

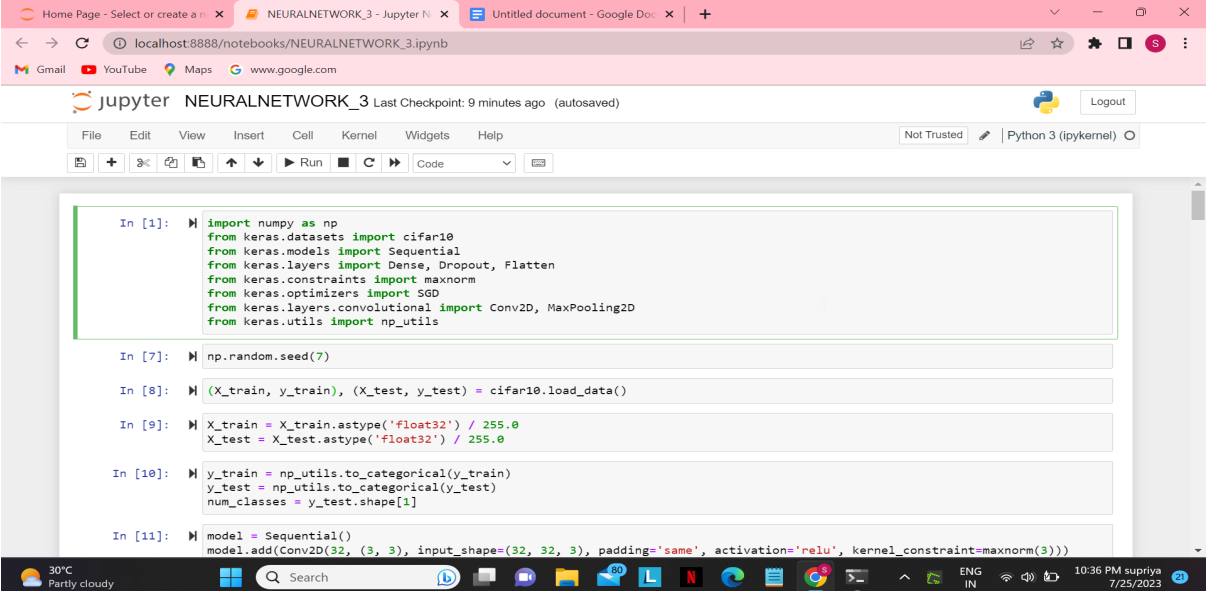
# NEURAL NETWORK ASSIGNMENT 3

NAME : SUPRIYA SAMA  
700744510

Video Link:

[https://drive.google.com/file/d/1P7UEvjwiLI0alcGcg5xHZGBGX4b\\_DJrj/view?usp=sharing](https://drive.google.com/file/d/1P7UEvjwiLI0alcGcg5xHZGBGX4b_DJrj/view?usp=sharing)

1. First I have imported numpy, cifar10 and other libraries using “import”. Then applied all the instructions given.



The screenshot shows a Jupyter Notebook titled 'NEURALNETWORK\_3' running on a local host. The code is as follows:

```
In [1]: import numpy as np
        from keras.datasets import cifar10
        from keras.models import Sequential
        from keras.layers import Dense, Dropout, Flatten
        from keras.constraints import maxnorm
        from keras.optimizers import SGD
        from keras.layers.convolutional import Conv2D, MaxPooling2D
        from keras.utils import np_utils

In [7]: np.random.seed(7)

In [8]: (X_train, y_train), (X_test, y_test) = cifar10.load_data()

In [9]: X_train = X_train.astype('float32') / 255.0
        X_test = X_test.astype('float32') / 255.0

In [10]: y_train = np_utils.to_categorical(y_train)
         y_test = np_utils.to_categorical(y_test)
         num_classes = y_test.shape[1]

In [11]: model = Sequential()
         model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3), padding='same', activation='relu', kernel_constraint=maxnorm(3)))
```

The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), a toolbar with icons for file operations and execution, and a status bar at the bottom showing system information like temperature (30°C) and time (10:36 PM).

The screenshot shows a Jupyter Notebook titled 'NEURALNETWORK\_3'. The first cell (In [11]) defines a sequential model with the following layers: Conv2D(32, (3, 3), input\_shape=(32, 32, 3), padding='same', activation='relu', kernel\_constraint=maxnorm(3)), Dropout(0.2), Conv2D(32, (3, 3), activation='relu', padding='same', kernel\_constraint=maxnorm(3)), MaxPooling2D(pool\_size=(2, 2), padding='same'), Flatten(), Dense(512, activation='relu', kernel\_constraint=maxnorm(3)), Dropout(0.5), and Dense(num\_classes, activation='softmax'). The second cell (In [12]) compiles the model with categorical\_crossentropy loss, SGD optimizer, and accuracy metric. The output shows the model summary:

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
dropout (Dropout)	(None, 32, 32, 32)	0
conv2d_1 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0

After applying all the instructions we got different outputs.

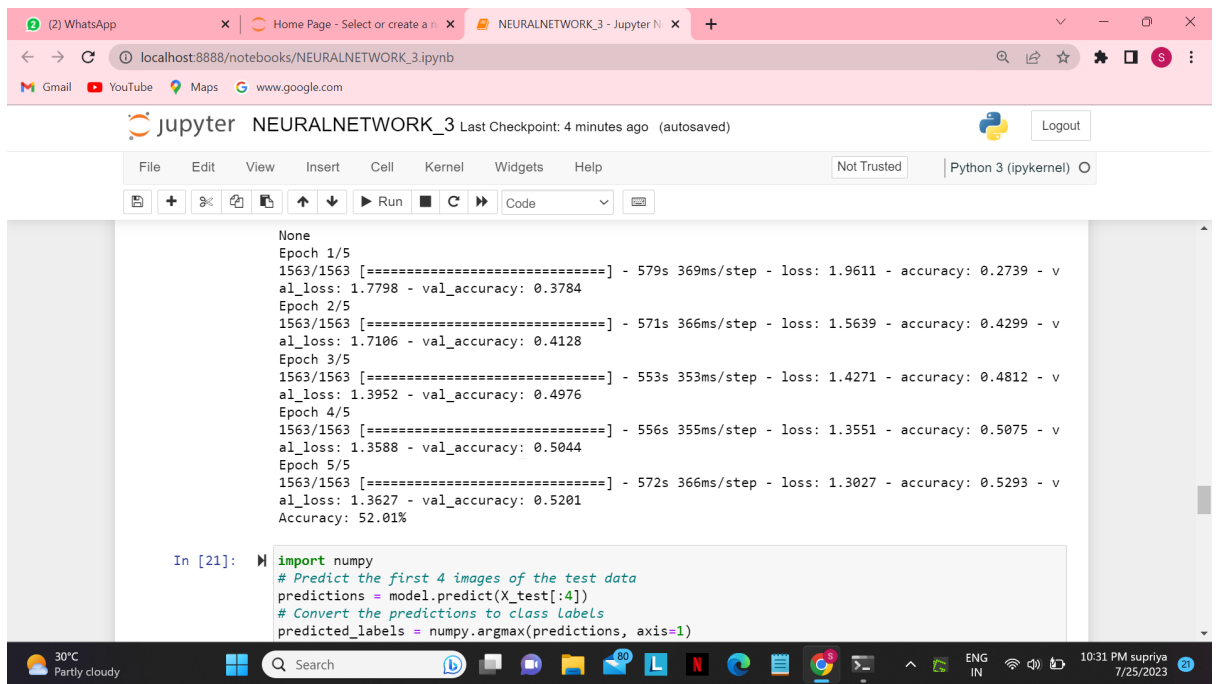
The screenshot shows the same Jupyter Notebook after training. The output of the training process (Out[13]) displays the progress over 5 epochs:

```
Epoch 1/5
1563/1563 [=====] - 291s 185ms/step - loss: 1.7104 - accuracy: 0.3804 - val_loss: 1.5010 - val_accuracy: 0.4521
Epoch 2/5
1563/1563 [=====] - 300s 192ms/step - loss: 1.3668 - accuracy: 0.5107 - val_loss: 1.2320 - val_accuracy: 0.5556
Epoch 3/5
1563/1563 [=====] - 306s 195ms/step - loss: 1.1919 - accuracy: 0.5785 - val_loss: 1.1294 - val_accuracy: 0.5968
Epoch 4/5
1563/1563 [=====] - 297s 190ms/step - loss: 1.0590 - accuracy: 0.6257 - val_loss: 1.0573 - val_accuracy: 0.6176
Epoch 5/5
1563/1563 [=====] - 293s 188ms/step - loss: 0.9420 - accuracy: 0.6688 - val_loss: 0.9865 - val_accuracy: 0.6567
```

The final cell (In [14]) evaluates the model on test data, resulting in an accuracy of 65.67%.

```
Out[13]: <keras.callbacks.History at 0x7a71141d9d80>
In [14]: scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
Accuracy: 65.67%
```

Here, we can see the accuracy is 65.67.

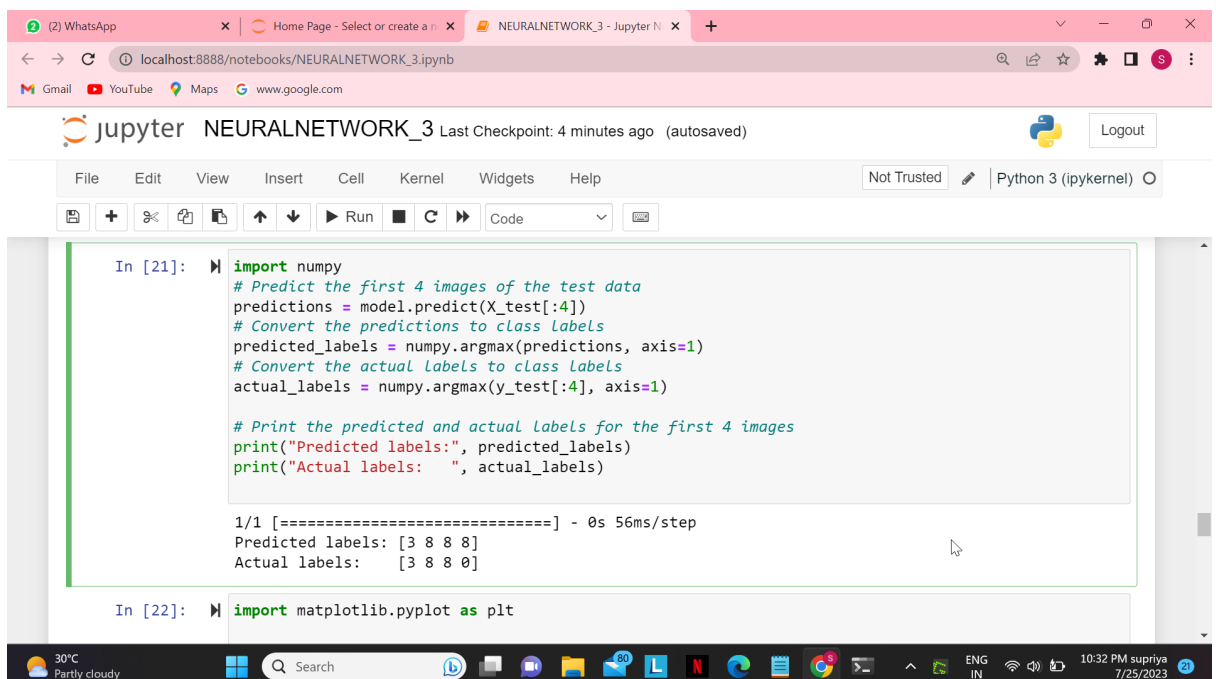


```
None
Epoch 1/5
1563/1563 [=====] - 579s 369ms/step - loss: 1.9611 - accuracy: 0.2739 - v
al_loss: 1.7798 - val_accuracy: 0.3784
Epoch 2/5
1563/1563 [=====] - 571s 366ms/step - loss: 1.5639 - accuracy: 0.4299 - v
al_loss: 1.7106 - val_accuracy: 0.4128
Epoch 3/5
1563/1563 [=====] - 553s 353ms/step - loss: 1.4271 - accuracy: 0.4812 - v
al_loss: 1.3952 - val_accuracy: 0.4976
Epoch 4/5
1563/1563 [=====] - 556s 355ms/step - loss: 1.3551 - accuracy: 0.5075 - v
al_loss: 1.3588 - val_accuracy: 0.5044
Epoch 5/5
1563/1563 [=====] - 572s 366ms/step - loss: 1.3027 - accuracy: 0.5293 - v
al_loss: 1.3627 - val_accuracy: 0.5201
Accuracy: 52.01%

In [21]: import numpy
# Predict the first 4 images of the test data
predictions = model.predict(X_test[:4])
# Convert the predictions to class labels
predicted_labels = numpy.argmax(predictions, axis=1)
```

When it comes to the next output, the accuracy has changed to 52.01. We got different loss and accuracy for each epoch.

2. First let's predict the 4 images of test data, then let's convert the predictions to class labels. After that let's convert the actual labels to class labels as shown in the figure.



```
In [21]: import numpy
# Predict the first 4 images of the test data
predictions = model.predict(X_test[:4])
# Convert the predictions to class labels
predicted_labels = numpy.argmax(predictions, axis=1)
# Convert the actual labels to class labels
actual_labels = numpy.argmax(y_test[:4], axis=1)

# Print the predicted and actual labels for the first 4 images
print("Predicted labels:", predicted_labels)
print("Actual labels: ", actual_labels)

1/1 [=====] - 0s 56ms/step
Predicted labels: [3 8 8 8]
Actual labels:   [3 8 8 0]

In [22]: import matplotlib.pyplot as plt
```

Here we can see the output, where the prediction labels are different from the actual labels.

### 3. Now let's plot the loss and accuracy.

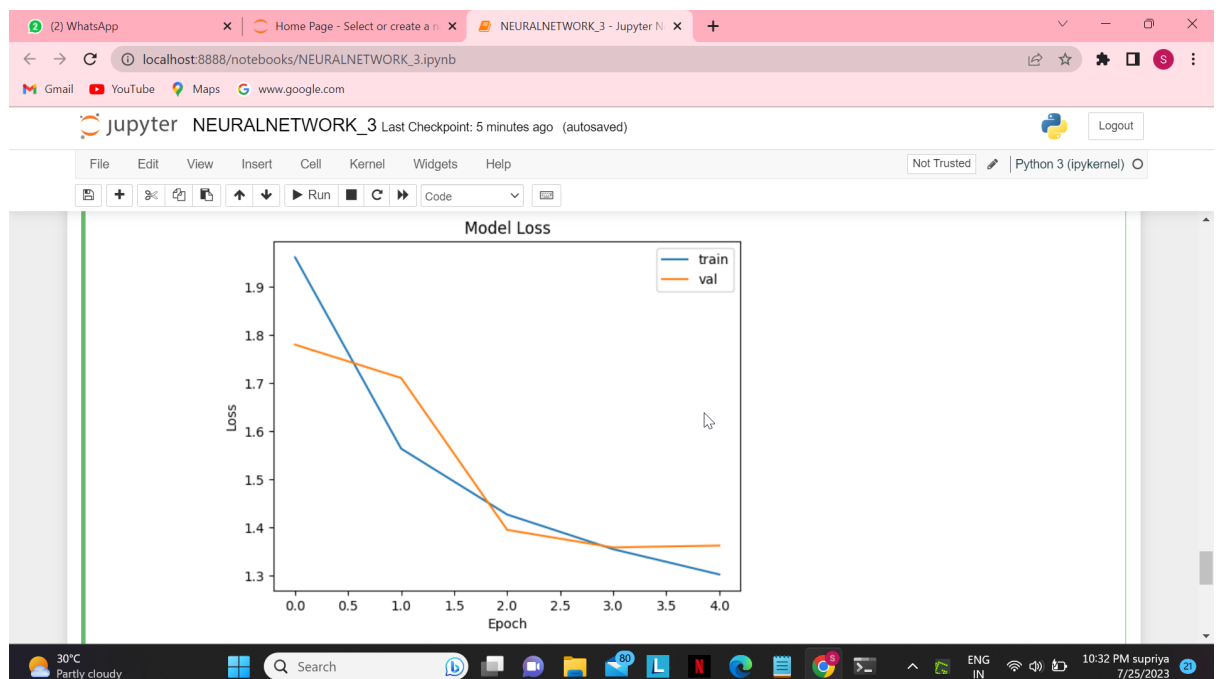
Here I have imported matplotlib.pyplot as plt.

```
In [22]: import matplotlib.pyplot as plt

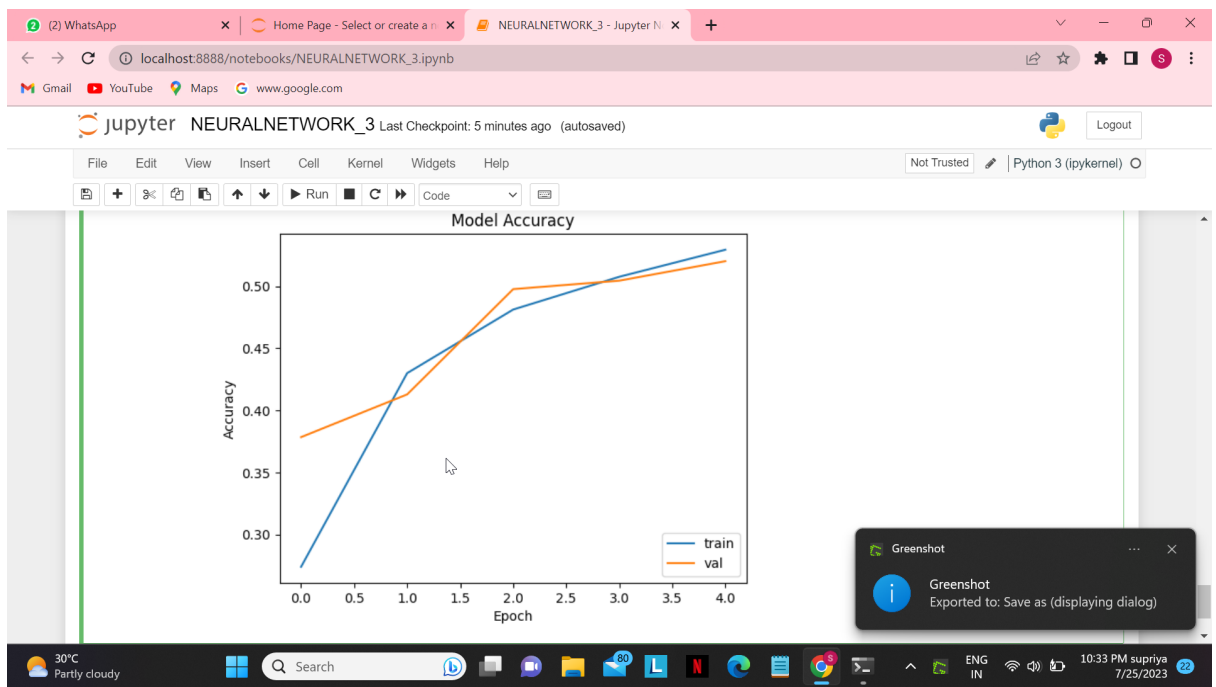
# Plot the training and validation loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()

# Plot the training and validation accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train', 'val'], loc='lower right')
plt.show()
```

Model Loss



This is the plot of training and validation loss.



This is the plot of training and validation accuracy.