

# NEURAL NETWORKS ASSIGNMENT 2

NAME: SUPRIYA SAMA  
700744510

Video Link:

<https://drive.google.com/file/d/14-b-SH2YtRGorSQQNwjAvfzbAuFC5sK/view?usp=sharing>

## 1. Use the use case in the class:

Add more Dense layers to the existing code and check how the accuracy changes.

Here I have added one more dense layer for the code and we got the accuracy of 0.6615, whereas with one layer we got the accuracy of 0.6671.

Accuracy is more than there is only one Dense layer.

The screenshot shows a Jupyter Notebook interface with several code cells and their outputs.

**In [23]:**

```
import pandas as pd
```

**In [38]:**

```
path_to_csv = "C:\\\\Users\\\\supri\\\\OneDrive\\\\Desktop\\\\diabetes.csv"
```

**In [39]:**

```
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(4, 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))

18/18 [=====] - 0s 2ms/step - loss: 0.6379 - acc: 0.6615
Model: "sequential_22"
```

**In [40]:**

```
18/18 [=====] - 0s 2ms/step - loss: 0.6379 - acc: 0.6615
Model: "sequential_22"

Layer (type)          Output Shape         Param #
dense_48 (Dense)     (None, 20)           180
dense_49 (Dense)     (None, 4)            84
dense_50 (Dense)     (None, 1)            5
=====
Total params: 269
Trainable params: 269
Non-trainable params: 0
```

**Output:**

```
None
6/6 [=====] - 0s 1ms/step - loss: 0.6861 - acc: 0.6198
[0.6860761046409607, 0.6197916865348816]
```

The notebook shows two code cells. The first cell imports pandas and defines the path to a CSV file. The second cell imports keras, defines the dataset, and creates a sequential model with three dense layers (20, 4, and 1 units respectively) and a sigmoid activation for the output. It then compiles the model with binary crossentropy loss and adam optimizer, fits it to the training data, and prints the summary and evaluation results. The output shows an accuracy of 0.6615. The third cell shows the detailed model summary, including layer types, output shapes, and parameters, followed by the evaluation results for the test set, which show an accuracy of 0.6198.

## 2. Change the data source to Breast Cancer dataset \* available in the source code folder and make required changes. Report accuracy of the model.

Here first I loaded the dataset then I created the model.

The screenshot shows a Jupyter Notebook interface running on a Windows desktop. The browser tab is titled "localhost:8888/notebooks/NEURALNETWORKASSIGNMENT2.1.ipynb". The notebook contains the following code:

```
In [1]: path_to_csv = "C:\\\\Users\\\\supri\\\\OneDrive\\\\Desktop\\\\breastcancer.csv"

In [33]: import keras
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

# Load dataset
cancer_data = load_breast_cancer()
X_train, X_test, Y_train, Y_test = train_test_split(cancer_data.data, cancer_data.target,
                                                    test_size=0.25, random_state=87)
np.random.seed(155)
my_nn = Sequential() # create model
my_nn.add(Dense(20, input_dim=30, activation='relu')) # hidden Layer 1
my_nn.add(Dense(1, activation='sigmoid')) # output Layer
my_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
my_nn_fitted = my_nn.fit(X_train, Y_train, epochs=100,
                         initial_epoch=0)
print(my_nn.summary())
print(my_nn.evaluate(X_test, Y_test))
Epoch 100/100
14/14 [=====] - 0s 2ms/step - loss: 0.3443 - acc: 0.9202
```

The screenshot shows the continuation of the Jupyter Notebook from the previous one. The notebook cell In [33] has been run, and its output is displayed:

```
Model: "sequential_17"
-----  
Layer (type)          Output Shape         Param #
-----  
dense_36 (Dense)      (None, 20)           620  
dense_37 (Dense)      (None, 1)            21  
-----  
Total params: 641  
Trainable params: 641  
Non-trainable params: 0  
-----  
None  
5/5 [=====] - 0s 4ms/step - loss: 0.2742 - acc: 0.9161
[0.2742051780223846, 0.9160839319229126]
```

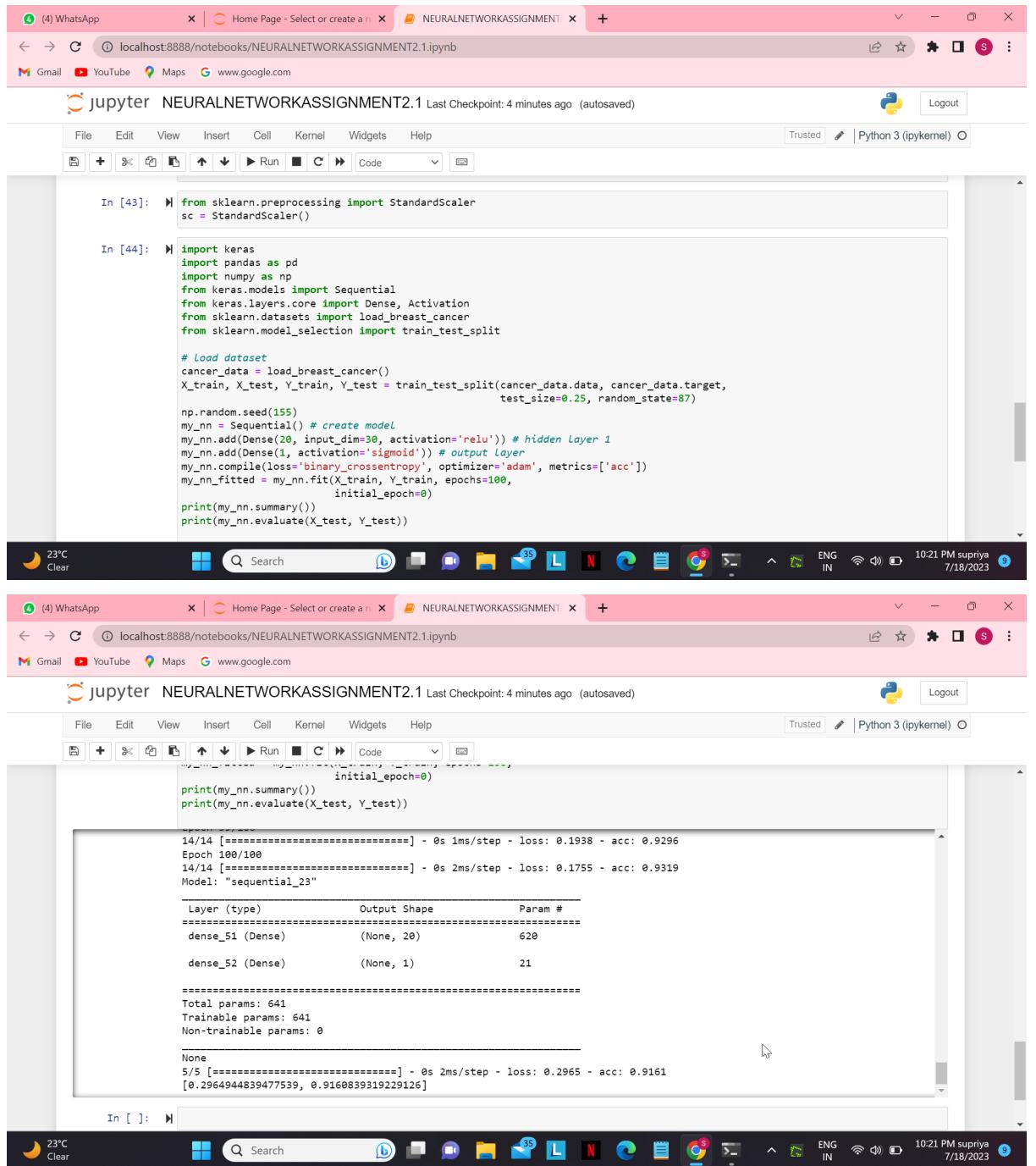
Below this, two new cells are shown:

```
In [42]: path_to_csv = "C:\\\\Users\\\\supri\\\\OneDrive\\\\Desktop\\\\breastcancer.csv"

In [43]: from sklearn.preprocessing import StandardScaler
```

We can see that the accuracy is 0.9161 and the loss is 0.2742.

### 3. Normalize the data before feeding the data to the model and check how the normalization change your accuracy.



```
In [43]: from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

In [44]: import keras
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

# Load dataset
cancer_data = load_breast_cancer()
X_train, X_test, Y_train, Y_test = train_test_split(cancer_data.data, cancer_data.target,
                                                    test_size=0.25, random_state=87)
np.random.seed(155)
my_nn = Sequential() # create model
my_nn.add(Dense(20, input_dim=30, activation='relu')) # hidden layer 1
my_nn.add(Dense(1, activation='sigmoid')) # output layer
my_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
my_nn_fitted = my_nn.fit(X_train, Y_train, epochs=100,
                         initial_epoch=0)
print(my_nn.summary())
print(my_nn.evaluate(X_test, Y_test))

In [45]: print(my_nn.summary())
print(my_nn.evaluate(X_test, Y_test))

14/14 [=====] - 0s 1ms/step - loss: 0.1938 - acc: 0.9296
Epoch 100/100
14/14 [=====] - 0s 2ms/step - loss: 0.1755 - acc: 0.9319
Model: "sequential_23"
Layer (type)          Output Shape         Param #
=====
dense_51 (Dense)      (None, 20)           620
dense_52 (Dense)      (None, 1)            21
=====
Total params: 641
Trainable params: 641
Non-trainable params: 0
None
5/5 [=====] - 0s 2ms/step - loss: 0.2965 - acc: 0.9161
[0.2964944839477539, 0.9160839319229126]
```

We got a loss of 0.2965 and accuracy of 0.9161.

### 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.

In [2]:

```
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:])
import matplotlib.pyplot as plt
#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()

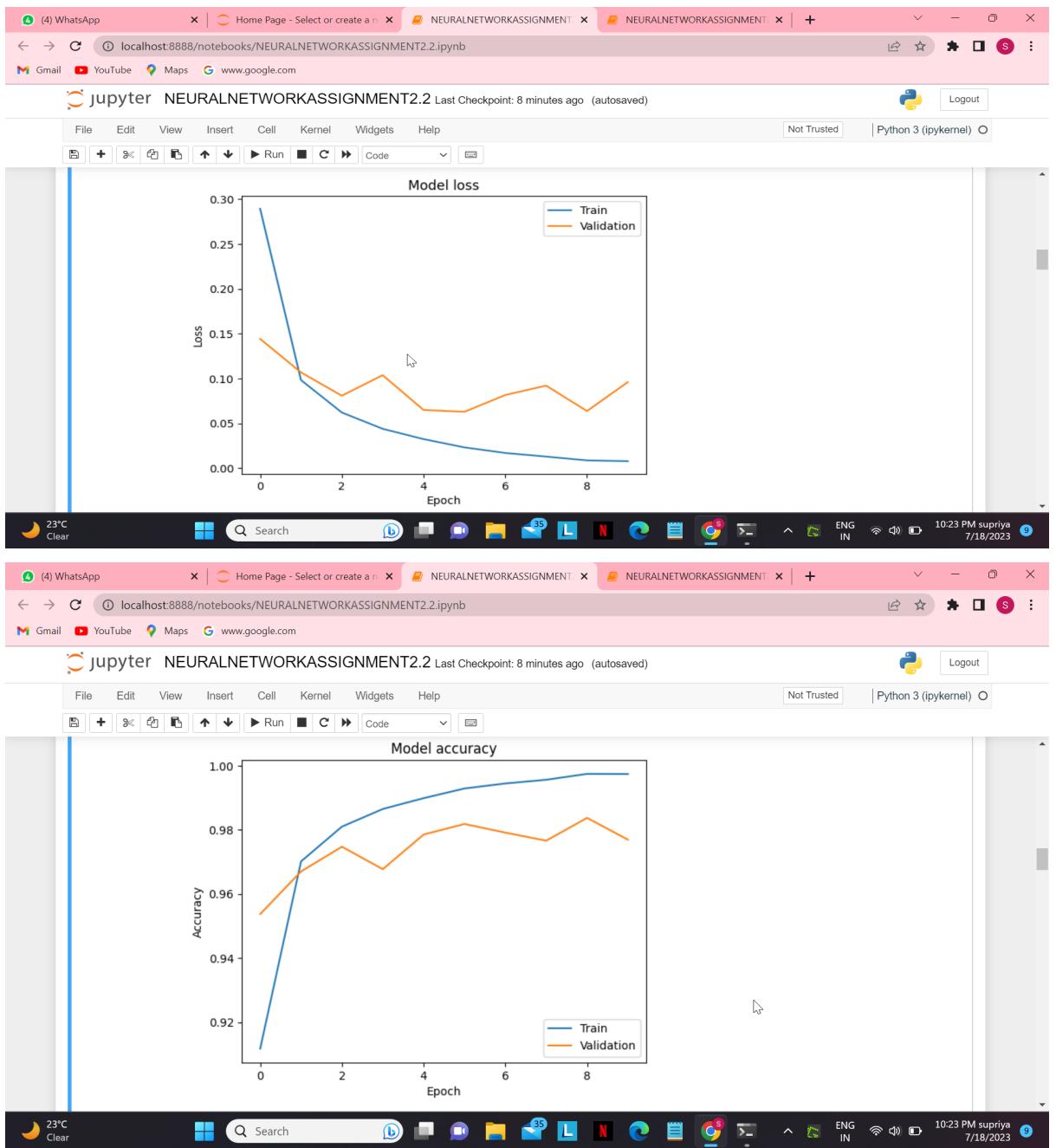
```

```
#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))

# 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', 'Validation'], loc='upper right')
plt.show()

# 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', 'Validation'], loc='lower right')
plt.show()
```



Here we can see the plot of the loss and accuracy for both training data and validation data.

## 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.

Here first we have to choose the image from the test data, then plot the image.

Screenshot of a Jupyter Notebook interface showing code execution and output.

The code in cell [3] is:

```
# choose an image from the test data
img_index = 0

# plot the image
plt.imshow(test_images[img_index], cmap='gray')
plt.show()

# make a prediction
img = test_data[img_index]
prediction = model.predict(np.array([img]))
print("Prediction: ", np.argmax(prediction))
```

The output shows a 28x28 pixel grayscale image of a handwritten digit labeled '7'.

Screenshot of a Jupyter Notebook interface showing code execution and output.

The code in cell [3] is:

```
print("Prediction: ", np.argmax(prediction))
```

The output shows a 28x28 pixel grayscale image of a handwritten digit labeled '7'.

We got the prediction 7.

**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.**

```
In [6]: 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

(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)

#createing network
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(dimData,)))


```

```
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))

#create a list of models to train
models = []

# model with 1 hidden Layer and tanh activation
model = Sequential()
model.add(Dense(512, activation='tanh', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
models.append(['1 hidden layer with tanh', model])

# model with 1 hidden Layer and sigmoid activation
model = Sequential()
model.add(Dense(512, activation='sigmoid', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
models.append(['1 hidden layer with sigmoid', model])


```

Here we used **tanh** and **sigmoid** activations.

The screenshot shows a Jupyter Notebook interface with the title "jupyter NEURALNETWORKASSIGNMENT2.2". The code cell contains several parts of Python code for building neural networks:

```
model.add(Dense(num_classes, activation='softmax'))
models.append('1 hidden layer with sigmoid', model)

# model with 2 hidden Layers and tanh activation
model = Sequential()
model.add(Dense(512, activation='tanh', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='tanh'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
models.append('2 hidden layers with tanh', model)

# model with 2 hidden Layers and sigmoid activation
model = Sequential()
model.add(Dense(512, activation='sigmoid', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='sigmoid'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
models.append('2 hidden layers with sigmoid', model)

# train each model and plot loss and accuracy curves
for name, model in models:
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

The status bar at the bottom indicates "23°C Clear" and the date/time "10:25 PM supriya 7/18/2023".

The screenshot shows a continuation of the Jupyter Notebook code from the previous screenshot:

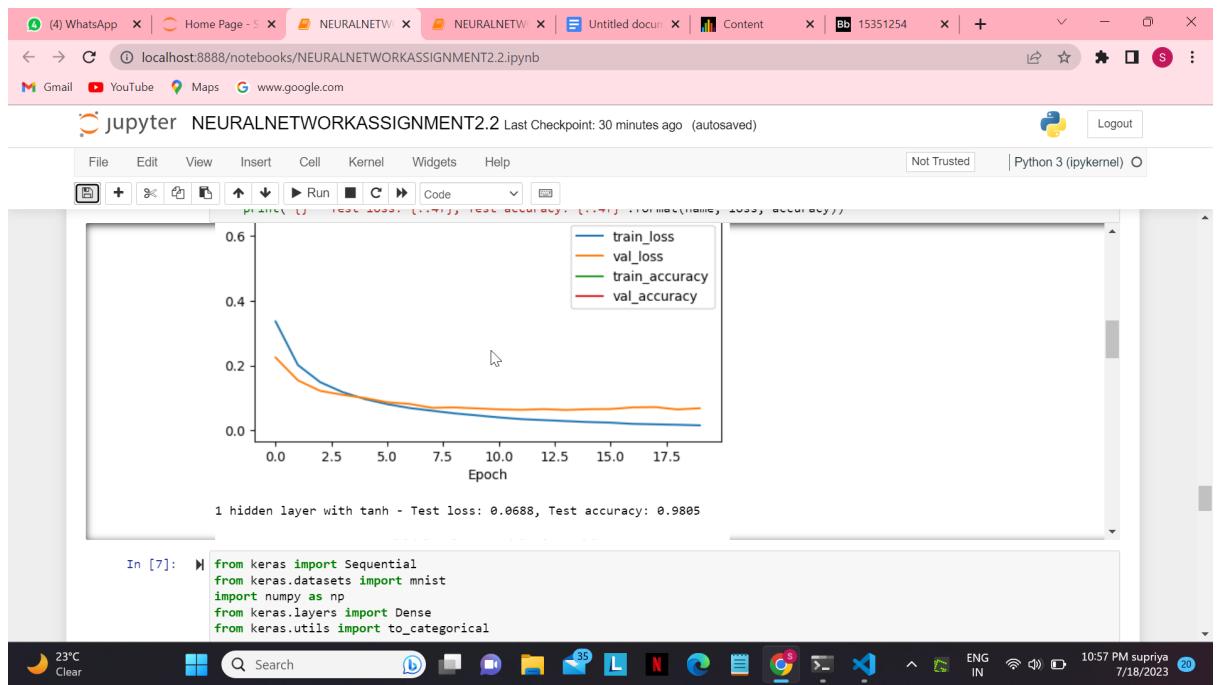
```
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
models.append('2 hidden layers with sigmoid', model)

# train each model and plot loss and accuracy curves
for name, model in models:
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    history = model.fit(x_train.reshape(-1, 784), y_train, validation_data=(x_test.reshape(-1, 784),
        epochs=20, batch_size=128, verbose=0)

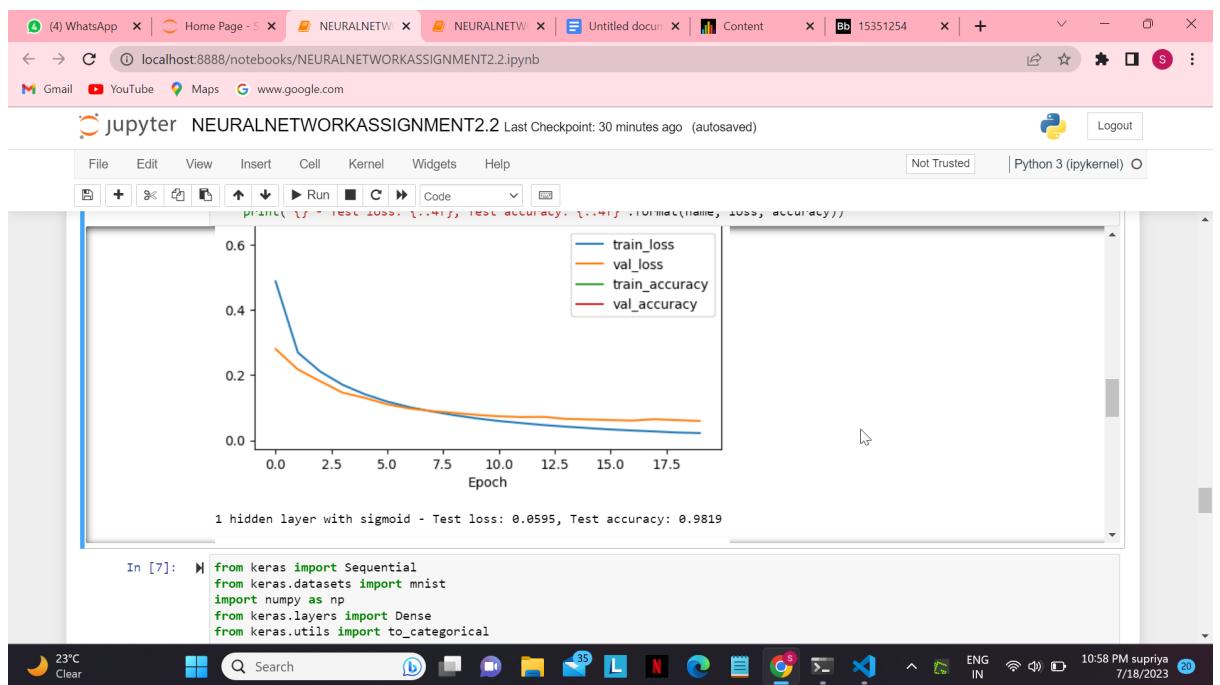
# plot loss and accuracy curves
plt.plot(history.history['loss'], label='train_loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.plot(history.history['accuracy'], label='train_accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.title(name)
plt.xlabel('Epoch')
plt.legend()
plt.show()

# evaluate the model on test data
loss, accuracy = model.evaluate(x_test.reshape(-1, 784), y_test, verbose=0)
print('{} - Test loss: {:.4f}, Test accuracy: {:.4f}'.format(name, loss, accuracy))
```

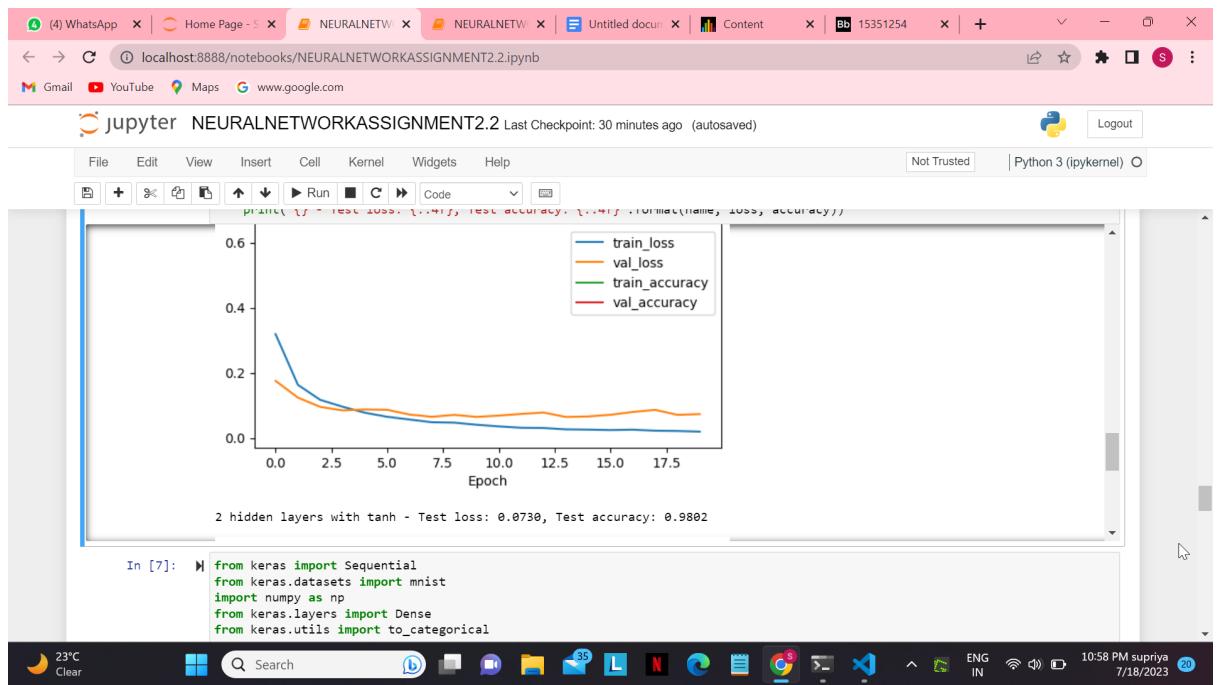
The status bar at the bottom indicates "23°C Clear" and the date/time "10:26 PM supriya 7/18/2023".



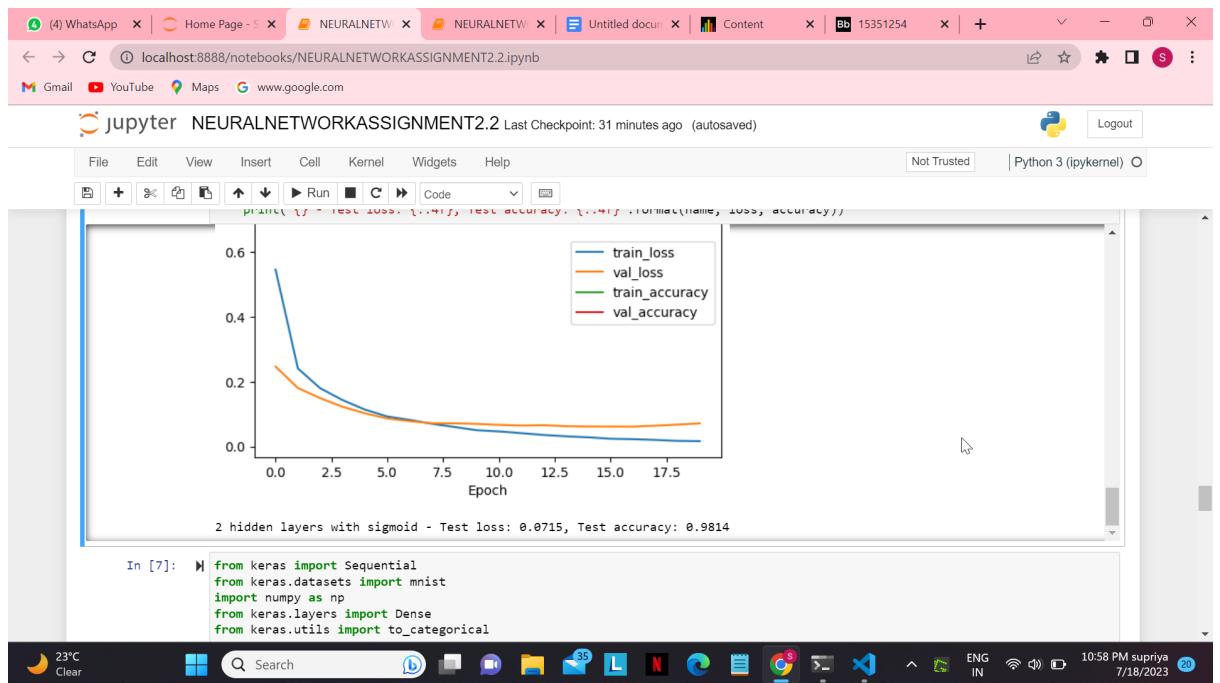
We can see the loss and accuracy for 1 hidden layer with tanh.



We can see the loss and accuracy for 1 hidden layer with sigmoid.



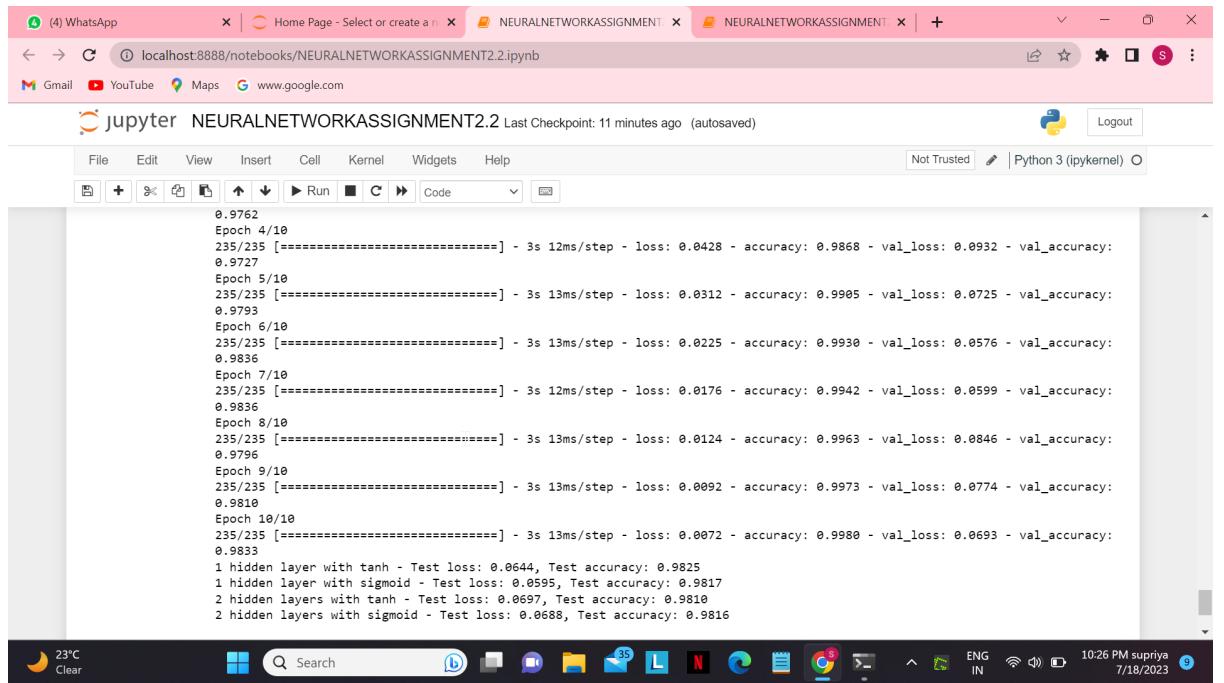
We can see the loss and accuracy for 2 hidden layer with tanh.



We can see the loss and accuracy for 2 hidden layer with sigmoid.

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

I run the same code without scaling the images. Here we can see the output, which is different from the earlier one.



```
0.9762
Epoch 4/10
235/235 [=====] - 3s 12ms/step - loss: 0.0428 - accuracy: 0.9868 - val_loss: 0.0932 - val_accuracy: 0.9727
Epoch 5/10
235/235 [=====] - 3s 13ms/step - loss: 0.0312 - accuracy: 0.9905 - val_loss: 0.0725 - val_accuracy: 0.9793
Epoch 6/10
235/235 [=====] - 3s 13ms/step - loss: 0.0225 - accuracy: 0.9938 - val_loss: 0.0576 - val_accuracy: 0.9836
Epoch 7/10
235/235 [=====] - 3s 12ms/step - loss: 0.0176 - accuracy: 0.9942 - val_loss: 0.0599 - val_accuracy: 0.9836
Epoch 8/10
235/235 [=====] - 3s 13ms/step - loss: 0.0124 - accuracy: 0.9963 - val_loss: 0.0846 - val_accuracy: 0.9796
Epoch 9/10
235/235 [=====] - 3s 13ms/step - loss: 0.0092 - accuracy: 0.9973 - val_loss: 0.0774 - val_accuracy: 0.9810
Epoch 10/10
235/235 [=====] - 3s 13ms/step - loss: 0.0072 - accuracy: 0.9980 - val_loss: 0.0693 - val_accuracy: 0.9833
1 hidden layer with tanh - Test loss: 0.0644, Test accuracy: 0.9825
1 hidden layer with sigmoid - Test loss: 0.0595, Test accuracy: 0.9817
2 hidden layers with tanh - Test loss: 0.0697, Test accuracy: 0.9810
2 hidden layers with sigmoid - Test loss: 0.0688, Test accuracy: 0.9816
```

We can see both the loss and accuracy has changed.