Affective Computing

Prof. Jainendra Shukla
Department of Computer Science and Engineering
Indraprastha Institute of Information Technology, Delhi

Week - 08
Lecture - 27
Tutorial: Multimodal Emotion Recognition

(Refer Slide Time: 00:22)

Affective Computing

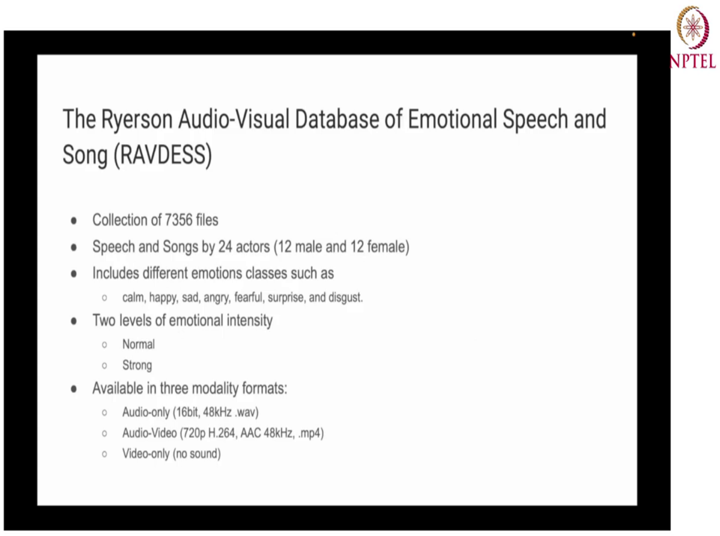
Tutorial on Multimodal Emotion Recognition

Hello everyone, welcome to this Affective Computing tutorial on Multi-modal Emotion Recognition. In this tutorial we will try to learn emotion recognition using audio and video information together.

By integrating information from different modality such as audio and video, one can gain a more complete and accurate understanding of person's emotion state and there are multiple different techniques for integrating information from multiple modalities including feature level fusion, decision level fusion and hybrid approaches that combine both.

Each approach has its own strength and weakness and the choice of technique will depend upon specific application and data availability.

(Refer Slide Time: 01:09)



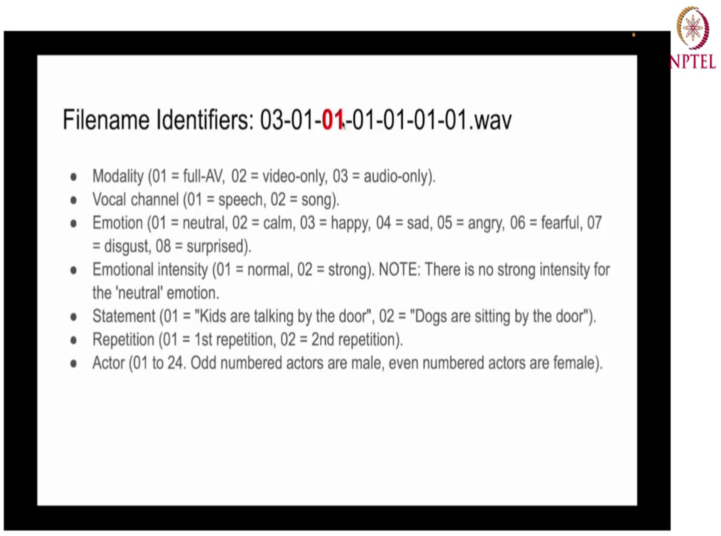
The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS)

- 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)

So, in this tutorial we will be using a RAVDESS dataset. The RAVDESS dataset is a popular dataset used for emotion recognition research. It contains recording of actors speaking and performing various emotional states including anger, disgust, fear, happiness, sadness and surprise.

The dataset includes both audio and video recordings which makes it a well-suited for multi-modal emotion recognition research. The audio recordings consist of 24 actors speaking scripted sentences in each of six emotional states. The video recording consist of same actor performing facial expression and body movements to convey the same six emotional states.

(Refer Slide Time: 01:58)

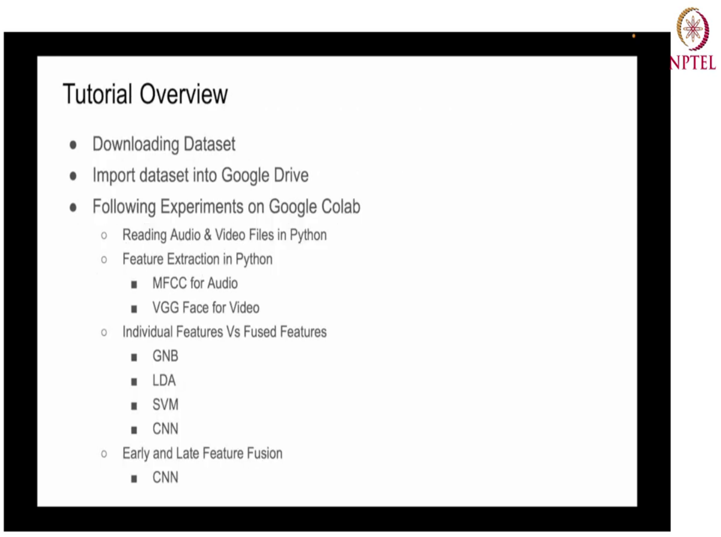


Filename Identifiers: 03-01-01-01-01-01-01.wav

- 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).

Now, coming to the filename convention in this dataset, each filename consist of seven numerical identifier in which third identifier tells about the emotional class.

(Refer Slide Time: 02:12)



Tutorial Overview

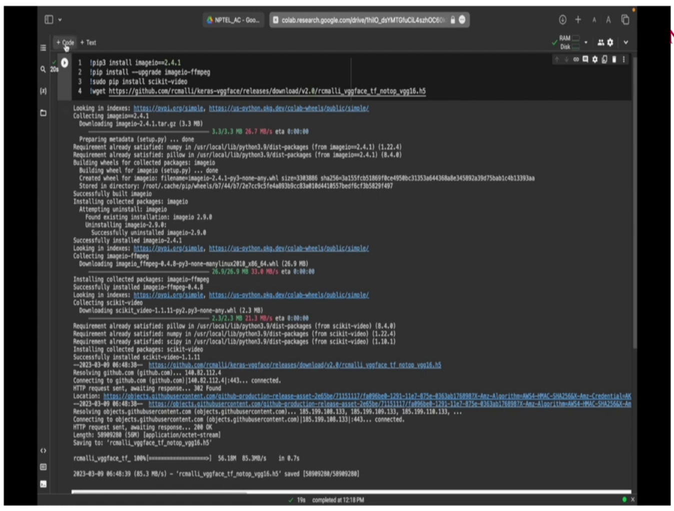
- Downloading Dataset
- Import dataset into Google Drive
- Following Experiments on Google Colab
  - Reading Audio & Video Files in Python
  - Feature Extraction in Python
    - MFCC for Audio
    - VGG Face for Video
  - Individual Features Vs Fused Features
    - GNB
    - LDA
    - SVM
    - CNN
  - Early and Late Feature Fusion
    - CNN

So, to get started with the multi-modal emotion recognition using the RAVDESS dataset, we will first need to import audio and video data, an extractor element feature from both modalities. After that we can use variety of machine learning models to classify the emotional states based on the extracted features. It is worth noting that the performance of your multi-modal emotion recognition system will depend on the quality of the feature extracted and fusion techniques used.

As well as the choice of machine learning model so, it is important to carefully evaluate the performance of the system and fine tune the network. So, we will start with downloading and coding the dataset into the Google Drive. After that, we will be using some predefined Python libraries to read audio and video files in Python environment. Later we will do some feature extraction, particularly this MFCC for audio data. And we will extract video feature using VGG 16 phase model.

After that, we will try to see individual features versus fused features performance. And we will be using Gaussian base linear discriminant analysis, support vector machine and our CNN network for this analysis. And later on we will be doing our early and late fusion and we will be using CNN for this. So, now let us jump directly upon the coding part.

(Refer Slide Time: 03:41)



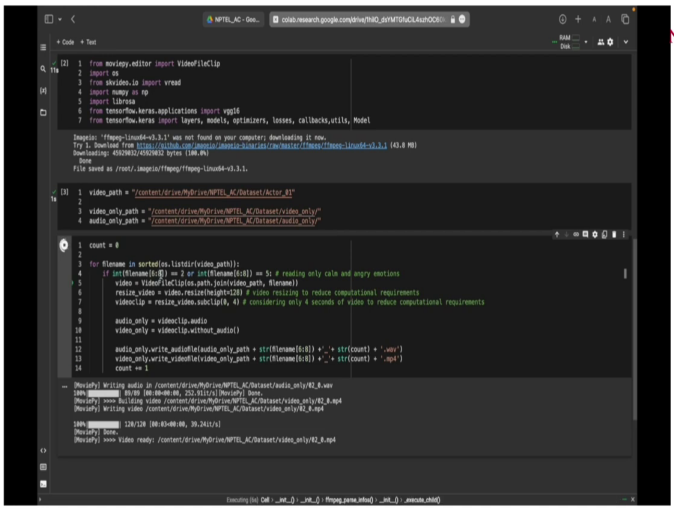
```
!pip3 install imageio==2.4.1
!pip install --upgrade imageio-ffmpeg
!sudo pip install scikit-video
!wget https://github.com/rcmalli/keras-vggface/releases/download/v2.0/rcmalli_vggface_tf_notop_vgg16.h5
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting imageio==2.4.1
  Downloading imageio-2.4.1.tar.gz (3.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.3/3.3 MB 26.7 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy in /usr/local/lib/python3.9/dist-packages (from imageio==2.4.1) (1.22.4)
Requirement already satisfied: pillow in /usr/local/lib/python3.9/dist-packages (from imageio==2.4.1) (8.4.0)
Building wheels for collected packages: imageio
  Building wheel for imageio (setup.py) ... done
  Created wheel for imageio: filename=imageio-2.4.1-py3-none-any.whl size=3303886 sha256=3a155fcb51869f0ce4950bc31353a644368a8e345892a39d75bab1c4b13393aa
  Stored in directory: /root/.cache/pip/wheels/b7/44/b7/2e7cc9c5fe4a893b9cc83a010d4410557bedf6cf3b5829f497
Successfully built imageio
Installing collected packages: imageio
  Attempting uninstall: imageio
    Found existing installation: imageio 2.9.0
    Uninstalling imageio-2.9.0:
      Successfully uninstalled imageio-2.9.0
Successfully installed imageio-2.4.1
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting imageio-ffmpeg
  Downloading imageio_ffmpeg-0.4.8-py3-none-manylinux2010_x86_64.whl (26.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 26.9/26.9 MB 33.0 MB/s eta 0:00:00
Installing collected packages: imageio-ffmpeg
Successfully installed imageio-ffmpeg-0.4.8
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting scikit-video
  Downloading scikit_video-1.1.11-py2.py3-none-any.whl (2.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.3/2.3 MB 21.3 MB/s eta 0:00:00
Requirement already satisfied: pillow in /usr/local/lib/python3.9/dist-packages (from scikit-video) (8.4.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.9/dist-packages (from scikit-video) (1.22.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.9/dist-packages (from scikit-video) (1.10.1)
Installing collected packages: scikit-video
Successfully installed scikit-video-1.1.11
--2023-03-09 06:48:38--  https://github.com/rcmalli/keras-vggface/releases/download/v2.0/rcmalli_vggface_tf_notop_vgg16.h5
Resolving github.com (github.com)... 140.82.112.4
Connecting to github.com (github.com)|140.82.112.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asset-2e65be/71151117/fa096be0-1291-11e7-875e-0363ab1768987X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AK
--2023-03-09 06:48:38--  https://objects.githubusercontent.com/github-production-release-asset-2e65be/71151117/fa096be0-1291-11e7-875e-0363ab1768987X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Am
Resolving objects.githubusercontent.com (objects.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to objects.githubusercontent.com (objects.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 58909280 (56M) [application/octet-stream]
Saving to: 'rcmalli_vggface_tf_notop_vgg16.h5'

rcmalli_vggface_tf_ 100%[===================>]  56.18M  85.3MB/s    in 0.7s

2023-03-09 06:48:39 (85.3 MB/s) - 'rcmalli_vggface_tf_notop_vgg16.h5' saved [58909280/58909280]
```

We will start with installing couple of libraries that might not be pre-installed in the Google colab environment. Once we have installed all these necessary libraries, we can begin with importing them in the code and we will start pre-processing the data and extract given feature for our multi-modal emotional recognition.

And I am also assuming that you have already downloaded the dataset and stored it into your respective Google drives. So, our downloading library code will look something like this. And it might take a couple of seconds to download all these files.

(Refer Slide Time: 04:37)



```
from moviepy.editor import VideoFileClip
import os
from skvideo.io import vread
import numpy as np
import librosa
from tensorflow.keras.applications import vgg16
from tensorflow.keras import layers, models, optimizers, losses, callbacks,utils, Model
```

```
Imageio: 'ffmpeg-linux64-v3.3.1' was not found on your computer; downloading it now.
Try 1. Download from https://github.com/imageio/imageio-binaries/raw/master/ffmpeg/ffmpeg-linux64-v3.3.1 (43.8 MB)
Downloading: 45929032/45929032 bytes (100.0%)
  Done
File saved as /root/.imageio/ffmpeg/ffmpeg-linux64-v3.3.1.
```

```
video_path = "/content/drive/MyDrive/NPTEL_AC/Dataset/Actor_01"

video_only_path = "/content/drive/MyDrive/NPTEL_AC/Dataset/video_only/"
audio_only_path = "/content/drive/MyDrive/NPTEL_AC/Dataset/audio_only/"
```

```
count = 0

for filename in sorted(os.listdir(video_path)):
    if int(filename[6:8]) == 2 or int(filename[6:8]) == 5: # reading only calm and angry emotions
        video = VideoFileClip(os.path.join(video_path, filename))
        resize_video = video.resize(height=128) # video resizing to reduce computational requirements
        videoclip = resize_video.subclip(0, 4) # considering only 4 seconds of video to reduce computational requirements

        audio_only = videoclip.audio
        video_only = videoclip.without_audio()

        audio_only.write_audiofile(audio_only_path + str(filename[6:8]) +'_'+ str(count) + '.wav')
        video_only.write_videofile(video_only_path + str(filename[6:8]) +'_'+ str(count) + '.mp4')
        count += 1
```

```
[MoviePy] Writing audio in /content/drive/MyDrive/NPTEL_AC/Dataset/audio_only/02_0.wav
100%|██████████| 89/89 [00:00<00:00, 252.91it/s][MoviePy] Done.
[MoviePy] >>>> Building video /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/02_0.mp4
[MoviePy] Writing video /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/02_0.mp4

100%|██████████| 120/120 [00:03<00:00, 39.24it/s]
[MoviePy] Done.
[MoviePy] >>>> Video ready: /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/02_0.mp4
```

So, after downloading my relevant packages, I will import all my relevant libraries for this tutorial and the code will look something like this. After downloading and importing the libraries, I will start with defining some path variables. In this tutorial, I will be basically defining three of my path variable. First will be our video path which essentially contains the exact dataset path or video files are stored.

And then I will also define two other paths where I will be storing video only information and audio only information extracted from the original dataset. One more thing, I will be just using actor 1 data for this analysis and the only purpose of using actor1 data is to reduce the computational demand.

So, as of now, my path's variables are all set. So, I will write first piece of code which will essentially take our video files from original data and extract video and audio information from that video file and save it to two separate folder. And the code will look something like this.

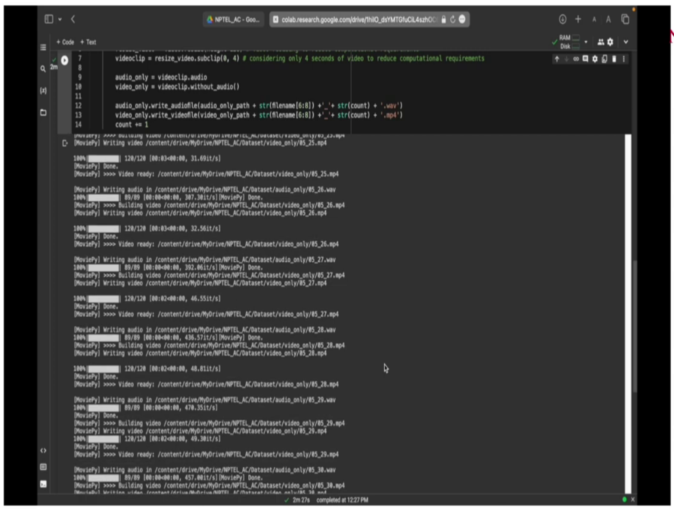
So, let me give you an overall overview of this piece of code but we are essentially doing here is we are iterating over all the file in the dataset folder and we will be only using emotion number 2 and emotion number 5, which essentially corresponds to calm and anger emotions.

The only reason I am using only two emotion classes just reducing down the computational power. As in the general case, Google colab only consist of 12 GB of RAM. For computational purpose only, I will be using just two of the emotion classes. Then after reading the video from their location, I will be resizing them into a smaller size and again, the reason is to reduce the computational requirement.

And after that, I will simply I will be simply considering first four second of the video to reduce the computational requirement and also to make a consistency among all the videos. Like there could be some instance that a particular video might consist of 5 second and other might be consisting of 4 second.

So, I am using a smallest threshold like 4 second and every video so, that all my data will be consisting along the time dimension. Now, I will extract audio information only using this command video clip dot audio. And later I will be storing these separate audio and video content in these two folders. So, now running this code and this code might take good amount of time as we are reading and writing the data.

(Refer Slide Time: 07:35)



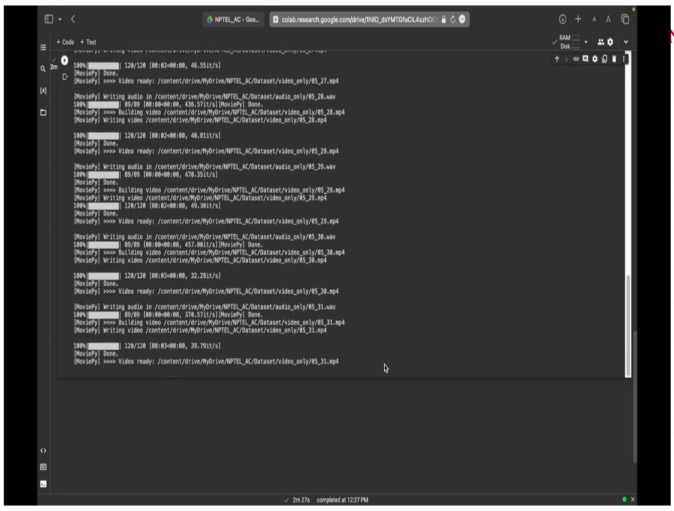
```
    videoclip = resize_video.subclip(0, 4) # considering only 4 seconds of video to reduce computational requirements

    audio_only = videoclip.audio
    video_only = videoclip.without_audio()

    audio_only.write_audiofile(audio_only_path + str(filename[6:8]) +'_'+ str(count) + '.wav')
    video_only.write_videofile(video_only_path + str(filename[6:8]) +'_'+ str(count) + '.mp4')
    count += 1
```

```
[MoviePy] Writing video /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/05_25.mp4

100%|██████████| 120/120 [00:03<00:00, 31.69it/s]
[MoviePy] Done.
[MoviePy] >>>> Video ready: /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/05_25.mp4

[MoviePy] Writing audio in /content/drive/MyDrive/NPTEL_AC/Dataset/audio_only/05_26.wav
100%|██████████| 89/89 [00:00<00:00, 307.30it/s][MoviePy] Done.
[MoviePy] >>>> Building video /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/05_26.mp4
[MoviePy] Writing video /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/05_26.mp4

100%|██████████| 120/120 [00:03<00:00, 32.56it/s]
[MoviePy] Done.
[MoviePy] >>>> Video ready: /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/05_26.mp4

[MoviePy] Writing audio in /content/drive/MyDrive/NPTEL_AC/Dataset/audio_only/05_27.wav
100%|██████████| 89/89 [00:00<00:00, 392.06it/s][MoviePy] Done.
[MoviePy] >>>> Building video /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/05_27.mp4
[MoviePy] Writing video /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/05_27.mp4

100%|██████████| 120/120 [00:02<00:00, 46.55it/s]
[MoviePy] Done.
[MoviePy] >>>> Video ready: /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/05_27.mp4

[MoviePy] Writing audio in /content/drive/MyDrive/NPTEL_AC/Dataset/audio_only/05_28.wav
100%|██████████| 89/89 [00:00<00:00, 436.57it/s][MoviePy] Done.
[MoviePy] >>>> Building video /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/05_28.mp4
[MoviePy] Writing video /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/05_28.mp4

100%|██████████| 120/120 [00:02<00:00, 48.81it/s]
[MoviePy] Done.
[MoviePy] >>>> Video ready: /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/05_28.mp4

[MoviePy] Writing audio in /content/drive/MyDrive/NPTEL_AC/Dataset/audio_only/05_29.wav
100%|██████████| 89/89 [00:00<00:00, 470.35it/s]
[MoviePy] Done.
[MoviePy] >>>> Building video /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/05_29.mp4
[MoviePy] Writing video /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/05_29.mp4
100%|██████████| 120/120 [00:02<00:00, 49.30it/s]
[MoviePy] Done.
[MoviePy] >>>> Video ready: /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/05_29.mp4

[MoviePy] Writing audio in /content/drive/MyDrive/NPTEL_AC/Dataset/audio_only/05_30.wav
100%|██████████| 89/89 [00:00<00:00, 457.00it/s][MoviePy] Done.
[MoviePy] >>>> Building video /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/05_30.mp4
```

NPTEL_AC - Goo...
colab.research.google.com/drive/1hilO_dsYMTGfuCiL4szhO
+ Code + Text
2m 27s completed at 12:27 PM

Audio and video files are created from the original dataset. Now, I will write another piece of code to bring those audio only and video only files in my Python environment.

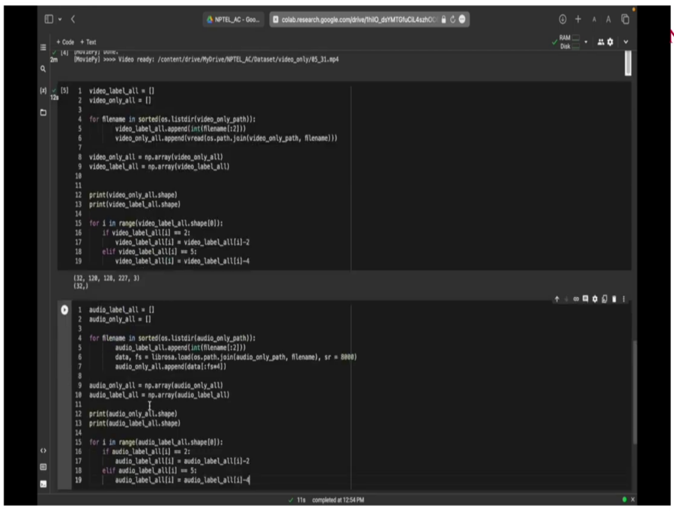
(Refer Slide Time: 07:50)



```
100%|██████████| 120/120 [00:02<00:00, 46.55it/s]
[MoviePy] Done.
[MoviePy] >>>> Video ready: /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/05_27.mp4

[MoviePy] Writing audio in /content/drive/MyDrive/NPTEL_AC/Dataset/audio_only/05_28.wav
100%|██████████| 89/89 [00:00<00:00, 436.57it/s][MoviePy] Done.
[MoviePy] >>>> Building video /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/05_28.mp4
[MoviePy] Writing video /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/05_28.mp4

100%|██████████| 120/120 [00:02<00:00, 48.81it/s]
[MoviePy] Done.
[MoviePy] >>>> Video ready: /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/05_28.mp4

[MoviePy] Writing audio in /content/drive/MyDrive/NPTEL_AC/Dataset/audio_only/05_29.wav
100%|██████████| 89/89 [00:00<00:00, 470.35it/s]
[MoviePy] Done.
[MoviePy] >>>> Building video /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/05_29.mp4
[MoviePy] Writing video /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/05_29.mp4
100%|██████████| 120/120 [00:02<00:00, 49.30it/s]
[MoviePy] Done.
[MoviePy] >>>> Video ready: /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/05_29.mp4

[MoviePy] Writing audio in /content/drive/MyDrive/NPTEL_AC/Dataset/audio_only/05_30.wav
100%|██████████| 89/89 [00:00<00:00, 457.00it/s][MoviePy] Done.
[MoviePy] >>>> Building video /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/05_30.mp4
[MoviePy] Writing video /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/05_30.mp4

100%|██████████| 120/120 [00:03<00:00, 32.29it/s]
[MoviePy] Done.
[MoviePy] >>>> Video ready: /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/05_30.mp4

[MoviePy] Writing audio in /content/drive/MyDrive/NPTEL_AC/Dataset/audio_only/05_31.wav
100%|██████████| 89/89 [00:00<00:00, 370.57it/s][MoviePy] Done.
[MoviePy] >>>> Building video /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/05_31.mp4
[MoviePy] Writing video /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/05_31.mp4

100%|██████████| 120/120 [00:03<00:00, 39.79it/s]
[MoviePy] Done.
[MoviePy] >>>> Video ready: /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/05_31.mp4
```

And my code will look something like this.

(Refer Slide Time: 07:55)



```
[MoviePy] >>>> Video ready: /content/drive/MyDrive/NPTEL_AC/Dataset/video_only/05_31.mp4
```

```python
video_label_all = []
video_only_all = []

for filename in sorted(os.listdir(video_only_path)):
        video_label_all.append(int(filename[:2]))
        video_only_all.append(vread(os.path.join(video_only_path, filename)))

video_only_all = np.array(video_only_all)
video_label_all = np.array(video_label_all)


print(video_only_all.shape)
print(video_label_all.shape)

for i in range(video_label_all.shape[0]):
    if video_label_all[i] == 2:
        video_label_all[i] = video_label_all[i]-2
    elif video_label_all[i] == 5:
        video_label_all[i] = video_label_all[i]-4
```

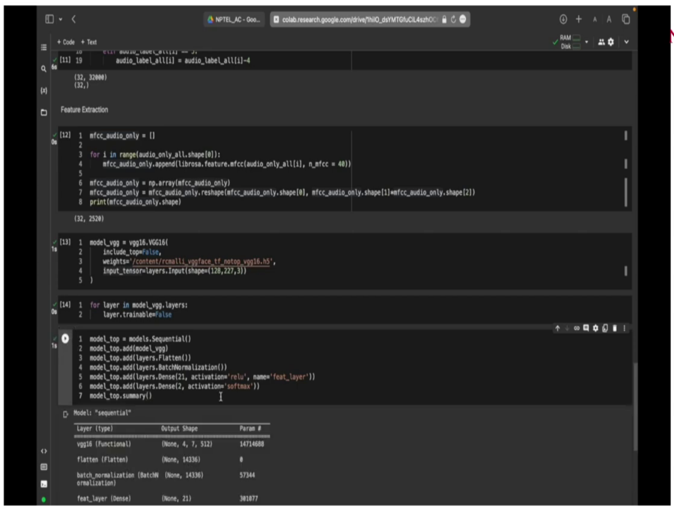
```
(32, 120, 128, 227, 3)
(32,)
```

```python
audio_label_all = []
audio_only_all = []

for filename in sorted(os.listdir(audio_only_path)):
        audio_label_all.append(int(filename[:2]))
        data, fs = librosa.load(os.path.join(audio_only_path, filename), sr = 8000)
        audio_only_all.append(data[:fs*4])

audio_only_all = np.array(audio_only_all)
audio_label_all = np.array(audio_label_all)

print(audio_only_all.shape)
print(audio_label_all.shape)

for i in range(audio_label_all.shape[0]):
    if audio_label_all[i] == 2:
        audio_label_all[i] = audio_label_all[i]-2
    elif audio_label_all[i] == 5:
        audio_label_all[i] = audio_label_all[i]-4
```

So, I will be just declaring to list video label all and video only all these two list will be storing the labels and the exact video only data. And I will be iterating through all the files in my video only path and appending into these two list. Later I will convert these two list into NumPy array for our ease of use.

And I will write another piece of small code which will essentially convert our label into 0 1 binary labels. After execution of this code, I will be using similar sort of code for bringing audio only data in my Python environment and my code will look something like this.

(Refer Slide Time: 09:15)



```
audio_label_all[i] = audio_label_all[i]-4
```

```
(32, 32000)
(32,)
```

Feature Extraction

```
mfcc_audio_only = []

for i in range(audio_only_all.shape[0]):
    mfcc_audio_only.append(librosa.feature.mfcc(audio_only_all[i], n_mfcc = 40))

mfcc_audio_only = np.array(mfcc_audio_only)
mfcc_audio_only = mfcc_audio_only.reshape(mfcc_audio_only.shape[0], mfcc_audio_only.shape[1]*mfcc_audio_only.shape[2])
print(mfcc_audio_only.shape)
```

```
(32, 2520)
```

```
model_vgg = vgg16.VGG16(
    include_top=False,
    weights='/content/rcmalli_vggface_tf_notop_vgg16.h5',
    input_tensor=layers.Input(shape=(128,227,3))
)
```

```
for layer in model_vgg.layers:
    layer.trainable=False
```

```
model_top = models.Sequential()
model_top.add(model_vgg)
model_top.add(layers.Flatten())
model_top.add(layers.BatchNormalization())
model_top.add(layers.Dense(21, activation='relu', name='feat_layer'))
model_top.add(layers.Dense(2, activation='softmax'))
model_top.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 4, 7, 512)         14714688

 flatten (Flatten)           (None, 14336)             0

 batch_normalization (BatchN  (None, 14336)            57344
 ormalization)

 feat_layer (Dense)          (None, 21)                301077
```

So, now as you can see that I have bought my video only and audio only data in my Python environment and the shape of video only data will look like 32 cross 120 cross 128 cross 227 cross 3.

So, what does these number actually mean is like there are 32 files there are 32 video files and each video files contain 120 frames since the video data was sampled at 30 hertz and we are taking just four second of data. So, there will be 120 frames and each frame will be consisting of 128 cross 227 dimension and there will be three channel red, green and blue.

Similarly, for our audio data we will be considering first four second of audio and all the audio will be sampled at 8000 hertz which essentially means that for one second of data there will be 8000 samples and using this strategy we will be collecting all those 32 files and each file will be consisting of 32000 sample which essentially means 4 second multiplied by 8000 sample rate. You and video only files in my Python environment I will start with the feature extraction part.

In our feature extraction part. So, in our feature extraction part the first feature will be mfcc feature which essentially called as MEL frequency cepstral coefficient and it is a commonly used feature extraction technique for audio analysis including in the text of multimodal emotion recognition. mfcc represent a compact yet efficient way of capturing the spectral characteristics of an audio signal and they have been shown to be effective at capturing emotion information in speed signals.

So, there are some basic steps that involve in I am putting the mfcc from audio signal and these steps consist of the very first step consist of applying a high passed filter to the signal to amplify the higher frequency component and reducing the impact of lower frequency components. Then second step mfcc process is frame segmentation where the signal is divided into short frames usually 20 to 30 milliseconds with some overlap.

Third step is windowing. A window function such as hamming window is applied to each frame to reduce the spectral leakage and to smooth of the signal and then we perform a Fourier transform where that windowed signal is transformed into frequency domain using the discrete Fourier transformation and after that there will be MEL frequency whopping. Transform DFT spectrum will be converted into a MEL frequency scale which essentially approximates the way human perceive the sound.

After that there is a usual cepstral analysis over these whop MEL frequencies. The MEL frequency spectrum is transformed into cepstral domain using inverse of discrete cosine transformation and then we select a subset of resulting cepstral coefficient as the mfcc coefficients.

These mfcc coefficients can be used as a feature for emotion recognition classification either on their own or maybe with the combination of other feature and for our purpose we will be using a standard library called librosa which will provide the function to easily compute mfcc from audio signals and to do so, our code will look something like this.

In this code we have declared a list where we will be storing our mfcc coefficients and we will be iterating through all the audio only files and we will be using this predefined function librosa dot feature dot mfcc and passing each file in this predefined function and there is a parameter in this function called number of mfcc coefficient and for that we have chosen a value equal to 40.

And after computation of these mfcc coefficients we will convert that list into a array and we will reshape this mfcc list as number of sample followed by number of coefficients. So, after extracting the mfcc features from audio files we will move towards extracting video features from the video only files. To do so, we will take the advantage of VGG16 phase architecture.

So, VGG16 is basically a deep convolution neural network architecture that was originally designed for image classification task. This architecture for image classification consist of 16 layers including 13 convolution layers and 3 fully connected layers. In the standard architecture input to the network is an RTP image of size 224 cross 224 pixels and this input layer will be followed by 13 convolution layers each convolution layer with a 3 cross 3 filter size and a stride of 1 pixel.

In this architecture the number of filter increases as we go deeper into the network basically ranging from 64 in the first layer to 512 in the last layer. Then after every two convolution layer a max pooling operation is applied to reduce the spatial resolution of the feature map through the convolution future maps coming through the convolution layers and after that a fully connected layer is applied which consist of 4096 neuron each and after that in the standard architecture there is a three layered multi-layer perceptron for final prediction and the final prediction is performed using softmax layer.

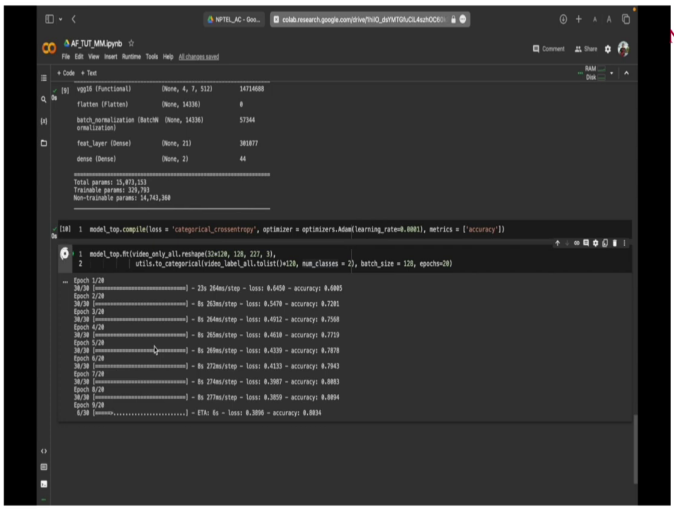
So, in our architecture we will be using this standard VGG16 phase model it is a pre-trained model and to do so, our code will look something like this and in this code instead of using 224 cross 224 image we will be using 128 cross 227 dimensional image and also I have downloaded this file this pre-trained weight file in my local drive and I have already given you the exact code for downloading this file.

So, after running this file our model is initiated. So, one essential step that we have to do in this pre-trained model is to freeze down all the convolution layer so, that they can extract relevant feature according to their pre-trained settings. To do so, my code will look something like this where I will iterate through all the layers in my vgg model and I will simply passed an argument layer dot trainable equal to false.

After making our feature learning layers as a non-trainable layers, we can put a another multi-layer perceptron over it on the code will look something like this. So, what does this over the top model is doing and this is taking our vgg model architecture and then we are simply putting a flattened layer which will flat down the extracted features into a single vector and then we will be perform batch normalization and then we put our 21 neuron layer with the activation of relu and we will name this layer as a feature layer.

So, this will be our main layer for extracting the features and then we will simply perform a classification operation on our dataset and extract the feature learned at this layer.

(Refer Slide Time: 18:38)



```
vgg16 (Functional)          (None, 4, 7, 512)         14714688

flatten (Flatten)           (None, 14336)             0

batch_normalization (BatchN  (None, 14336)            57344
ormalization)

feat_layer (Dense)          (None, 21)                301077

dense (Dense)               (None, 2)                 44

=================================================================
Total params: 15,073,153
Trainable params: 329,793
Non-trainable params: 14,743,360
_________________________________________________________________
```

```python
model_top.compile(loss = 'categorical_crossentropy', optimizer = optimizers.Adam(learning_rate=0.0001), metrics = ['accuracy'])
```

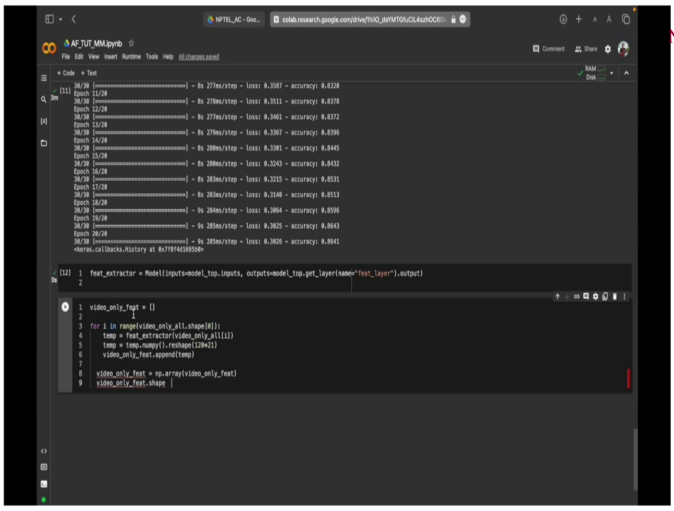
```python
model_top.fit(video_only_all.reshape(32*120, 128, 227, 3),
              utils.to_categorical(video_label_all.tolist()*120, num_classes = 2), batch_size = 128, epochs=20)
```

```
Epoch 1/20
30/30 [==============================] - 23s 264ms/step - loss: 0.6450 - accuracy: 0.6005
Epoch 2/20
30/30 [==============================] - 8s 263ms/step - loss: 0.5470 - accuracy: 0.7201
Epoch 3/20
30/30 [==============================] - 8s 264ms/step - loss: 0.4912 - accuracy: 0.7568
Epoch 4/20
30/30 [==============================] - 8s 265ms/step - loss: 0.4610 - accuracy: 0.7719
Epoch 5/20
30/30 [==============================] - 8s 269ms/step - loss: 0.4339 - accuracy: 0.7878
Epoch 6/20
30/30 [==============================] - 8s 272ms/step - loss: 0.4133 - accuracy: 0.7943
Epoch 7/20
30/30 [==============================] - 8s 274ms/step - loss: 0.3987 - accuracy: 0.8083
Epoch 8/20
30/30 [==============================] - 8s 277ms/step - loss: 0.3859 - accuracy: 0.8094
Epoch 9/20
 6/30 [=====>........................] - ETA: 6s - loss: 0.3896 - accuracy: 0.8034
```

So, you can see that our vgg model has given 4 cross 7 cross 5 12 dimensional feature shape then we have flatten that out and after performing the batch normalization we will be taking this 21 dimensional feature. Now, I can simply compile my model using categorical cross entropy loss and with my Adam optimizer with learning rate equal to 0.0001.

After this I can simply fit my model with the video data for two classes with a batch size of 128 and number of epochs let us say 20. So, fitting this model over the 20 box this thing also might take some time. So, please have some patience.

(Refer Slide Time: 20:17)



```
30/30 [==============================] - 8s 277ms/step - loss: 0.3587 - accuracy: 0.8320
Epoch 11/20
30/30 [==============================] - 8s 278ms/step - loss: 0.3511 - accuracy: 0.8378
Epoch 12/20
30/30 [==============================] - 8s 277ms/step - loss: 0.3461 - accuracy: 0.8372
Epoch 13/20
30/30 [==============================] - 8s 279ms/step - loss: 0.3367 - accuracy: 0.8396
Epoch 14/20
30/30 [==============================] - 8s 280ms/step - loss: 0.3301 - accuracy: 0.8445
Epoch 15/20
30/30 [==============================] - 8s 280ms/step - loss: 0.3243 - accuracy: 0.8432
Epoch 16/20
30/30 [==============================] - 8s 283ms/step - loss: 0.3215 - accuracy: 0.8531
Epoch 17/20
30/30 [==============================] - 8s 283ms/step - loss: 0.3140 - accuracy: 0.8513
Epoch 18/20
30/30 [==============================] - 9s 284ms/step - loss: 0.3064 - accuracy: 0.8596
Epoch 19/20
30/30 [==============================] - 9s 285ms/step - loss: 0.3025 - accuracy: 0.8643
Epoch 20/20
30/30 [==============================] - 9s 285ms/step - loss: 0.3026 - accuracy: 0.8641
<keras.callbacks.History at 0x7f0f4d1695b0>
```

```python
feat_extractor = Model(inputs=model_top.inputs, outputs=model_top.get_layer(name="feat_layer").output)
```

```python
video_only_feat = []

for i in range(video_only_all.shape[0]):
    temp = feat_extractor(video_only_all[i])
    temp = temp.numpy().reshape(120*21)
    video_only_feat.append(temp)

video_only_feat = np.array(video_only_feat)
video_only_feat.shape
```

So, as you can see we have already trained our model now and we are getting a accuracy somewhat around 86.41 percent which is a good discriminating accuracy saying that we are able to discriminate between the two emotion classes using VGG16 feature. Now, we will simply write a code to extract that feature layer this feature layer for that I will write a simple code there. I will create a feature extractor which is essentially taking model input as our input and output will be the intermediate layer which is our feature layer.

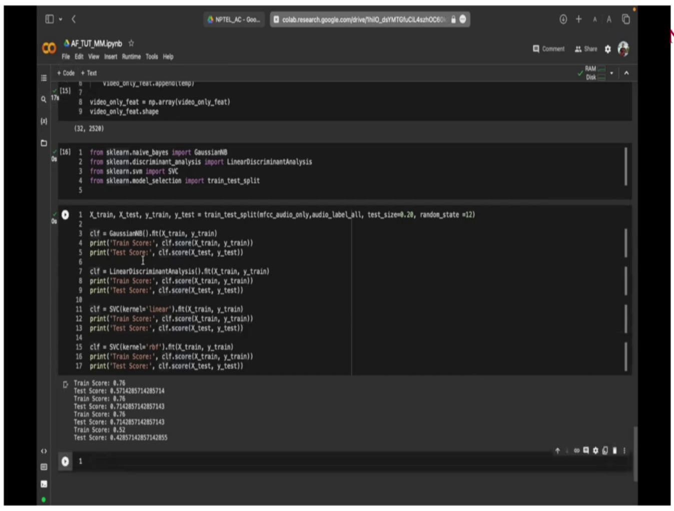
Now, I can pass all my data point to this feature extractor and we will get the corresponding features. For that I can write a simple code and the code will look something like this. Here I have declared a list called video only feature list and I will iterate through all the files and I will pass those file into this feature extractor function over here.

And then I will simply extract the feature convert them into the numpy array and I will append these features to a this video only feature list and I will convert that list into a numpy array for our ease of use. One another thing that you have to look here is after extracting the numpy array I am reshaping it to 120 cross 21.

So, what does this basically means is since our original data consist of 32 files each files containing 120 images of dimension 128 cross 227 cross 3. So, basically for each file I am passing all these 120 images and for one particular image I am getting a 21 dimensional feature from my VGG 16 architecture.

Now, I am simply appending these feature sequentially so, that I can get a long feature for these 120 images and these 120 images basically correspond to our 4 seconds of video. So, in this way I am creating feature for all these 32 images.

(Refer Slide Time: 23:11)



```
    video_only_feat.append(temp)

video_only_feat = np.array(video_only_feat)
video_only_feat.shape
```

```
(32, 2520)
```

```
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(mfcc_audio_only,audio_label_all, test_size=0.20, random_state =12)

clf = GaussianNB().fit(X_train, y_train)
print('Train Score:', clf.score(X_train, y_train))
print('Test Score:', clf.score(X_test, y_test))

clf = LinearDiscriminantAnalysis().fit(X_train, y_train)
print('Train Score:', clf.score(X_train, y_train))
print('Test Score:', clf.score(X_test, y_test))

clf = SVC(kernel='linear').fit(X_train, y_train)
print('Train Score:', clf.score(X_train, y_train))
print('Test Score:', clf.score(X_test, y_test))

clf = SVC(kernel='rbf').fit(X_train, y_train)
print('Train Score:', clf.score(X_train, y_train))
print('Test Score:', clf.score(X_test, y_test))
```

```
Train Score: 0.76
Test Score: 0.5714285714285714
Train Score: 0.76
Test Score: 0.7142857142857143
Train Score: 0.76
Test Score: 0.7142857142857143
Train Score: 0.52
Test Score: 0.42857142857142855
```

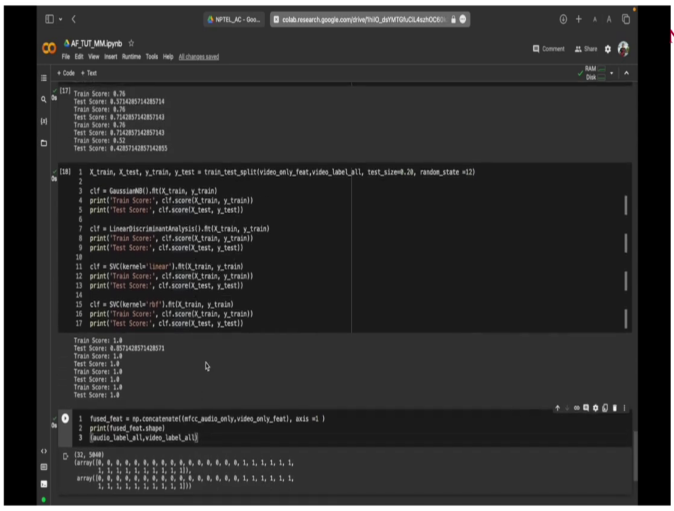
So, running this code we got 32 files each having dimension 2520. So, by now we have created our audio features as well as our video features. Now, we will simply first try to see how these audio feature and video features are working with respect to our basic classifier which our Gaussian base, linear discriminant analysis and our support vector machine. To do so, I will import all these classifiers from the standard sklearn library and my classification code will look something like this.

Here first of all I have I will be using my audio features which are mfcc audio only features and I will divide these feature into my train and test plate where my test split size will be 20 percent and I will first call my Gaussian base classifier and fit that data on my training set and then again I will call my linear discriminant analysis classifier then support vector machine with linear kernel and after that support vector machine with my RBF kernel.

Let us see how these feature are actually performing with these basic classifiers. As you can see we are getting a training score of somewhat around 76 percent using Gaussian base linear discriminant analysis and our support vector classifiers and support vector machine with RBF kernel is giving a lower score. So, which can also signify that the discriminating boundary between these two classes are more of a linear boundary.

After this we will perform same thing with our video features and the code will look something like this.

(Refer Slide Time: 25:17)


NPTEL_AC - Goo...
colab.research.google.com/drive/1hilO_dsYMTGfuCiL4szhOC60
AF_TUT_MM.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

```
Train Score: 0.76
Test Score: 0.5714285714285714
Train Score: 0.76
Test Score: 0.7142857142857143
Train Score: 0.76
Test Score: 0.7142857142857143
Train Score: 0.52
Test Score: 0.42857142857142855
```

```
X_train, X_test, y_train, y_test = train_test_split(video_only_feat,video_label_all, test_size=0.20, random_state =12)

clf = GaussianNB().fit(X_train, y_train)
print('Train Score:', clf.score(X_train, y_train))
print('Test Score:', clf.score(X_test, y_test))

clf = LinearDiscriminantAnalysis().fit(X_train, y_train)
print('Train Score:', clf.score(X_train, y_train))
print('Test Score:', clf.score(X_test, y_test))

clf = SVC(kernel='linear').fit(X_train, y_train)
print('Train Score:', clf.score(X_train, y_train))
print('Test Score:', clf.score(X_test, y_test))

clf = SVC(kernel='rbf').fit(X_train, y_train)
print('Train Score:', clf.score(X_train, y_train))
print('Test Score:', clf.score(X_test, y_test))
```

```
Train Score: 1.0
Test Score: 0.8571428571428571
Train Score: 1.0
Test Score: 1.0
Train Score: 1.0
Test Score: 1.0
Train Score: 1.0
Test Score: 1.0
```

```
fused_feat = np.concatenate((mfcc_audio_only,video_only_feat), axis =1 )
print(fused_feat.shape)
(audio_label_all,video_label_all)
```

```
(32, 5040)
(array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1]),
 array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1]))
```

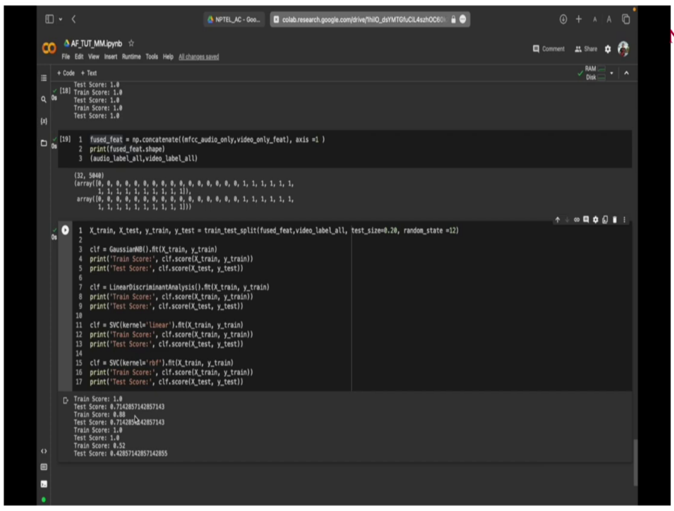
This is similar code this is similar code as we are doing with the audio feature. Instead of audio feature which I will simply use my video only features and they are corresponding label and the test size will again be 20 percent and we will split it into our train and test splits and then we will call our Gaussian base linear discriminant analysis support vector machine with linear kernel and support vector machine with our RBF kernel.

So, let us run this code and see how our video features are performing. So, as we can see our video features are really good and they are giving a accuracy of somewhat around 100 percent for each instances including our S codes. But since we are already getting this much of accuracy ah. So, let us try to see how our accuracy changes if we simply combine these two features together.

So, what I am basically trying to say is, I will simply concrete my audio features, audio only features and my video only features together and then I will pass these fused feature in my in these basic machine learning classifier and I will try to see how my accuracy changes with that. So, for my fusion part my code will look something like this. So, I have simply concatenated these two audio features and video features and let us see how the dimension goes ok.

So, we are having these 32 files and each having 5040 dimensional feature and I will also printed these my audio label and video label just to show you that these are same with respect to the indexing. So, we can use them interchangeably. Now, let us try to use our classification code and see how my fused features are working.

(Refer Slide Time: 27:37)



```
Test Score: 1.0
Train Score: 1.0
Test Score: 1.0
Train Score: 1.0
Test Score: 1.0
```

```
fused_feat = np.concatenate((mfcc_audio_only,video_only_feat), axis =1 )
print(fused_feat.shape)
(audio_label_all,video_label_all)
```

```
(32, 5040)
(array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1]),
 array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1]))
```

```
X_train, X_test, y_train, y_test = train_test_split(fused_feat,video_label_all, test_size=0.20, random_state =12)

clf = GaussianNB().fit(X_train, y_train)
print('Train Score:', clf.score(X_train, y_train))
print('Test Score:', clf.score(X_test, y_test))

clf = LinearDiscriminantAnalysis().fit(X_train, y_train)
print('Train Score:', clf.score(X_train, y_train))
print('Test Score:', clf.score(X_test, y_test))

clf = SVC(kernel='linear').fit(X_train, y_train)
print('Train Score:', clf.score(X_train, y_train))
print('Test Score:', clf.score(X_test, y_test))

clf = SVC(kernel='rbf').fit(X_train, y_train)
print('Train Score:', clf.score(X_train, y_train))
print('Test Score:', clf.score(X_test, y_test))
```

```
Train Score: 1.0
Test Score: 0.7142857142857143
Train Score: 0.88
Test Score: 0.7142857142857143
Train Score: 1.0
Test Score: 1.0
Train Score: 0.52
Test Score: 0.42857142857142855
```

So, for this I will simply reuse my code and instead of my video only features here I will use fused features and since our audio labels and video label are same with respect to the index, I can use either of the label again the test set size is 20 percent and let us see how it works.

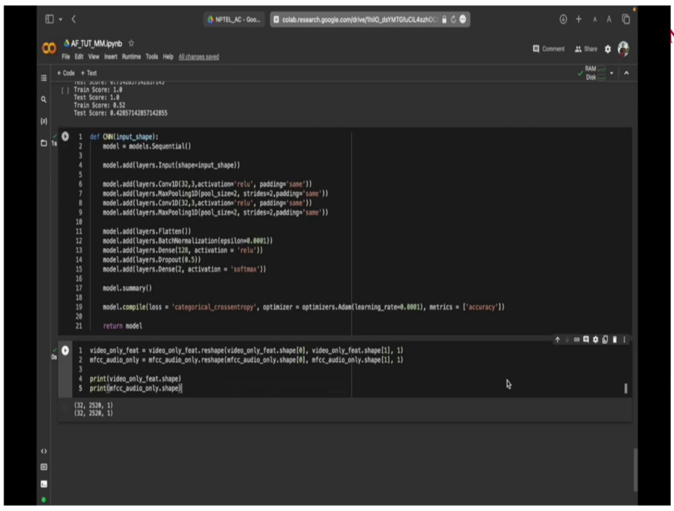
As we can see here is these are working slightly lesser as compared to our video only feature in our first case which is our GaussianNB case and also for our LdA case the accuracy is dropping, but for the SVM case we are getting a similar accuracies as our video classifier video features are giving.

So, this basically is saying that feature fusion is giving us a better or at least the same amount of performance with our SVM classifier and the reason why these classification accuracy decreases there could be multiple factors like one potential factor could be the curse of dimensionality since my feature dimension has too much high right now.

So, GaussianNBs and linear discriminant analysis might be suffering from this curse of dimensionality that is why they are giving a poor performance, but in case of a linear SVM I am getting a similar accuracy over here. So, till now we tried with traditional machine learning classifiers and now onwards we will be using 1 D CN architecture to classify the image and audio data encodings into emotion classes.

First, we will separately test the CNN architecture on audio and video data. Later on we will be using the fuse modality as the CNN input and in this case we are hypothesizing that the given audio and video embedding that we generated using MXCC and VGG face respectively are the two representation of the original data and since we have created these embeddings by separately concatenating each time frames we can treat these embedding as a time domain data and use 1D CNN classifier on top of it.

(Refer Slide Time: 30:31)



```
Train Score: 1.0
Test Score: 1.0
Train Score: 0.52
Test Score: 0.42857142857142855
```

```python
def CNN(input_shape):
    model = models.Sequential()

    model.add(layers.Input(shape=input_shape))

    model.add(layers.Conv1D(32,3,activation='relu', padding='same'))
    model.add(layers.MaxPooling1D(pool_size=2, strides=2,padding='same'))
    model.add(layers.Conv1D(32,3,activation='relu', padding='same'))
    model.add(layers.MaxPooling1D(pool_size=2, strides=2,padding='same'))

    model.add(layers.Flatten())
    model.add(layers.BatchNormalization(epsilon=0.0001))
    model.add(layers.Dense(128, activation = 'relu'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(2, activation = 'softmax'))

    model.summary()

    model.compile(loss = 'categorical_crossentropy', optimizer = optimizers.Adam(learning_rate=0.0001), metrics = ['accuracy'])

    return model
```

```python
video_only_feat = video_only_feat.reshape(video_only_feat.shape[0], video_only_feat.shape[1], 1)
mfcc_audio_only = mfcc_audio_only.reshape(mfcc_audio_only.shape[0], mfcc_audio_only.shape[1], 1)

print(video_only_feat.shape)
print(mfcc_audio_only.shape)
```

```
(32, 2520, 1)
(32, 2520, 1)
```

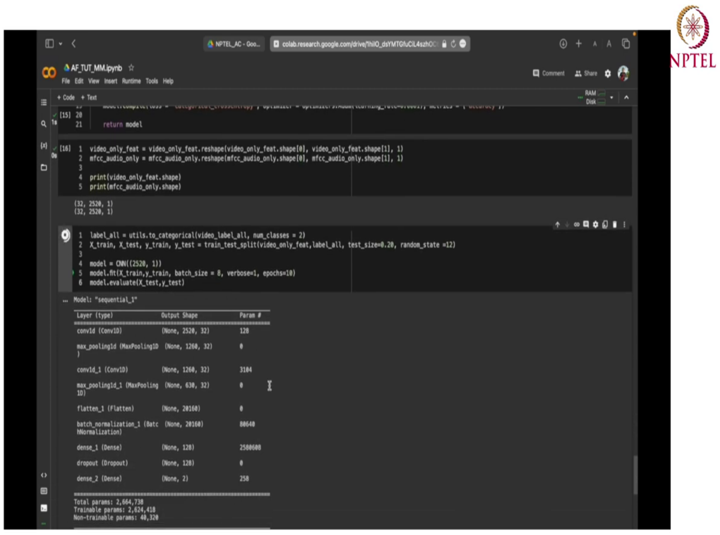
So, our CNN architecture will look something like this where we will be sequentially where we will be passing the input shape and there will be two convolution layers each convolution layer followed by a max pooling operation then we will flatten the output from the second max pool layer and we will pass through the flattened output through batch normalization layer.

And later we will collect and later we will be using 128 dimensional dense layer followed by a dropout layer for decreasing any chance of over fitting and later with the last dense layer which will be consisting of two neurons using softmax activation we will classify our given data into the emotion class.

Also, we will be using categorical cross entropy as a loss function and the optimizer we will be using is adam optimizer with learning rate of 0.0001 and the metric will be accuracy.

So, let me run this function. So, as our classifier is ready, we just need to reshape our video and mfcc audio features in a way that can be input into this CNN architecture. So, for that I will be using simple reshaping operations and our corresponding data will look something like this for both video and mfcc features.

(Refer Slide Time: 32:14)



```
    return model
```

```
video_only_feat = video_only_feat.reshape(video_only_feat.shape[0], video_only_feat.shape[1], 1)
mfcc_audio_only = mfcc_audio_only.reshape(mfcc_audio_only.shape[0], mfcc_audio_only.shape[1], 1)

print(video_only_feat.shape)
print(mfcc_audio_only.shape)
```

```
(32, 2520, 1)
(32, 2520, 1)
```

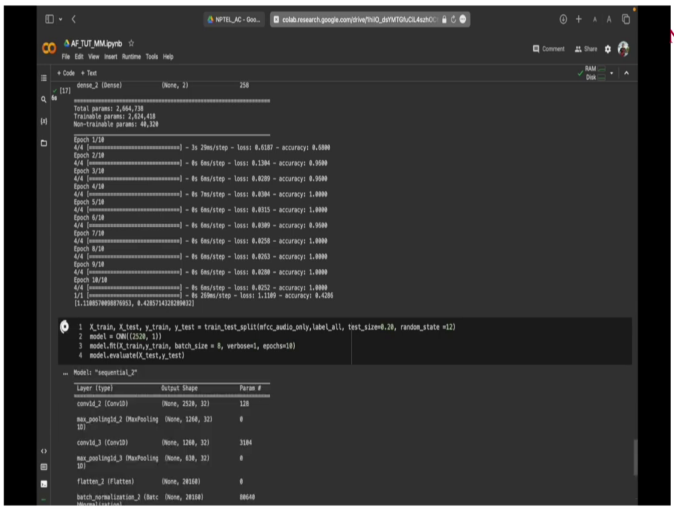
```
label_all = utils.to_categorical(video_label_all, num_classes = 2)
X_train, X_test, y_train, y_test = train_test_split(video_only_feat,label_all, test_size=0.20, random_state =12)

model = CNN((2520, 1))
model.fit(X_train,y_train, batch_size = 8, verbose=1, epochs=10)
model.evaluate(X_test,y_test)
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 2520, 32)          128

 max_pooling1d (MaxPooling1D  (None, 1260, 32)         0
 )

 conv1d_1 (Conv1D)           (None, 1260, 32)          3104

 max_pooling1d_1 (MaxPooling  (None, 630, 32)          0
 1D)

 flatten_1 (Flatten)         (None, 20160)             0

 batch_normalization_1 (Batc  (None, 20160)            80640
 hNormalization)

 dense_1 (Dense)             (None, 128)               2580608

 dropout (Dropout)           (None, 128)               0

 dense_2 (Dense)             (None, 2)                 258

=================================================================
Total params: 2,664,738
Trainable params: 2,624,418
Non-trainable params: 40,320
_________________________________________________________________
```

Later I will simply train our model and for that I will be first converting our labels into categorical labels as we are using categorical cross entropy function as our loss function and we will divide our data into respective train and test set with 20 percent test size and we will simply fit our model with batch size equal to 8 and epoch equal to 10. Let us see how it performs.

So, here you can see the summary of our given model and our model ok its already trained.

(Refer Slide Time: 32:50)



```
dense_2 (Dense)            (None, 2)             258

=================================================================
Total params: 2,664,738
Trainable params: 2,624,418
Non-trainable params: 40,320
_________________________________________________________________
Epoch 1/10
4/4 [==============================] - 3s 29ms/step - loss: 0.6187 - accuracy: 0.6800
Epoch 2/10
4/4 [==============================] - 0s 6ms/step - loss: 0.1304 - accuracy: 0.9600
Epoch 3/10
4/4 [==============================] - 0s 6ms/step - loss: 0.0289 - accuracy: 0.9600
Epoch 4/10
4/4 [==============================] - 0s 7ms/step - loss: 0.0304 - accuracy: 1.0000
Epoch 5/10
4/4 [==============================] - 0s 6ms/step - loss: 0.0315 - accuracy: 1.0000
Epoch 6/10
4/4 [==============================] - 0s 6ms/step - loss: 0.0309 - accuracy: 0.9600
Epoch 7/10
4/4 [==============================] - 0s 6ms/step - loss: 0.0258 - accuracy: 1.0000
Epoch 8/10
4/4 [==============================] - 0s 6ms/step - loss: 0.0263 - accuracy: 1.0000
Epoch 9/10
4/4 [==============================] - 0s 6ms/step - loss: 0.0280 - accuracy: 1.0000
Epoch 10/10
4/4 [==============================] - 0s 6ms/step - loss: 0.0252 - accuracy: 1.0000
1/1 [==============================] - 0s 269ms/step - loss: 1.1109 - accuracy: 0.4286
[1.1108570098876953, 0.4285714328289032]
```

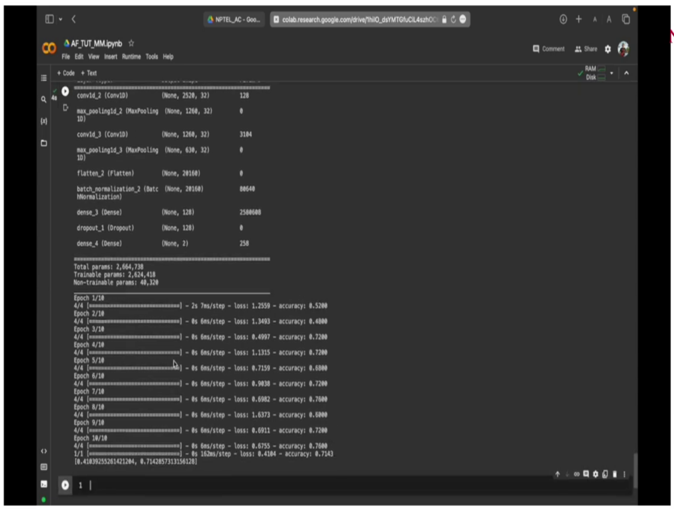
```python
X_train, X_test, y_train, y_test = train_test_split(mfcc_audio_only,label_all, test_size=0.20, random_state =12)
model = CNN((2520, 1))
model.fit(X_train,y_train, batch_size = 8, verbose=1, epochs=10)
model.evaluate(X_test,y_test)
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_2 (Conv1D)           (None, 2520, 32)          128

 max_pooling1d_2 (MaxPooling  (None, 1260, 32)         0
 1D)

 conv1d_3 (Conv1D)           (None, 1260, 32)          3104

 max_pooling1d_3 (MaxPooling  (None, 630, 32)          0
 1D)

 flatten_2 (Flatten)         (None, 20160)             0

 batch_normalization_2 (Batc  (None, 20160)            80640
```

So, we can see over here that our CNN architecture is giving a good training accuracy, but the test accuracy is somewhat around 42 percent which is essentially below chance level and this thing is happening with video features only ok. So, let me try the same thing with audio features and let us see how my CNN architecture is working with audio features.

So, in case of audio features we will again divide our audio data into their respective train and test splits with the test size of 20 percent and we will call our model. We will call a fresh instance of our model and then we will fit it on our training data with a batch size of 8 and number of epoch equal to say 10 and they will try to evaluate it on our test data let us see how it performs ok.

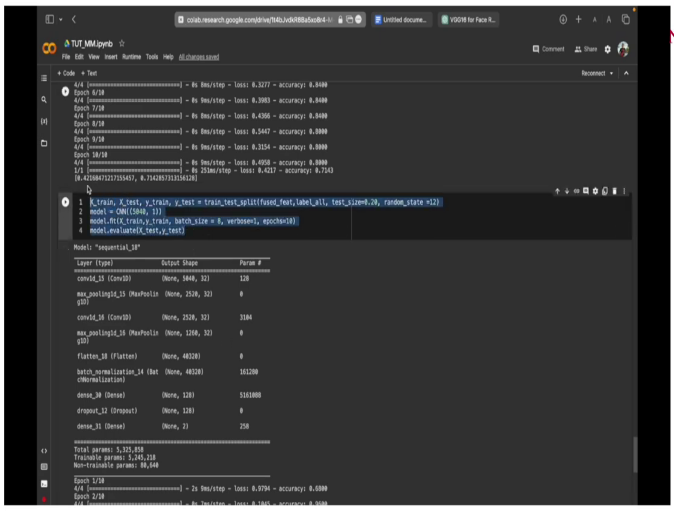
(Refer Slide Time: 33:57)



```
=================================================================
 conv1d_2 (Conv1D)           (None, 2520, 32)          128

 max_pooling1d_2 (MaxPooling  (None, 1260, 32)         0
 1D)

 conv1d_3 (Conv1D)           (None, 1260, 32)          3104

 max_pooling1d_3 (MaxPooling  (None, 630, 32)          0
 1D)

 flatten_2 (Flatten)         (None, 20160)             0

 batch_normalization_2 (Batc  (None, 20160)            80640
 hNormalization)

 dense_3 (Dense)             (None, 128)               2580608

 dropout_1 (Dropout)         (None, 128)               0

 dense_4 (Dense)             (None, 2)                 258

=================================================================
Total params: 2,664,738
Trainable params: 2,624,418
Non-trainable params: 40,320
_________________________________________________________________
Epoch 1/10
4/4 [==============================] - 2s 7ms/step - loss: 1.2559 - accuracy: 0.5200
Epoch 2/10
4/4 [==============================] - 0s 6ms/step - loss: 1.3493 - accuracy: 0.4800
Epoch 3/10
4/4 [==============================] - 0s 6ms/step - loss: 0.4997 - accuracy: 0.7200
Epoch 4/10
4/4 [==============================] - 0s 6ms/step - loss: 1.1315 - accuracy: 0.7200
Epoch 5/10
4/4 [==============================] - 0s 6ms/step - loss: 0.7159 - accuracy: 0.6800
Epoch 6/10
4/4 [==============================] - 0s 6ms/step - loss: 0.9038 - accuracy: 0.7200
Epoch 7/10
4/4 [==============================] - 0s 6ms/step - loss: 0.6982 - accuracy: 0.7600
Epoch 8/10
4/4 [==============================] - 0s 6ms/step - loss: 1.6373 - accuracy: 0.6000
Epoch 9/10
4/4 [==============================] - 0s 6ms/step - loss: 0.6911 - accuracy: 0.7200
Epoch 10/10
4/4 [==============================] - 0s 6ms/step - loss: 0.6755 - accuracy: 0.7600
1/1 [==============================] - 0s 162ms/step - loss: 0.4104 - accuracy: 0.7143
[0.41039255261421204, 0.7142857313156128]
```

So, as we can see that using these 1 D CNN are audio features are giving better test accuracy as compared to our video features. Now, we are interested to see how the fusion of these two feature will perform in 1 D CNN architecture. So, for that I will simply use the fused version of the feature and the code will look something like this.

(Refer Slide Time: 34:27)

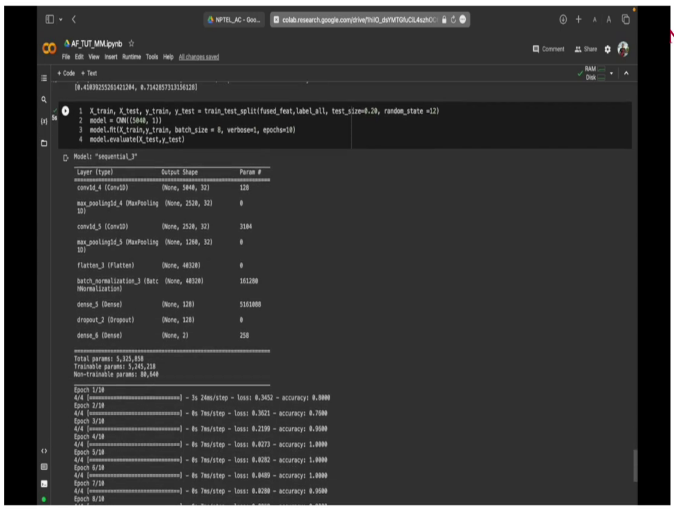


```
4/4 [==============================] - 0s 8ms/step - loss: 0.3277 - accuracy: 0.8400
Epoch 6/10
4/4 [==============================] - 0s 9ms/step - loss: 0.3983 - accuracy: 0.8400
Epoch 7/10
4/4 [==============================] - 0s 8ms/step - loss: 0.4366 - accuracy: 0.8400
Epoch 8/10
4/4 [==============================] - 0s 8ms/step - loss: 0.5447 - accuracy: 0.8000
Epoch 9/10
4/4 [==============================] - 0s 9ms/step - loss: 0.3154 - accuracy: 0.8000
Epoch 10/10
4/4 [==============================] - 0s 9ms/step - loss: 0.4958 - accuracy: 0.8000
1/1 [==============================] - 0s 251ms/step - loss: 0.4217 - accuracy: 0.7143
[0.42168471217155457, 0.7142857313156128]
```

```
X_train, X_test, y_train, y_test = train_test_split(fused_feat,label_all, test_size=0.20, random_state =12)
model = CNN((5040, 1))
model.fit(X_train,y_train, batch_size = 8, verbose=1, epochs=10)
model.evaluate(X_test,y_test)
```

```
Model: "sequential_18"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_15 (Conv1D)          (None, 5040, 32)          128

 max_pooling1d_15 (MaxPoolin  (None, 2520, 32)         0
 g1D)

 conv1d_16 (Conv1D)          (None, 2520, 32)          3104

 max_pooling1d_16 (MaxPoolin  (None, 1260, 32)         0
 g1D)

 flatten_18 (Flatten)        (None, 40320)             0

 batch_normalization_14 (Bat  (None, 40320)            161280
 chNormalization)

 dense_30 (Dense)            (None, 128)               5161088

 dropout_12 (Dropout)        (None, 128)               0

 dense_31 (Dense)            (None, 2)                 258

=================================================================
Total params: 5,325,858
Trainable params: 5,245,218
Non-trainable params: 80,640
_________________________________________________________________
Epoch 1/10
4/4 [==============================] - 2s 9ms/step - loss: 0.9794 - accuracy: 0.6800
Epoch 2/10
4/4 [==============================] - 0s 7ms/step - loss: 0.1045 - accuracy: 0.9600
```

(Refer Slide Time: 34:34)



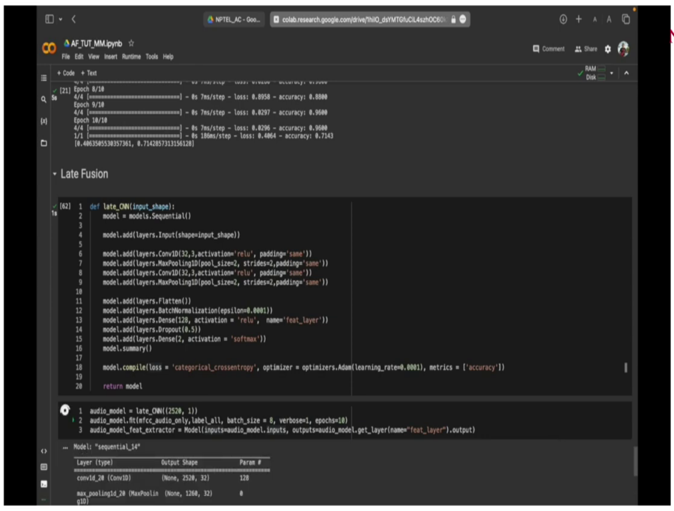
```
[0.4103925526142I204, 0.7142857313156128]
```

```
X_train, X_test, y_train, y_test = train_test_split(fused_feat,label_all, test_size=0.20, random_state =12)
model = CNN((5040, 1))
model.fit(X_train,y_train, batch_size = 8, verbose=1, epochs=10)
model.evaluate(X_test,y_test)
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_4 (Conv1D)           (None, 5040, 32)          128

 max_pooling1d_4 (MaxPooling  (None, 2520, 32)         0
 1D)

 conv1d_5 (Conv1D)           (None, 2520, 32)          3104

 max_pooling1d_5 (MaxPooling  (None, 1260, 32)         0
 1D)

 flatten_3 (Flatten)         (None, 40320)             0

 batch_normalization_3 (Batc  (None, 40320)            161280
 hNormalization)

 dense_5 (Dense)             (None, 128)               5161088

 dropout_2 (Dropout)         (None, 128)               0

 dense_6 (Dense)             (None, 2)                 258

=================================================================
Total params: 5,325,858
Trainable params: 5,245,218
Non-trainable params: 80,640
_________________________________________________________________
Epoch 1/10
4/4 [==============================] - 3s 24ms/step - loss: 0.3452 - accuracy: 0.8000
Epoch 2/10
4/4 [==============================] - 0s 7ms/step - loss: 0.3621 - accuracy: 0.7600
Epoch 3/10
4/4 [==============================] - 0s 7ms/step - loss: 0.2199 - accuracy: 0.9600
Epoch 4/10
4/4 [==============================] - 0s 7ms/step - loss: 0.0273 - accuracy: 1.0000
Epoch 5/10
4/4 [==============================] - 0s 7ms/step - loss: 0.0282 - accuracy: 1.0000
Epoch 6/10
4/4 [==============================] - 0s 7ms/step - loss: 0.0489 - accuracy: 1.0000
Epoch 7/10
4/4 [==============================] - 0s 7ms/step - loss: 0.0280 - accuracy: 0.9600
Epoch 8/10
```

Here I am using this fuse feature with the set size 20 percent and dividing into my respective filter sets calling a fresh instance of our CNN architecture and fitting our training data with again same batch size 8 and epoch equal to 10 ok.

(Refer Slide Time: 35:03)



```
Epoch 8/10
4/4 [==============================] - 0s 7ms/step - loss: 0.8958 - accuracy: 0.8800
Epoch 9/10
4/4 [==============================] - 0s 7ms/step - loss: 0.0297 - accuracy: 0.9600
Epoch 10/10
4/4 [==============================] - 0s 7ms/step - loss: 0.0296 - accuracy: 0.9600
1/1 [==============================] - 0s 186ms/step - loss: 0.4064 - accuracy: 0.7143
[0.4063505530357361, 0.7142857313156128]
```

Late Fusion

```
def late_CNN(input_shape):
    model = models.Sequential()

    model.add(layers.Input(shape=input_shape))

    model.add(layers.Conv1D(32,3,activation='relu', padding='same'))
    model.add(layers.MaxPooling1D(pool_size=2, strides=2,padding='same'))
    model.add(layers.Conv1D(32,3,activation='relu', padding='same'))
    model.add(layers.MaxPooling1D(pool_size=2, strides=2,padding='same'))

    model.add(layers.Flatten())
    model.add(layers.BatchNormalization(epsilon=0.0001))
    model.add(layers.Dense(128, activation = 'relu',  name='feat_layer'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(2, activation = 'softmax'))
    model.summary()

    model.compile(loss = 'categorical_crossentropy', optimizer = optimizers.Adam(learning_rate=0.0001), metrics = ['accuracy'])

    return model
```

```
audio_model = late_CNN((2520, 1))
audio_model.fit(mfcc_audio_only,label_all, batch_size = 8, verbose=1, epochs=10)
audio_model_feat_extractor = Model(inputs=audio_model.inputs, outputs=audio_model.get_layer(name="feat_layer").output)
```

```
Model: "sequential_14"
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_20 (Conv1D)          (None, 2520, 32)          128

 max_pooling1d_20 (MaxPoolin  (None, 1260, 32)         0
 g1D)
```

And in this case, we can see that our model is able to learn at least 71 percent which is somewhat equivalent to the audio level features. Here we can see like fusing feature I am adding we can get at least the accuracy single test performing feature. And again, here though we have not done any sort of hyper parameter tuning and the data is also very less over here. So, I mean there is still a chance that we can get a better accuracy over here and this is a classic example of our early feature fusion.

So, what is typically happening here is we took two features concatenated it together and passed it to our network. So, after performing early fusion we will now perform late feature fusion where late feature fusion refers to a method where feature extracted from multiple sources such as different sensors or modalities are combined at a later stage in the processing pipeline.

After initial processing of these individual sources, this allows for more flexible and efficient feature representation as it enables the combinations of diverse and complementary information. In late fusion technique the feature are extracted from different sources are first processed independently often using different techniques or architectures and then combined in the later stage of the pipeline such as a fully connected layer.

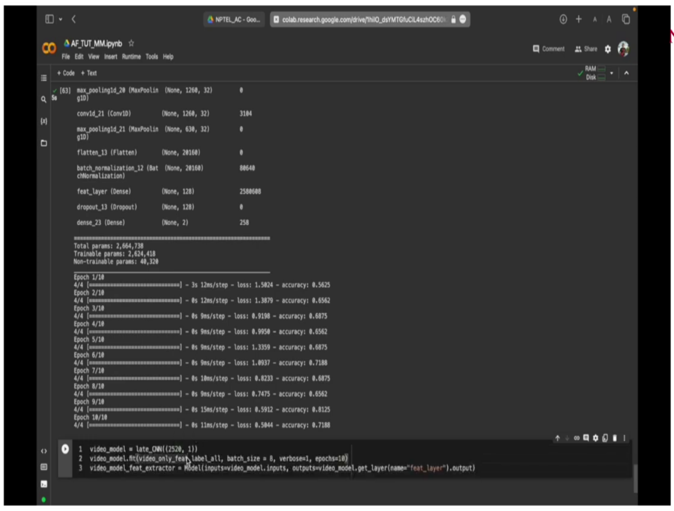
It can be also be done using various techniques such as concatenation of two features coming from different modalities maybe their element wise additions or element wise multiplication or maybe some sort of a weighted averaging of two features. So, in our case we will simply take our video features and audio features.

And pass them to a different CNN architectures and from the feature representation layer of their respective CNN architecture, we will collect the late features and then we will manually concatenate them to a late feature embedding and using that embedding we will obtain a multi-layer perceptron and see the efficacy of our late fusion technique.

So, to do so, we will first create a model and this will be called as late-CNN model and the agenda of this model will be to extract relevant features from this feature representation layer which essentially consist of 128 neurons and the activation function is relu over here. So, running this code. So, after that we can simply now call our late CNN layer as a audio model and fit it on our mfcc audio only features with a batch size of 8 and the number of epochs will be 10.

After we fit the model, we will again make our feature extractor and extract the features from this feature representation layer and the code will look something like this.

(Refer Slide Time: 39:03)

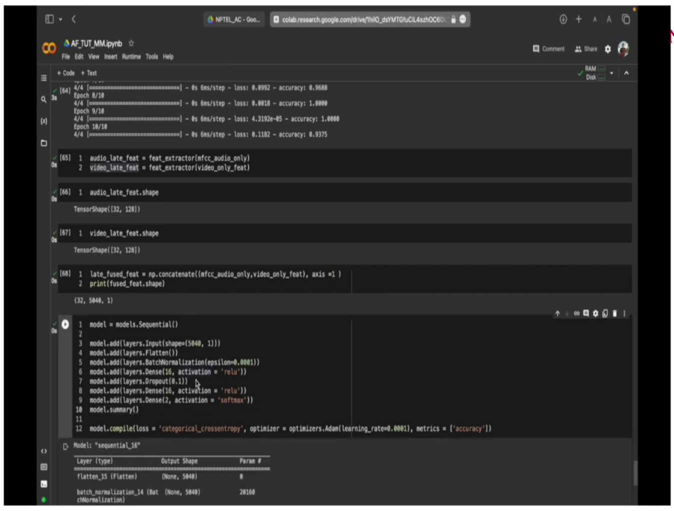


```
max_pooling1d_20 (MaxPoolin  (None, 1260, 32)        0
g1D)

conv1d_21 (Conv1D)          (None, 1260, 32)        3104

max_pooling1d_21 (MaxPoolin  (None, 630, 32)         0
g1D)

flatten_13 (Flatten)        (None, 20160)           0

batch_normalization_12 (Bat  (None, 20160)          80640
chNormalization)

feat_layer (Dense)          (None, 128)             2580608

dropout_13 (Dropout)        (None, 128)             0

dense_23 (Dense)            (None, 2)               258

=================================================================
Total params: 2,664,738
Trainable params: 2,624,418
Non-trainable params: 40,320
_________________________________________________________________
Epoch 1/10
4/4 [==============================] - 3s 12ms/step - loss: 1.5024 - accuracy: 0.5625
Epoch 2/10
4/4 [==============================] - 0s 12ms/step - loss: 1.3879 - accuracy: 0.6562
Epoch 3/10
4/4 [==============================] - 0s 9ms/step - loss: 0.9198 - accuracy: 0.6875
Epoch 4/10
4/4 [==============================] - 0s 9ms/step - loss: 0.9950 - accuracy: 0.6562
Epoch 5/10
4/4 [==============================] - 0s 9ms/step - loss: 1.3359 - accuracy: 0.6875
Epoch 6/10
4/4 [==============================] - 0s 9ms/step - loss: 1.0937 - accuracy: 0.7188
Epoch 7/10
4/4 [==============================] - 0s 10ms/step - loss: 0.8233 - accuracy: 0.6875
Epoch 8/10
4/4 [==============================] - 0s 9ms/step - loss: 0.7475 - accuracy: 0.6562
Epoch 9/10
4/4 [==============================] - 0s 15ms/step - loss: 0.5912 - accuracy: 0.8125
Epoch 10/10
4/4 [==============================] - 0s 11ms/step - loss: 0.5044 - accuracy: 0.7188
```

```python
video_model = late_CNN((2520, 1))
video_model.fit(video_only_feat, label_all, batch_size = 8, verbose=1, epochs=10)
video_model_feat_extractor = Model(inputs=video_model.inputs, outputs=video_model.get_layer(name="feat_layer").output)
```

Now, same thing we will perform with our video features.

(Refer Slide Time: 39:17)



```
4/4 [==============================] - 0s 6ms/step - loss: 0.0992 - accuracy: 0.9688
Epoch 8/10
4/4 [==============================] - 0s 6ms/step - loss: 0.0018 - accuracy: 1.0000
Epoch 9/10
4/4 [==============================] - 0s 6ms/step - loss: 4.3192e-05 - accuracy: 1.0000
Epoch 10/10
4/4 [==============================] - 0s 6ms/step - loss: 0.1182 - accuracy: 0.9375
```

```python
audio_late_feat = feat_extractor(mfcc_audio_only)
video_late_feat = feat_extractor(video_only_feat)
```

```python
audio_late_feat.shape
```

```
TensorShape([32, 128])
```

```python
video_late_feat.shape
```

```
TensorShape([32, 128])
```

```python
late_fused_feat = np.concatenate((mfcc_audio_only,video_only_feat), axis =1 )
print(fused_feat.shape)
```

```
(32, 5040, 1)
```

```python
model = models.Sequential()

model.add(layers.Input(shape=(5040, 1)))
model.add(layers.Flatten())
model.add(layers.BatchNormalization(epsilon=0.0001))
model.add(layers.Dense(16, activation = 'relu'))
model.add(layers.Dropout(0.1))
model.add(layers.Dense(16, activation = 'relu'))
model.add(layers.Dense(2, activation = 'softmax'))
model.summary()

model.compile(loss = 'categorical_crossentropy', optimizer = optimizers.Adam(learning_rate=0.0001), metrics = ['accuracy'])
```

```
Model: "sequential_16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_15 (Flatten)        (None, 5040)              0

 batch_normalization_14 (Bat  (None, 5040)             20160
 chNormalization)
```

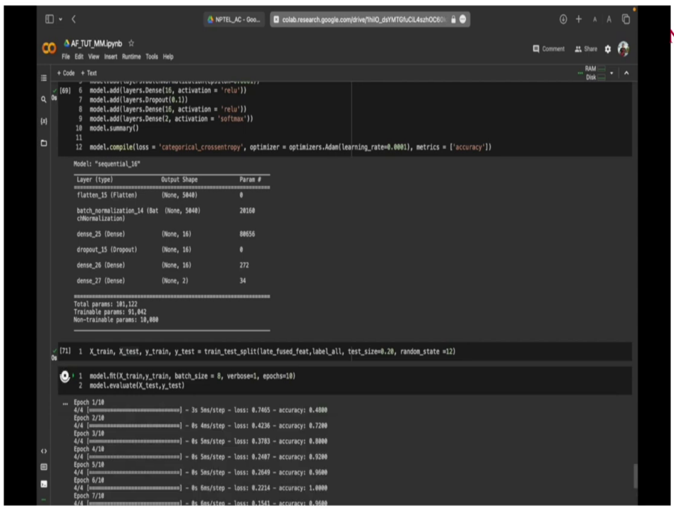
And then now since we have our audio and video model train, we will simply extract their feature layer using this code and save them into a separate variables. So, if I show the shape of this variable, it will it will be a tensor of 32 cross 128. So, basically 32 are number of our sample and 128 is our limited representation of that size of that latent representation. Similarly, for the video we have the similar shape.

Now, we can simply concatenate these two features these two late representation and make a late fused latent representation. For that our code will look something like this. Now, we have this late representation we will create an another multi-layer perceptron and we will train that perceptron on our training data which essentially will be coming from these late fused features and test on the respective test data and we will see how this late fusion technique will work.

So, our model will look something like this it is a very simple model where we will simply flatten down the these features and then we perform a batch normalization and there will be two tens layer consisting of 16 neurons and relu activation function and I will be putting a drop-down layer of with 10 probability to regularize this network and later we will classify the emotion classes using softmax classifier.

And I will again training this model with categorical cross entropy loss function with adam optimizer the at the learning rate of 0.0001 percent.

(Refer Slide Time: 41:47)



```
model.add(layers.Dense(16, activation = 'relu'))
model.add(layers.Dropout(0.1))
model.add(layers.Dense(16, activation = 'relu'))
model.add(layers.Dense(2, activation = 'softmax'))
model.summary()

model.compile(loss = 'categorical_crossentropy', optimizer = optimizers.Adam(learning_rate=0.0001), metrics = ['accuracy'])
```

```
Model: "sequential_16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_15 (Flatten)        (None, 5040)              0

 batch_normalization_14 (Bat  (None, 5040)             20160
 chNormalization)

 dense_25 (Dense)            (None, 16)                80656

 dropout_15 (Dropout)        (None, 16)                0

 dense_26 (Dense)            (None, 16)                272

 dense_27 (Dense)            (None, 2)                 34

=================================================================
Total params: 101,122
Trainable params: 91,042
Non-trainable params: 10,080
_________________________________________________________________
```

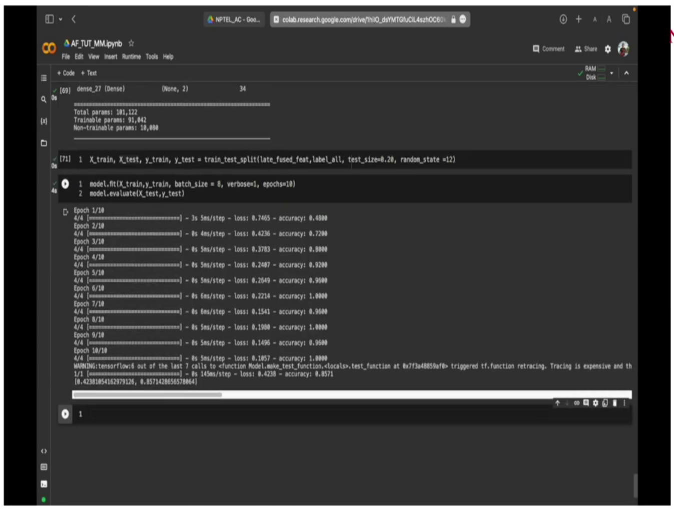
```
X_train, X_test, y_train, y_test = train_test_split(late_fused_feat,label_all, test_size=0.20, random_state =12)
```

```
model.fit(X_train,y_train, batch_size = 8, verbose=1, epochs=10)
model.evaluate(X_test,y_test)
```

```
Epoch 1/10
4/4 [==============================] - 3s 5ms/step - loss: 0.7465 - accuracy: 0.4800
Epoch 2/10
4/4 [==============================] - 0s 4ms/step - loss: 0.4236 - accuracy: 0.7200
Epoch 3/10
4/4 [==============================] - 0s 5ms/step - loss: 0.3783 - accuracy: 0.8000
Epoch 4/10
4/4 [==============================] - 0s 5ms/step - loss: 0.2407 - accuracy: 0.9200
Epoch 5/10
4/4 [==============================] - 0s 5ms/step - loss: 0.2649 - accuracy: 0.9600
Epoch 6/10
4/4 [==============================] - 0s 6ms/step - loss: 0.2214 - accuracy: 1.0000
Epoch 7/10
4/4 [==============================] - 0s 6ms/step - loss: 0.1541 - accuracy: 0.9600
```

Now, a model is compiled I can simply divide my late fused data into train and test splits and the code will look something like this and our test size is again 20 percent and now I will simply run my model and evaluate it on test set let us see how does it perform now ok.

(Refer Slide Time: 42:19)



```
dense_27 (Dense)          (None, 2)              34

=================================================================
Total params: 101,122
Trainable params: 91,042
Non-trainable params: 10,080
_________________________________________________________________
```

```
X_train, X_test, y_train, y_test = train_test_split(late_fused_feat,label_all, test_size=0.20, random_state =12)
```

```
model.fit(X_train,y_train, batch_size = 8, verbose=1, epochs=10)
model.evaluate(X_test,y_test)
```

```
Epoch 1/10
4/4 [==============================] - 3s 5ms/step - loss: 0.7465 - accuracy: 0.4800
Epoch 2/10
4/4 [==============================] - 0s 4ms/step - loss: 0.4236 - accuracy: 0.7200
Epoch 3/10
4/4 [==============================] - 0s 5ms/step - loss: 0.3783 - accuracy: 0.8000
Epoch 4/10
4/4 [==============================] - 0s 5ms/step - loss: 0.2407 - accuracy: 0.9200
Epoch 5/10
4/4 [==============================] - 0s 5ms/step - loss: 0.2649 - accuracy: 0.9600
Epoch 6/10
4/4 [==============================] - 0s 6ms/step - loss: 0.2214 - accuracy: 1.0000
Epoch 7/10
4/4 [==============================] - 0s 6ms/step - loss: 0.1541 - accuracy: 0.9600
Epoch 8/10
4/4 [==============================] - 0s 5ms/step - loss: 0.1980 - accuracy: 1.0000
Epoch 9/10
4/4 [==============================] - 0s 5ms/step - loss: 0.1496 - accuracy: 0.9600
Epoch 10/10
4/4 [==============================] - 0s 5ms/step - loss: 0.1057 - accuracy: 1.0000
WARNING:tensorflow:6 out of the last 7 calls to <function Model.make_test_function.<locals>.test_function at 0x7f3a48859af0> triggered tf.function retracing. Tracing is expensive and th
1/1 [==============================] - 0s 145ms/step - loss: 0.4238 - accuracy: 0.8571
[0.42381054162979126, 0.8571428656578064]
```

So, yeah as we can see here we are getting better accuracy in this late fusion case as compared to our early fusion. In late fusion we are getting test accuracy of 85 percent whereas, in our early fusion case our test accuracies was 71 percent. So, in line with the literature we were having two different sort of modalities we extracted relevant features of these two modalities using some sort of a machine learning algorithm in our case a convolution neural network.

And later we extracted the latent representation from these machine learning algorithm from the CNN and combined those latent representation and did of late fusion and it is giving a better accuracy than our earlier techniques. So, finally, concluding this tutorial in this tutorial we started with we started with working with audio and video information.

We extracted mfcc features from audio modality and we used a pre-trained VGG16 phase architecture to get features from our video data later using these features we. Firstly, separately trained our machine learning models and saw how they are performing then we try to fuse the given embeddings and saw how these classified again performed later we did a CNN based approach where we first tested early feature fusion method and then we tested later feature fusion method.

And we saw that in case of data where there are multiple modalities coming these late fusion techniques works better.

Thank you.