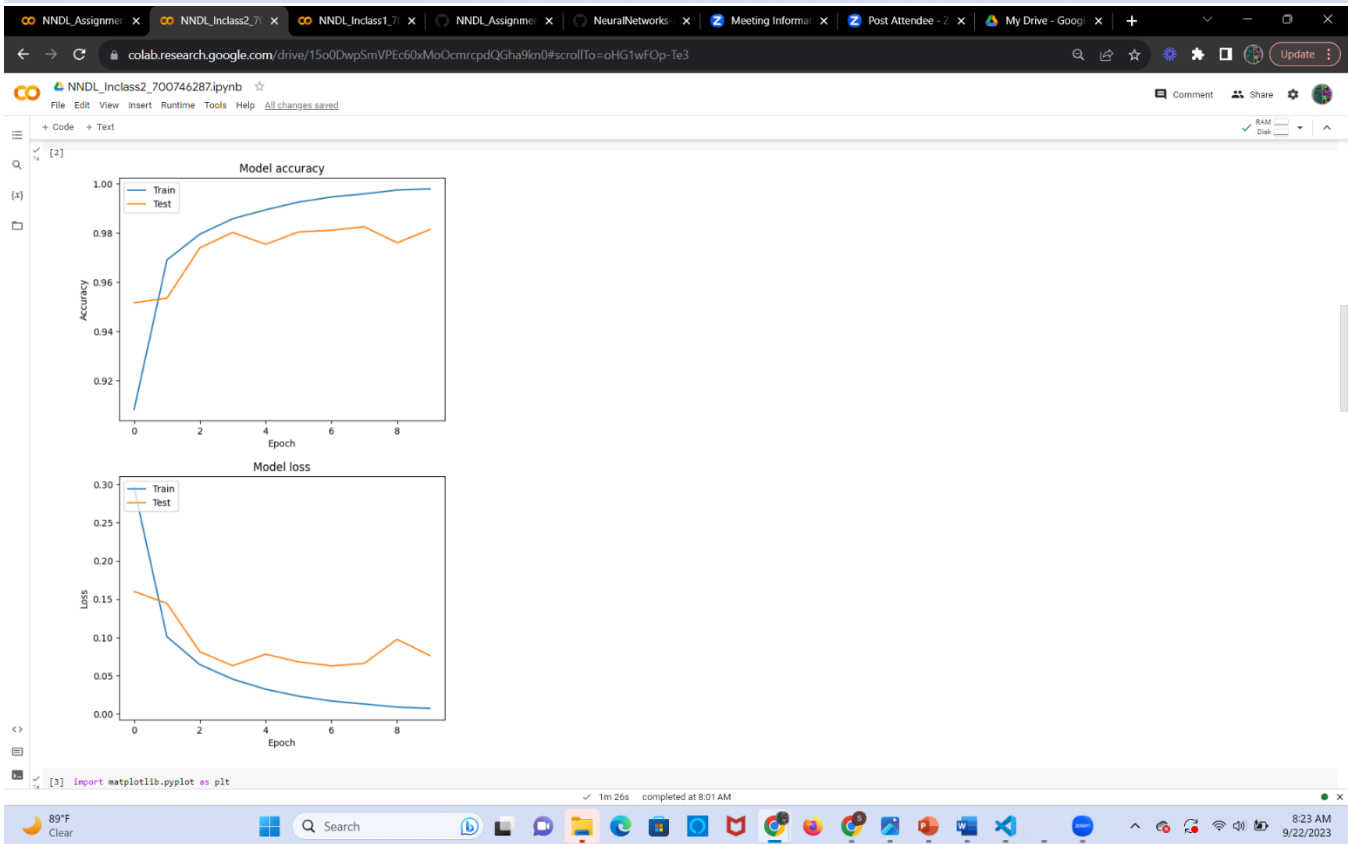
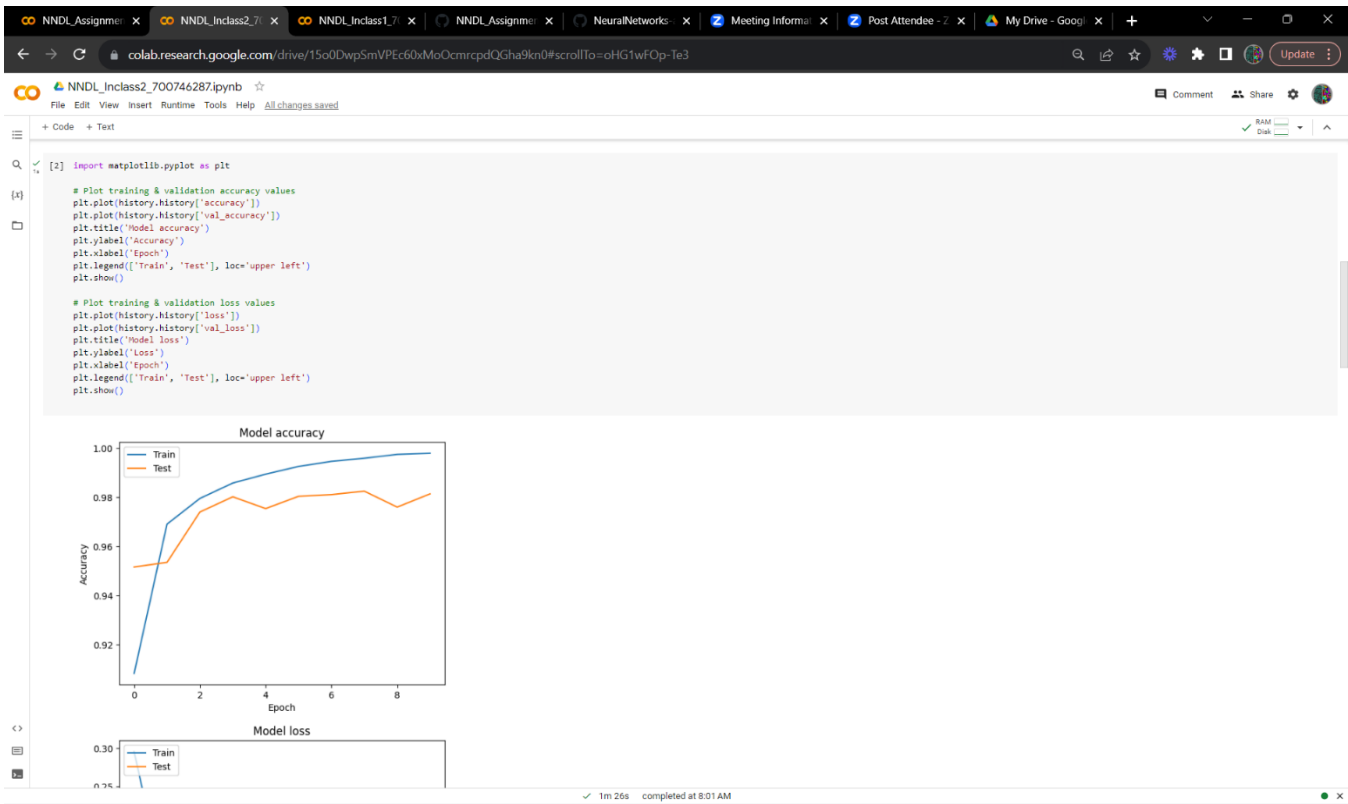
print(train_images.shape[1:])
#process the data
#1. convert each image of shape 28*28 to 784 dimensional which will be fed to the network as a single feature
dimData = np.prod(train_images.shape[1:])
print(dimData)
train_data = train_images.reshape(train_images.shape[0], dimData)
test_data = test_images.reshape(test_images.shape[0], 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 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(10, activation='softmax'))

model.compile(optimizer='nadam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(train_data, train_labels_one_hot, batch_size=64, 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
784
Epoch 1/10
235/235 [=====] - 12s 44ms/step - loss: 0.2961 - accuracy: 0.9083 - val_loss: 0.1600 - val_accuracy: 0.9516
Epoch 2/10
235/235 [=====] - 6s 27ms/step - loss: 0.1008 - accuracy: 0.9691 - val_loss: 0.1444 - val_accuracy: 0.9535
Epoch 3/10
235/235 [=====] - 7s 31ms/step - loss: 0.0644 - accuracy: 0.9795 - val_loss: 0.0812 - val_accuracy: 0.9740
Epoch 4/10
235/235 [=====] - 7s 28ms/step - loss: 0.0453 - accuracy: 0.9858 - val_loss: 0.0628 - val_accuracy: 0.9802
Epoch 5/10
235/235 [=====] - 8s 33ms/step - loss: 0.0322 - accuracy: 0.9894 - val_loss: 0.0780 - val_accuracy: 0.9754
Epoch 6/10
235/235 [=====] - 6s 28ms/step - loss: 0.0231 - accuracy: 0.9926 - val_loss: 0.0680 - val_accuracy: 0.9804
1m 26s completed at 8:01 AM
```



colab.research.google.com/drive/15o0DwpSmVPEc60xMoOcmrcpdQGha9kn0#scrollTo=oHG1wFOp-Te3

File Edit View Insert Runtime Tools Help All changes saved

Code Text

0.00

Epoch

[2]

[3]

```
import matplotlib.pyplot as plt

# select a random image from test data
image_index = 1234
img = test_images[image_index]

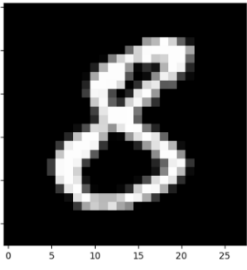
# plot the image
plt.imshow(img, cmap='gray')

# 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))
```

1/1 [=====] - 0s 192ms/step
Predicted digit: 8



[4]

```
from keras import Sequential
from keras.datasets import mnist
```

1m 26s completed at 8:01 AM

colab.research.google.com/drive/15o0DwpSmVPEc60xMoOcmrcpdQGha9kn0#scrollTo=oHG1wFOp-Te3

File Edit View Insert Runtime Tools Help All changes saved

Code Text

[4]

```
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 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='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=32, epochs=10, verbose=1,
                    validation_data=(test_data, test_labels_one_hot))
```

(28, 28)
784
Epoch 1/10
235/235 [=====] - 11s 39ms/step - loss: 0.3381 - accuracy: 0.8973 - val_loss: 0.2139 - val_accuracy: 0.9368
Epoch 2/10
235/235 [=====] - 6s 24ms/step - loss: 0.1461 - accuracy: 0.9560 - val_loss: 0.1632 - val_accuracy: 0.9495
Epoch 3/10
235/235 [=====] - 6s 27ms/step - loss: 0.0968 - accuracy: 0.9788 - val_loss: 0.1189 - val_accuracy: 0.9630
Epoch 4/10
235/235 [=====] - 6s 27ms/step - loss: 0.0694 - accuracy: 0.9788 - val_loss: 0.0963 - val_accuracy: 0.9697
Epoch 5/10
235/235 [=====] - 6s 25ms/step - loss: 0.0521 - accuracy: 0.9842 - val_loss: 0.0796 - val_accuracy: 0.9766
Epoch 6/10
235/235 [=====] - 7s 28ms/step - loss: 0.0379 - accuracy: 0.9889 - val_loss: 0.1094 - val_accuracy: 0.9641
Epoch 7/10

1m 26s completed at 8:01 AM

colab.research.google.com/drive/15o0DwpSmVPEc60xMoOcmrpdQGha9kn0#scrollTo=oHG1wFOP-Te3

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
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 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(10, activation='softmax'))

model.compile(optimizer='nadam', 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('Test loss: {test_loss:.3f}, Test accuracy: {test_acc:.3f}')
```

(28, 28)

784

Epoch 1/10

235/235 [-----] - 9s 36ms/step - loss: 7.1491 - accuracy: 0.8677 - val_loss: 0.7785 - val_accuracy: 0.9087

Epoch 2/10

235/235 [-----] - 7s 30ms/step - loss: 0.4086 - accuracy: 0.9458 - val_loss: 0.6582 - val_accuracy: 0.9045

Epoch 3/10

235/235 [-----] - 7s 31ms/step - loss: 0.2493 - accuracy: 0.9581 - val_loss: 0.2749 - val_accuracy: 0.9565

Epoch 4/10

235/235 [-----] - 8s 35ms/step - loss: 0.1989 - accuracy: 0.9678 - val_loss: 0.4394 - val_accuracy: 0.9549

Epoch 5/10

235/235 [-----] - 7s 28ms/step - loss: 0.1865 - accuracy: 0.9710 - val_loss: 0.3683 - val_accuracy: 0.9534

Epoch 6/10

235/235 [-----] - 8s 33ms/step - loss: 0.1456 - accuracy: 0.9756 - val_loss: 0.3580 - val_accuracy: 0.9586

Epoch 7/10

235/235 [-----] - 7s 28ms/step - loss: 0.1459 - accuracy: 0.9781 - val_loss: 0.4030 - val_accuracy: 0.9602

✓ 1m 26s completed at 8:01 AM

89°F Clear

Search

8:23 AM 9/22/2023