


Affective Computing
Prof. Gulshan Sharma
Department of Computer Science and Engineering
Indraprastha Institute of Information Technology, Delhi

Lecture - 14
Tutorial: Emotion Recognition using Images

Hi, everyone. My name is Gulshan Sharma. I am the TA for this NPTEL Affective Computing Course. In this Tutorial, we will be working on Emotion Recognition using Images.

(Refer Slide Time: 00:37)



The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS)

- Collection of 7356 files
- Speech and Songs by 24 actors (12 male and 12 female)
- Includes different emotions classes such as
 - calm, happy, sad, angry, fearful, surprise, and disgust.
- Two levels of emotional intensity
 - Normal
 - Strong
- Available in three modality formats:
 - Audio-only (16bit, 48kHz .wav)
 - Audio-Video (720p H.264, AAC 48kHz, .mp4)
 - **Video-only (no sound)**

We will be using RAVDESS dataset which consists of 7,356 files divided into 8 emotion classes which are calm, happiness, sadness, anger, fearful, surprise and disgust. For this tutorial, we will be using video only data.

(Refer Slide Time: 01:01)




Filename Identifiers: 03-01-01-01-01-01-01.wav

- Modality (01 = full-AV, 02 = video-only, 03 = audio-only).
- Vocal channel (01 = speech, 02 = song).
- Emotion (01 = neutral, 02 = calm, 03 = happy, 04 = sad, 05 = angry, 06 = fearful, 07 = disgust, 08 = surprised).
- Emotional intensity (01 = normal, 02 = strong). NOTE: There is no strong intensity for the 'neutral' emotion.
- Statement (01 = "Kids are talking by the door", 02 = "Dogs are sitting by the door").
- Repetition (01 = 1st repetition, 02 = 2nd repetition).
- Actor (01 to 24. Odd numbered actors are male, even numbered actors are female).

So, this is the file name identifier. It consists of seven sub identifiers in which 3rd sub identifiers tell us about the emotion class.

(Refer Slide Time: 01:14)



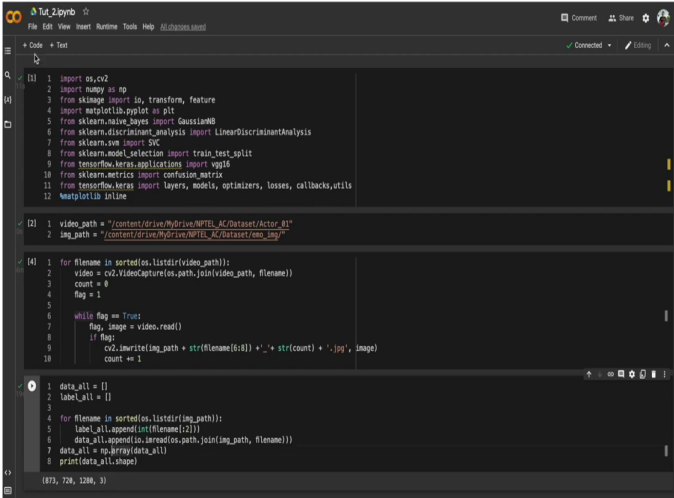
Tutorial Overview

- Following Experiments on Google Colab
 - Extracting Frames from Video File in Python
 - HoG based Emotion Recognition
 - GNB
 - LDA
 - SVM
 - Classifying emotions using VGG-16 (Pretrained on VGG Face dataset)

So, let me give you the overview of this tutorial. We will start with extracting frames on the video file in Python, then we will be extracting a HoG which is Histogram of oriented Gradient feature from this images. And, we will try to classify these features using our machine learning classifier like Gaussian Naïve Bayes, Linear Discriminant Analysis and Support Vector Vision.

After that, we will be using VGG-16 architecture which will be pre-trained on VGG phase dataset and we will try to classify these emotional images. Let me tell you this VGG about this VGG face dataset. So, VGG face is a dataset of 2.6 million face images of 2622 people that is used to train this VGG16 architecture. So, let us start with the coding part now.

(Refer Slide Time: 02:12)



```
1 import cv2
2 import numpy as np
3 from skimage import io, transform, feature
4 import matplotlib.pyplot as plt
5 from sklearn.naive_bayes import GaussianNB
6 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
7 from sklearn.svm import SVC
8 from sklearn.model_selection import train_test_split
9 from tensorflow.keras.applications import vgg16
10 from sklearn.metrics import confusion_matrix
11 from tensorflow.keras import layers, models, optimizers, losses, callbacks, utils
12 import os, sys, time

13 video_path = "/content/drive/MyDrive/NPTEL_AC/Dataset/Actor_01"
14 img_path = "/content/drive/MyDrive/NPTEL_AC/Dataset/emo_img"

15 for filename in sorted(os.listdir(video_path)):
16     video = cv2.VideoCapture(os.path.join(video_path, filename))
17     count = 0
18     flag = 1
19     while flag == True:
20         flag, image = video.read()
21         if flag:
22             cv2.imwrite(img_path + str(filename[6:8]) + "_" + str(count) + ".jpg", image)
23             count += 1

24 data_all = []
25 label_all = []
26 for filename in sorted(os.listdir(img_path)):
27     label_all.append(int(filename[2]))
28     data_all.append(io.imread(os.path.join(img_path, filename)))
29 print(data_all.shape)
```

So, in our coding part, we will start with importing all the essential libraries that will be required during the programming. After importing this library, I will start with data fetching part and for this part, I am assuming that everyone has already downloaded the data and saved in their respective Google drives. So, here I will create couple of variables which will be essentially signifying the dataset path.

After creating these variables, I will be writing a code to extract frame from the video and saving the frames into another directory. So, the code will look something like this. In this code, we will simply iterate through all the video files available in our folder and we will read each file using this cv2 library.

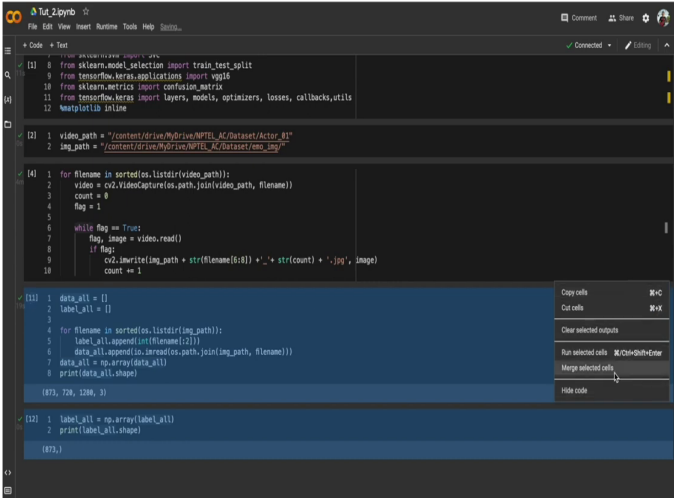
So, there is a function in our open cv library called video capture which will essentially capture all the videos with respect to the input path. And later, we will simply extract the

frame out of this video and save it to another directory. This code might take a couple of minutes to execute completely. So, please be patient. So, after completion of this code block, frames are extracted from the video files and saved into another folder at this location.

Now, our next task will be to extract all the images from this particular folder and save them into a separate numpy arrays. For that, we will writing another code. So, the code will look something like this where we have defined our two list data all and label all list and we are iterating through all of our files and we are simply reading those file names, extracting the label out of those file and saving them into the and appending them into the label all list.

Later on, we are simply reading all these image files using our sk image library and saving them into data all list. Let me run this code, ok. So, as we can see that we are getting these 873 files, 873 images whose dimension are 720 cross 1280 and three simply the number of channels which represent here RGB images. Let me also convert the label all into what we call it numpy array.

(Refer Slide Time: 05:23)



```
File Edit View Insert Runtime Tools Help Settings
+ Code + Text
from sklearn.model_selection import train_test_split
from tensorflow.keras.applications import vgg16
from sklearn.metrics import confusion_matrix
from tensorflow.keras import layers, models, optimizers, losses, callbacks, utils
matplotlib inline

[2]: 1 video_path = "/content/drive/MyDrive/NPTEL_IC/DataSet/Actor 01"
     2 img_path = "/content/drive/MyDrive/NPTEL_IC/DataSet/Img"

[4]: 1 for filename in sorted(os.listdir(video_path)):
     2     video = cv2.VideoCapture(os.path.join(video_path, filename))
     3     count = 0
     4     flag = 1
     5
     6     while flag == True:
     7         flag, image = video.read()
     8         if flag:
     9             cv2.imwrite(img_path + str(filename[6:8]) + '_' + str(count) + '.jpg', image)
    10         count += 1

[11]: 1 data_all = []
     2 label_all = []
     3
     4 for filename in sorted(os.listdir(img_path)):
     5     label_all.append(int(filename[2]))
     6     data_all.append(os.path.join(img_path, filename))
     7 data_all = np.array(data_all)
     8 print(data_all.shape)

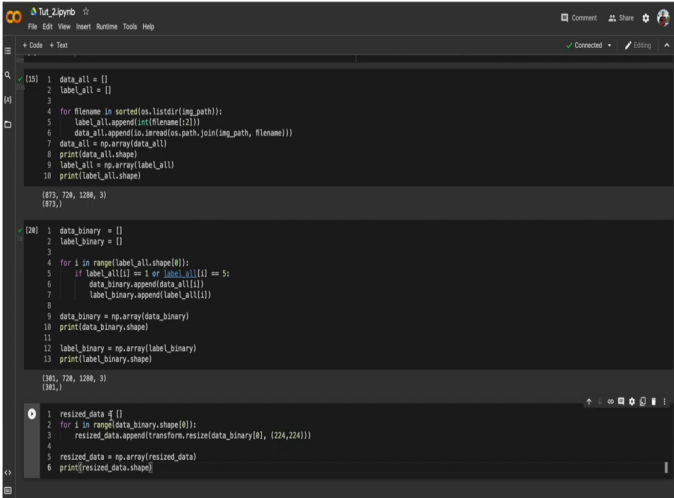
(873, 720, 320, 3)

[12]: 1 label_all = np.array(label_all)
     2 print(label_all.shape)

(873,)
```

So, for that, I will simply reuse my existing code, ok and we are getting 873 labels corresponding to these files. Let me also join these two cells for a simplicity.

(Refer Slide Time: 05:43)



```
1 data_all = []
2 label_all = []
3
4 for filename in sorted(os.listdir(img_path)):
5     label_all.append(int(filename[2]))
6     data_all.append(os.path.join(img_path, filename))
7 data_all = np.array(data_all)
8 print(data_all.shape)
9 label_all = np.array(label_all)
10 print(label_all.shape)
(173, 728, 1280, 3)
(173,)
```

```
1 data_binary = []
2 label_binary = []
3
4 for i in range(label_all.shape[0]):
5     if label_all[i] == 1 or label_all[i] == 5:
6         data_binary.append(data_all[i])
7         label_binary.append(label_all[i])
8
9 data_binary = np.array(data_binary)
10 print(data_binary.shape)
11
12 label_binary = np.array(label_binary)
13 print(label_binary.shape)
(181, 728, 1280, 3)
(181,)
```

```
1 resized_data = []
2 for i in range(data_binary.shape[0]):
3     resized_data.append(transform.resize(data_binary[i], (224,224)))
4
5 resized_data = np.array(resized_data)
6 print(resized_data.shape)
```

After that, as now we have data all the data and their corresponding label. As we have already seen there are 8 classes in this problem. So, just to save some computational time and make this problem a little bit easier, we will try to make it a sort of a binary classification where we will choose any two random classes. To do so, code will look something like this.

There will be clear to list data binary and label binary list and we will iterate through all the labels, all our original labels. And if our label is equal to 1 or the label is equal to 5, in both of these cases, we will simply append those label into our label binary list and all the data corresponding to these labels will be appended into data binary list. Later, we will simply transform these list into numpy array.

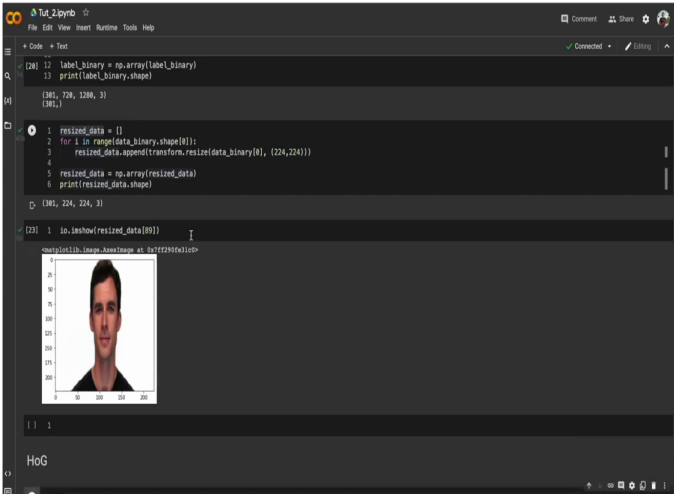
And, the shape of these array will look something like this. Now, as you can see that we are having lesser number of images, which will help us to get some computational efficiency over

here. And remember, I am talking about computational efficiency, not about the model efficiency or some sort of a training efficiency over here. This is just for the easier demonstration.

Now, we will try another sort of a computational optimization here, like apart from using 720 cross 1280 image, which is a huge dimensional image, which is a huge image, we will try to resize it into a smaller dimension, let us say 224 cross 224 image. So, for that, I will also write another piece of code and the code will look something like this.

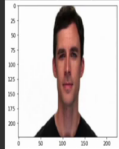
Here we are declaring a resized list called resized data, which will be storing over all the resized data. And we will be using this sk image in built function called resize transformed resize and we will pass the original data and the respective shape, which we have to resize.

(Refer Slide Time: 08:12)




```
File Edit View Insert Runtime Tools Help
+ Code + Text
(20) 12 label_binary = np.array(label_binary)
13 print(label_binary.shape)
[18]: (720, 1280, 3)
[19]:
1 resized_data = []
2 for i in range(data_binary.shape[0]):
3     resized_data.append(transform.resize(data_binary[i], (224, 224)))
4
5 resized_data = np.array(resized_data)
6 print(resized_data.shape)
[20]: (224, 224, 3)
(21) 1 cv.imshow(resized_data[0])
```

cv.imshow(resized_data[0])



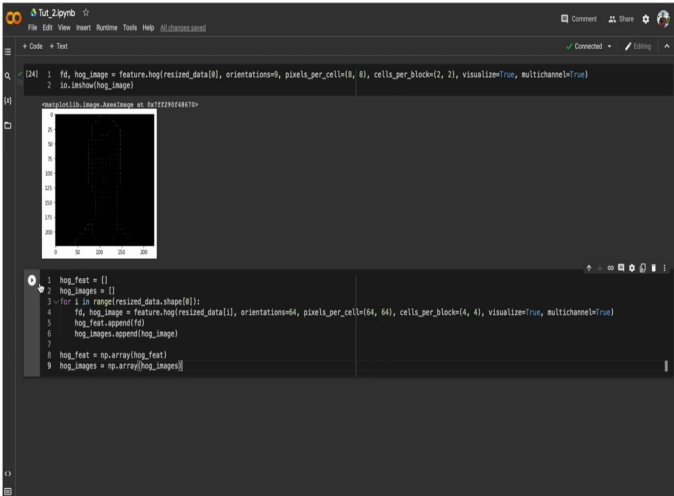
HoG



So, after running this code, no, as you can see after execution of this code, our new image size is 224 cross 224. So, to visualize this data, I will be using another function from psychic image library and this function is called io dot im show. And here I will simply pass some image that have resized and image will look something like this; maybe I can use another image say 89.

So, by now we have extracted all these images, and we have stored them into their respective folders. And we have made this as a binary classification problem and resized our images. Now, onwards, I will be, I will try to extract a histogram of gradients from these images and we will train couple of classifier over that and learn how to classify this image into the emotion classes. For that, so, to extract a HoG features, I will be again using an inbuilt function from sk image library.

(Refer Slide Time: 09:45)



The screenshot shows a Jupyter Notebook interface with the following code and output:

```
[24]: 1 fd, hog_image = feature.hog(resized_data[8], orientations=9, pixels_per_cell=(8, 8), cells_per_block=(2, 2), visualizer=True, multichannel=True)
      2 io.imshow(hog_image)
```

The output displays a grayscale visualization of the Histogram of Gradients (HoG) for a specific image. The visualization shows a dark background with a grid of small, light-colored rectangular blocks, representing the gradient orientations in the original image.

```
1 hog_feats = []
2 hog_images = []
3 for i in range(resized_data.shape[0]):
4     fd, hog_image = feature.hog(resized_data[i], orientations=64, pixels_per_cell=(8, 8), cells_per_block=(4, 4), visualizer=True, multichannel=True)
5     hog_feats.append(fd)
6     hog_images.append(hog_image)
7
8 hog_feats = np.array(hog_feats)
9 hog_images = np.array(hog_images)
```

And, this will look something like this, where I will be passing a resized image data and I will be passing some sort of a hyper parameter into this block. These hyper parameter orientation, which essentially represent the number of histogram bins. In original at HoG paper, they mention this parameter value to 9. And we will be using same value over here.

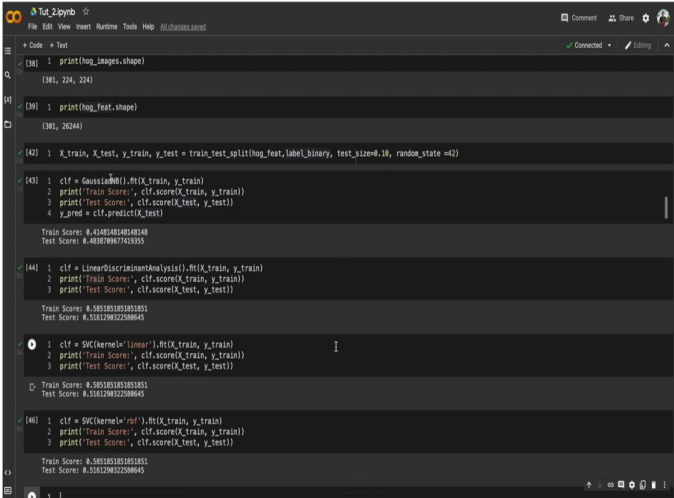
And, another hyper parameter in HoG function is pixels per cell, which essentially represents cell size for histogram and we will be using 8 x 8 pixel per cell. And our third hyper parameter is cells per block, which represent block size for the histogram normalization. And, after running this code, our image will look something like this.

Maybe I can zoom into a bit and as you can see, this HoG feature can represent the boundaries of you know this face and so, basically, this is the HoG image generated from this function and FD is the feature descriptors. Now, using FD we will try to classify the emotion classes. But before that, I need to calculate these HoG image and features for all the data set.

So, the HoG code will look something like this. Here in this code, I have declared two list name hog feature and hog images. So, what are we basically doing over here is we are iterating through all the resized images and we are using this in built hog extractor and saving these extracted images and features in these two list and later we are converting this list into NumPy array.

So, let me run this code to this block, maybe I can show you the shape of this hog feature and hog image.

(Refer Slide Time: 11:55)



```
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Test
[38] 1 print(hog_images.shape)
[38]: (387, 224, 224)
[39] 1 print(hog_feat.shape)
[39]: (387, 26244)
[42] 1 X_train, X_test, y_train, y_test = train_test_split(hog_feat, label_binary, test_size=0.10, random_state=42)
[43] 1 clf = GaussianNB().fit(X_train, y_train)
2 print('Train Score:', clf.score(X_train, y_train))
3 print('Test Score:', clf.score(X_test, y_test))
4 y_pred = clf.predict(X_test)
Train Score: 0.41821481548148
Test Score: 0.4035907743333
[44] 1 clf = LinearDiscriminantAnalysis().fit(X_train, y_train)
2 print('Train Score:', clf.score(X_train, y_train))
3 print('Test Score:', clf.score(X_test, y_test))
Train Score: 0.5053851851851851
Test Score: 0.5181298322588645
[45] 1 clf = SVC(kernel='linear').fit(X_train, y_train)
2 print('Train Score:', clf.score(X_train, y_train))
3 print('Test Score:', clf.score(X_test, y_test))
Train Score: 0.5053851851851851
Test Score: 0.5181298322588645
[46] 1 clf = SVC(kernel='rbf').fit(X_train, y_train)
2 print('Train Score:', clf.score(X_train, y_train))
3 print('Test Score:', clf.score(X_test, y_test))
Train Score: 0.5053851851851851
Test Score: 0.5181298322588645
```

So, for that, I will simply write shape of these images and shape of hog features, ok. So, these are respective shape of the features and these are the images over there. hog images over there. Now, we do have extracted this hog feature over here. So, we are ready to play with our machine learning classifier.

For that, I will start with dividing the data into their respective training and testing set. For that, I will be using this in-built function from sklearn and which is trained as split function and the code will look something like this. Here we are taking test size equal to 0.10, which essentially means 10 percent of the data will be taken as a test data. So, after execution of this code.

So, after dividing this data into their respective training test set, we will be using our first classifier, which will be Gaussian Naive Bayes. And the code will look something like this.

So, we have already imported this Gaussian Naive Bayes classifier and we are fitting it upon the our trained data. And later we are simply showing our training data score and test data score, ok.

So, as we can see that our training and test data, accuracies are for 41 percent and 48 percent respectively, which is not so good. So, we will be using our another classifier called linear discriminant analysis over here and the code will be similar. I will simply drag my code over here and apart from a Gaussian Naive Bayes, I will be using linear discriminant analysis over here and the rest the code is same.

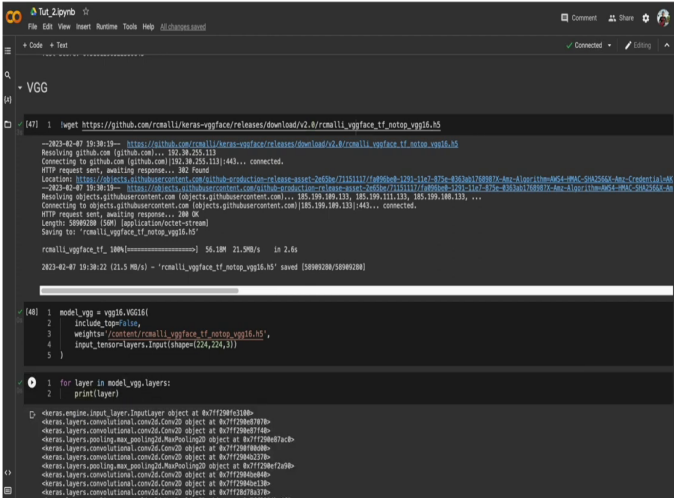
And in this case, we got a slightly higher train score and test accuracy over here. Finally, I will be using my linear score vector machine and whose code will look something like this. So, in this case, as I can see, the linear SVC is giving similar score as our linear discriminant analysis is giving. Let me try a same code with rbf kernel, ok.

Surprisingly, here we are also getting similar results using this hog feature. One more thing I would like to tell you is the since here I am using those standard hyperparameter in our hog feature extractor, one can try to experiment with changing the values of these hyperparameter and try to see is there any sort of effect in the classification accuracy by changing these particular values. That could be one exercise that you have to do on your own.

And, now I have done my VC classification part using HoG feature. Now, onward, I will be using a standard VGG 16 architecture, new network architecture to classify these emotions. And in this VGG 16 architecture, we will be using a pre-determined weights of VGG face dataset consists of 2.6 million face images.

So, the weight will be already pre-trained weights and using that model, pre-trained model and pre-trained weight, I will try to classify these emotions. So, for this, I first need to download the exact VGG 16 weight file.

(Refer Slide Time: 16:08)



```
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
+ VGG
(47) 1 wget https://github.com/rcmalli/keras-vggface/releases/download/v2.0/rcmalli_vggface_tf_notop_vgg16.h5
-----2023-02-07 19:38:19----- https://github.com/rcmalli/keras-vggface/releases/download/v2.0/rcmalli_vggface_tf_notop_vgg16.h5
Resolving github.com [github.com... 192.30.255.133]
Connecting to github.com [github.com|192.30.255.133]:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asset-206977111/189096b-2021-21e1-375e-823ba1708073?ac_token=9654-59x255x-4m-Crossent11xh5
-----2023-02-07 19:38:19----- https://objects.githubusercontent.com/github-production-release-asset-206977111/189096b-2021-21e1-375e-823ba1708073?ac_token=9654-59x255x-4m-Crossent11xh5
Resolving objects.githubusercontent.com [objects.githubusercontent.com... 185.199.108.133]
Connecting to objects.githubusercontent.com [objects.githubusercontent.com|185.199.108.133]:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5090208 (5.0M) [application/octet-stream]
Saving to: 'rcmalli_vggface_tf_notop_vgg16.h5'

rcmalli_vggface_tf_100%[=====] 56.18M  21.90M/s  in 2.6s

2023-02-07 19:38:22 (21.5 MB/s) - 'rcmalli_vggface_tf_notop_vgg16.h5' saved [5090208/5090208]

(48) 1 model_vgg = vgg16.VGG16(
2     include_top=False,
3     weights='/content/rcmalli_vggface_tf_notop_vgg16.h5',
4     input_tensor=layers.Input(shape=(224, 224, 3)),
5 )

1 for layer in model_vgg.layers:
2     print(layer)
<keras.engine.input_layer.InputLayer object at 0x7ff7206318b0>
<keras.layers.convolutional.Conv2D.Conv2D object at 0x7ff720631970>
<keras.layers.convolutional.Conv2D.Conv2D object at 0x7ff720631f40>
<keras.layers.pooling_max_pooling2d.MaxPool2D object at 0x7ff720631f60>
<keras.layers.convolutional.Conv2D.Conv2D object at 0x7ff720631f80>
<keras.layers.convolutional.Conv2D.Conv2D object at 0x7ff720632370>
<keras.layers.pooling_max_pooling2d.MaxPool2D object at 0x7ff720632390>
<keras.layers.convolutional.Conv2D.Conv2D object at 0x7ff7206323b0>
<keras.layers.convolutional.Conv2D.Conv2D object at 0x7ff7206323d0>
```

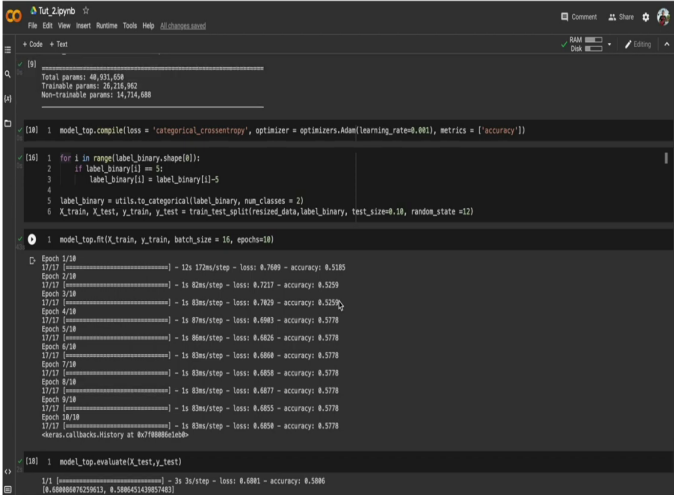
So, let me first separate this section over here. Say yeah, I will download my VGG 16-weight file and I will be using this particular link over here where these VGG face weights are released and let me download it. It might take some time according to your internet speed. Once it is downloaded, you can see this file over here.

So, it is in your local drive over here. Now, my weights are downloaded. I can simply use my VGG 16 model and I will code the look something like this where I will be using my VGG 16-pre-defined model and I will be including these weights. This is the basic this is basically the exact path of the weights and our input images like 224 cross 224 cross 3. So, after running this code, I can see my model is there. So, I can simply try to see the structure of this model.

using Adam Optimizer and the learning rate will be set at 0.001 and the validation metric will be accuracy.

So, ok, my model is compiled by now. Now, I just need to basically again define my training test set over here. So, after successful, so, after successful compilation of the model, our agenda will be to train this model, so, our agenda will be to fit this model. But, before fitting, I just need to, you know, I just need to divide my data into respective train and test set.

(Refer Slide Time: 22:03)



```
[0] Total params: 46,931,658
Trainable params: 26,716,962
Non-trainable params: 14,714,688

[0] 1 model_top.compile(loss = 'categorical_crossentropy', optimizer = optimizers.Adam(learning_rate=0.001), metrics = ['accuracy'])

[16] 1 for i in range(label_binary.shape[0]):
2     if label_binary[i] == 5:
3         label_binary[i] = label_binary[i]-5
4
5 label_binary = utils.to_categorical(label_binary, num_classes = 2)
6 X_train, X_test, y_train, y_test = train_test_split(resized_data, label_binary, test_size=0.18, random_state =12)

[0] 1 model_top.fit(X_train, y_train, batch_size = 16, epochs=18)

Epoch 1/18 ----- 12s 172ms/step - loss: 8.7689 - accuracy: 0.5385
Epoch 2/18 ----- 15 83ms/step - loss: 8.7217 - accuracy: 0.5259
Epoch 3/18 ----- 15 83ms/step - loss: 8.7829 - accuracy: 0.5219
Epoch 4/18 ----- 15 87ms/step - loss: 8.6983 - accuracy: 0.5778
Epoch 5/18 ----- 15 86ms/step - loss: 8.6826 - accuracy: 0.5778
Epoch 6/18 ----- 15 83ms/step - loss: 8.6868 - accuracy: 0.5778
Epoch 7/18 ----- 15 83ms/step - loss: 8.6858 - accuracy: 0.5778
Epoch 8/18 ----- 15 83ms/step - loss: 8.6877 - accuracy: 0.5778
Epoch 9/18 ----- 15 83ms/step - loss: 8.6855 - accuracy: 0.5778
Epoch 10/18 ----- 15 83ms/step - loss: 8.6858 - accuracy: 0.5778
Epoch 11/18 ----- 15 83ms/step - loss: 8.6858 - accuracy: 0.5778
Epoch 12/18 ----- 15 83ms/step - loss: 8.6858 - accuracy: 0.5778
Epoch 13/18 ----- 15 83ms/step - loss: 8.6858 - accuracy: 0.5778
Epoch 14/18 ----- 15 83ms/step - loss: 8.6858 - accuracy: 0.5778
Epoch 15/18 ----- 15 83ms/step - loss: 8.6858 - accuracy: 0.5778
Epoch 16/18 ----- 15 83ms/step - loss: 8.6858 - accuracy: 0.5778
Epoch 17/18 ----- 15 83ms/step - loss: 8.6858 - accuracy: 0.5778
Epoch 18/18 ----- 15 83ms/step - loss: 8.6858 - accuracy: 0.5778
-----> keras.callbacks.History at 8x788886e10db

[18] 1 model_top.evaluate(X_test, y_test)

[1] [0.0000009729613, 0.5886451439837463] 3s 3s/step - loss: 8.0081 - accuracy: 0.5886
```

And, also sort of pre-processing over here where I will be converting my label 5 to level 0. This is just a prefacing step over here and I will be converting these labels into categorical labels as we are using categorical cross entropy function. So, let me run this code, ok. So, we have successfully splitted our data into our respective train and test sets.

Now, our agenda will be to simply train this model, get this model. So, for this, I will be using model underscore top dot fit and I will be passing my train data and corresponding labels and for bed size, I am taking a bed size of 16 and number of epoch equal to 10.

And this training part will also take a couple of minutes. So, please have some patience. So, after training this model up to 10 epochs, we can see that our training accuracy is somewhat around 57 percent. Now, let us try to check what will be our validation accuracy as in test accuracy, which will be equivalent to ok 58 percent of the accuracy is over here.

So, what I can include from here is these networks, although our neural network is working comparatively better than our PC classifier, but still these networks are not trained up to its potential. We might need to do a rigorous hyper parameter tuning in terms of learning rate, loss function, optimizer, size of this particular MLP over here and maybe we can get a better result than this. But, the agenda of this tutorial is to just give you a hands-on-experience to how to code these classifiers.

Thank you.