



Applied Accelerated Artificial Intelligence
Prof. Satyadhyan Chickerur
Department of Computer Science and Engineering
Indian Institute of Technology, Palakkad

Lecture - 55
Applied AI: Smart City (Intelligent Video Analytics)
Session 1 - Part 2




(Refer Slide Time: 00:19)



Demo 2 : Video Classification :

[Example : 1 : Training and Classification and Inferencing](#)

[Example : 2 : Multi class video Classification using Pretrained Model](#)



Video classification example, 1 minute, yeah. So, this we have already there, so yeah.

(Refer Slide Time: 00:34)

video_classification

Table of contents

- Video Classification with a CNN-RNN Architecture
- Data collection
- Setup
- Define hyperparameters
- Data preparation
- The sequence model
- Inference
- Next steps
- Section

Video Classification with a CNN-RNN Architecture

Author: [Sayan Paul](#)
 Date created: 2021/05/28
 Last modified: 2021/06/05
 Description: Training a video classifier with transfer learning and a recurrent model on the UCF101 dataset.

This example demonstrates **video classification**, an important use-case with applications in recommendations, security, and so on. We will be using the **UCF101 dataset** to build our video classifier. The dataset consists of videos categorized into different actions, like cricket shot, punching, biking, etc. This dataset is commonly used to build action recognizers, which are an application of video classification.

A video consists of an ordered sequence of frames. Each frame contains spatial information, and the sequence of those frames contains temporal information. To model both of these aspects, we use a hybrid architecture that consists of convolutions (for spatial processing) as well as recurrent layers (for temporal processing). Specifically, we'll use a Convolutional Neural Network (CNN) and a Recurrent Neural Network (RNN) consisting of **GRU layers**. This kind of hybrid architecture is popularly known as a **CNN-RNN**.

This example requires TensorFlow 2.5 or higher, as well as TensorFlow Docs, which can be installed using the following command:

```
[1] !pip install -q git+https://github.com/tensorflow/docs
```

Building wheel for tensorflow-docs (setup.py) ... done

Data collection

In order to keep the runtime of this example relatively short, we will be using a subsampled version of the original UCF101 dataset. You

So, this example basically talks off a CNN-RNN architecture and this basically is trying to train a video classifier with transfer learning and it uses a recurrent neural network model and it uses UCF101 dataset. So, this demonstration actually gives us the video classification it is an important use case and this particular UCF101 dataset consists of different actions right it has got cricket shot, punching, biking ok.

And this particular data set is generally used to build action recognizers right which are an application of video classification. So, this is the first step wherein you are trying to set up your environment right.

(Refer Slide Time: 01:35)

Setup

```
from tensorflow_docs.vis import embed
from tensorflow import keras
from IPython.display import Image

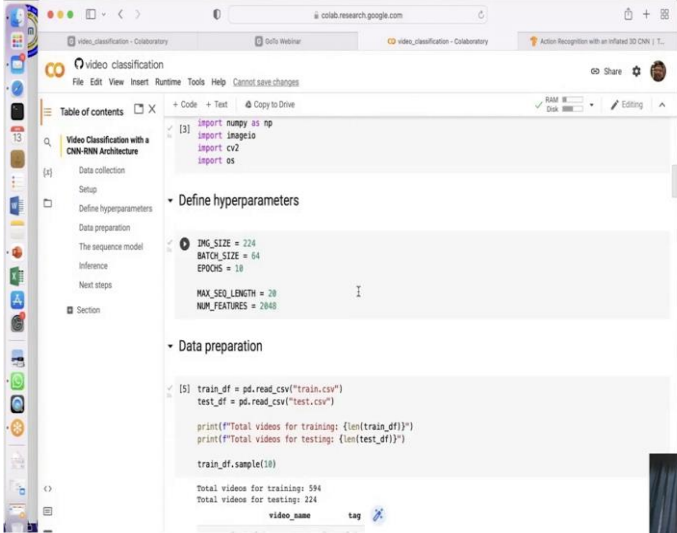
import matplotlib.pyplot as plt
import tensorflow as tf
import pandas as pd
import numpy as np
import imageio
import cv2
import os
```

Define hyperparameters

```
IMG_SIZE = 224
BATCH_SIZE = 64
EPOCHS = 10
MAX_SEQ_LENGTH = 20
NUM_FEATURES = 2048
```

And then this is where you collect in the data right or the data set UCF101 then you basically set up your environment wherein you are using CV2 pandas, matplotlib, tensor flow ok keras and all of that.

(Refer Slide Time: 01:54)



```
import numpy as np
import imageio
import cv2
import os
```

Define hyperparameters

```
IMG_SIZE = 224
BATCH_SIZE = 64
EPOCHS = 18
MAX_SEQ_LENGTH = 28
NUM_FEATURES = 2848
```

Data preparation

```
train_df = pd.read_csv("train.csv")
test_df = pd.read_csv("test.csv")

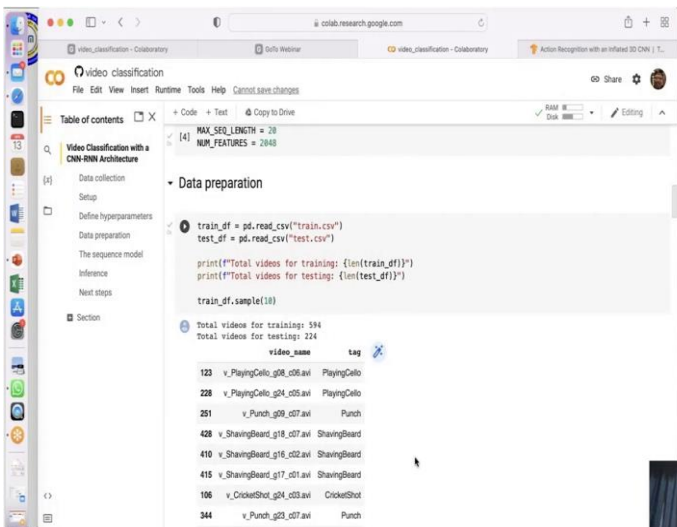
print(f"Total videos for training: {len(train_df)}")
print(f"Total videos for testing: {len(test_df)}")

train_df.sample(100)
```

video_name	tag
v_PlayingCello_g08_c06.avi	PlayingCello
v_PlayingCello_g04_c05.avi	PlayingCello
v_Punch_g09_c07.avi	Punch
v_ShavingBeard_g18_c07.avi	ShavingBeard
v_ShavingBeard_g16_c02.avi	ShavingBeard
v_ShavingBeard_g17_c01.avi	ShavingBeard
v_CricketShot_g04_c03.avi	CricketShot
v_Punch_g23_c07.avi	Punch
v_PlayingCello_c15_c02.avi	PlayingCello

Then you define the hyper parameters what is the image size, what is the batch size, what are what is the epoch right and what is the maximum sequence length and number of features which you are trying to do.

(Refer Slide Time: 02:08)



```
train_df = pd.read_csv("train.csv")
test_df = pd.read_csv("test.csv")

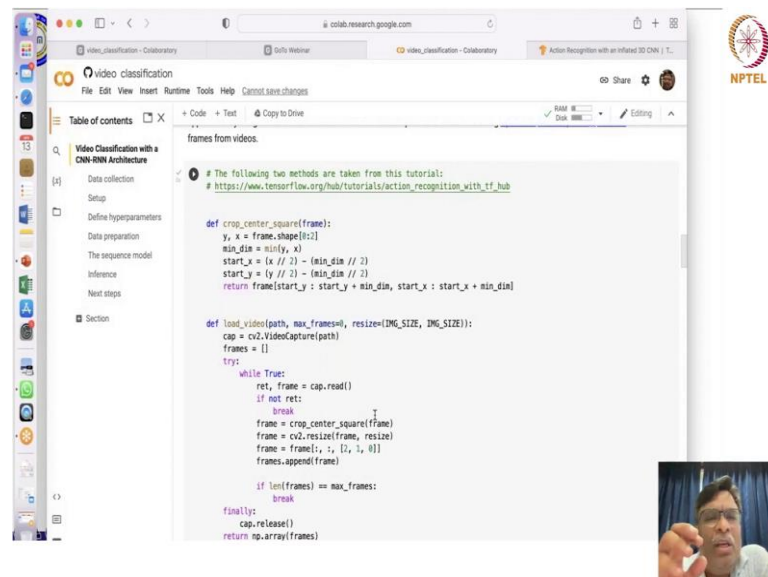
print(f"Total videos for training: {len(train_df)}")
print(f"Total videos for testing: {len(test_df)}")

train_df.sample(100)
```

video_name	tag
v_PlayingCello_g08_c06.avi	PlayingCello
v_PlayingCello_g04_c05.avi	PlayingCello
v_Punch_g09_c07.avi	Punch
v_ShavingBeard_g18_c07.avi	ShavingBeard
v_ShavingBeard_g16_c02.avi	ShavingBeard
v_ShavingBeard_g17_c01.avi	ShavingBeard
v_CricketShot_g04_c03.avi	CricketShot
v_Punch_g23_c07.avi	Punch
v_PlayingCello_c15_c02.avi	PlayingCello

Then you have this data preparation wherein how many videos are being used for training, how many videos are being used for testing right. So, you are trying to actually use this for training purpose and testing purpose and then you basically are trying to do ok.

(Refer Slide Time: 02:29)

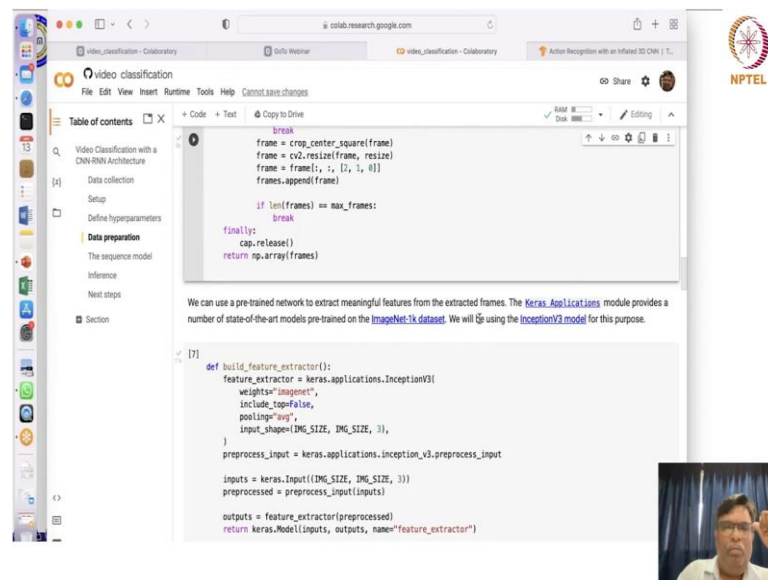


Action recognition in a sense you are trying to actually do action recognition with tf hub. So, this basic idea is that you are trying to generate a sequence model right. So, you are trying to load in the video ok you are trying to load in the video what is the frame we are trying to use ok.

And then we are going to actually crop the frame right for getting the center and then trying to understand the region of interest and then once you are able to extract right all these frames. So, you are trying to actually do to get the region of interest ok which basically gives you the action recognition.

So; obviously, in a frame in a static frame there will be some region of interest right and that region of interest you need to actually crop ok.

(Refer Slide Time: 03:48)

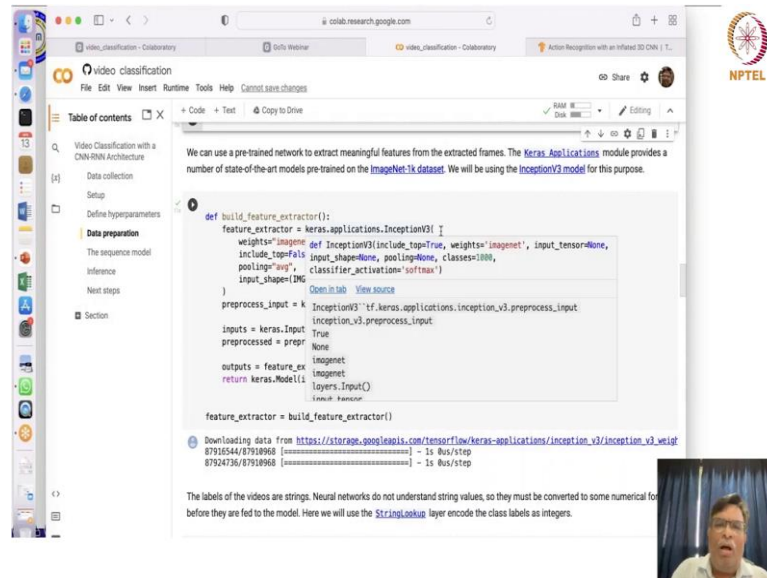


And then each of these frames ok could be or should be actually used ok buy a pretrained network ok to extract certain meaningful features right from these frames. So, now what effectively happens is, you have found out the region of interest in the frames and out of these region of interest or these portions of your frames you need to actually extract meaningful information right. So, effectively you are trying to develop a feature extractor ok.

Now, when you talk of a feature extractor, the idea is that you are trying to actually get those features which are more important right to actually generalize for various videos which you are going to get input for or you are going to get those videos as input right. Otherwise, a question basically is or might be asked that when you are trying to do a classification and you are trying to use CNN and RNN architecture.

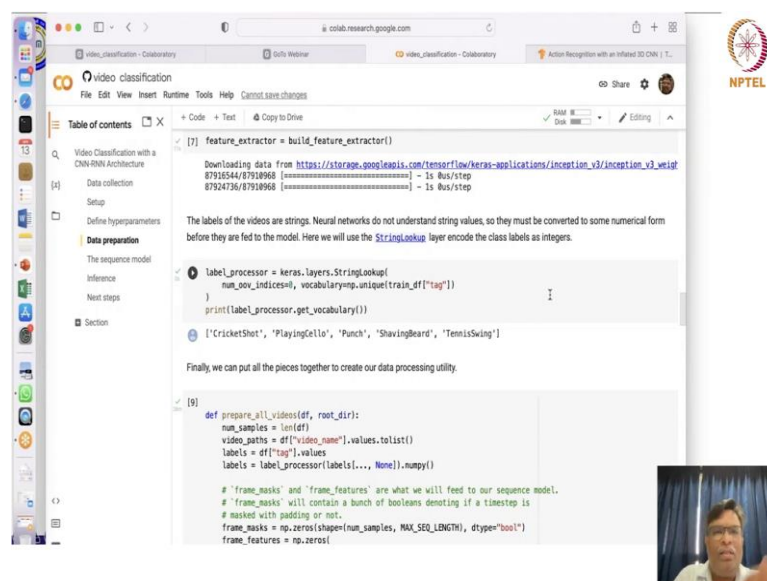
Why is that you are going to again extract features when CNN and RNN is actually capable of doing it on its own? Ok. So, the idea here is that the features are extracted or features might be extracted, but which are the specific features of our own interest ok is something which we need to actually boil down to from lot of features which this type of a network would have given us right.

(Refer Slide Time: 05:52)



So, we are trying to develop a feature extractor here right ok and here we are using inception v3 right for that purpose. So, now if you want to see the source and all, all of this is available in this.

(Refer Slide Time: 06:11)



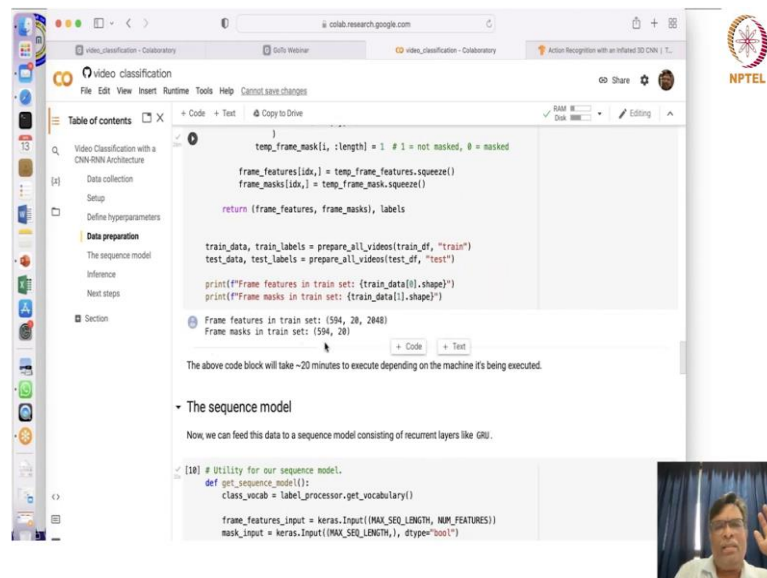
Now you have extracted the features or you have built the feature extractor now the labels for each of these videos are strings and when you say a neural network or for that matter a deep neural network it is not going to understand the string values. So, you are supposed to convert it into some form ok which is basically to be used by a neural network model.

So, you are going to convert it into some numerical form ok and then there is something which wherein we use string loop lookup ok which basically is a layer which encodes the class labels as integers. So, there has to be an encoding done for each of these classes as integers right. So, this is what is done here ok. So, if you see the label processor it is going to give us the labels like cricket shot, playing cello punch, shaving beard, tennis swing.

So, these are from that particular data set right. So, we are talking of video classification right. So, now, once you actually see all this. So, what have we done till now? We have actually done these steps wherein we have defined our hyper parameters, we have prepared in the data ok which needs to be used ok and then we have tried to find out ok which are the actions which we need to actually do things and then we have to basically build a feature extractor ok.

And then we can basically link each of these class of labels to each of these videos right and then you are going to prepare all the videos for each of these videos you are going to actually do all of this ok.

(Refer Slide Time: 08:33)



```
temp_frame_mask[i, :length] = 1 # 1 = not masked, 0 = masked
frame_features[idx,] = temp_frame_features.squeeze()
frame_masks[idx,] = temp_frame_mask.squeeze()
return (frame_features, frame_masks), labels

train_data, train_labels = prepare_all_videos(train_df, "train")
test_data, test_labels = prepare_all_videos(test_df, "test")
print(f"Frame features in train set: {train_data[0].shape}")
print(f"Frame masks in train set: {train_data[1].shape}")
Frame features in train set: (594, 20, 2848)
Frame masks in train set: (594, 20)

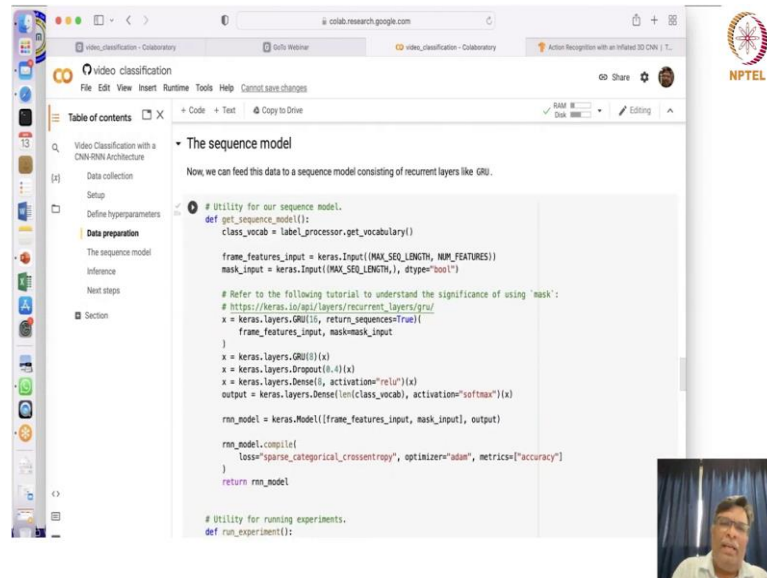
The above code block will take ~20 minutes to execute depending on the machine it's being executed.

The sequence model
Now, we can feed this data to a sequence model consisting of recurrent layers like GRU.

[18]: # Utility for our sequence model.
def get_sequence_model():
    class_vocab = label_processor.get_vocabulary()
    frame_features_input = keras.Input((MAX_SEQ_LENGTH, NUM_FEATURES))
    mask_input = keras.Input((MAX_SEQ_LENGTH,), dtype="bool")
```

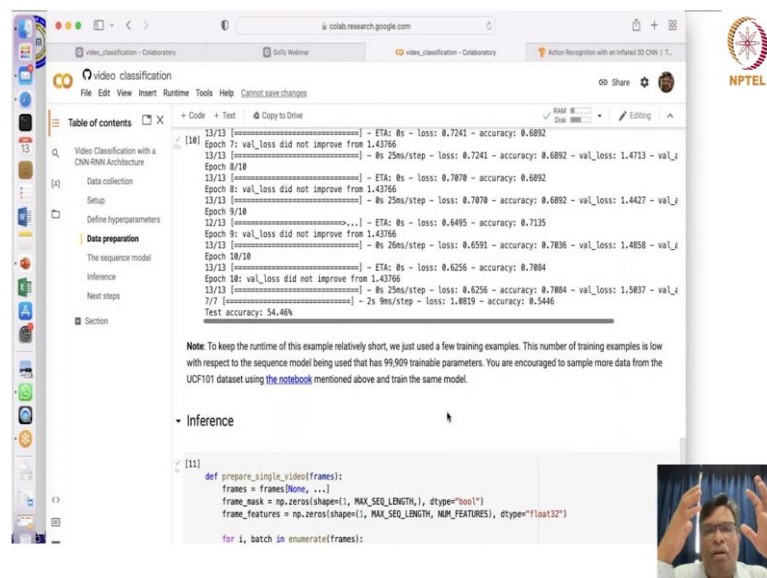
And then if you see the frame features in the training set is this and frame masks in the training set is this, now this we have already executed because it will take time right.

(Refer Slide Time: 08:48)



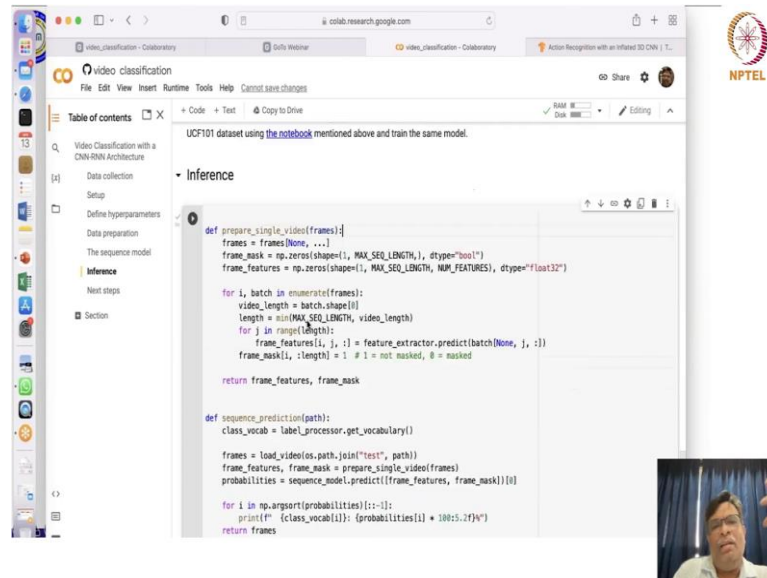
And then the sequence model is basically to feed this data to a sequence model ok of recurrent layers ok. So, what basically is being done here is, you are going to understand right why these mask models are required right. And then you basically try to run these experiments, train it ok for each of these epochs which we have already done.

(Refer Slide Time: 09:16)



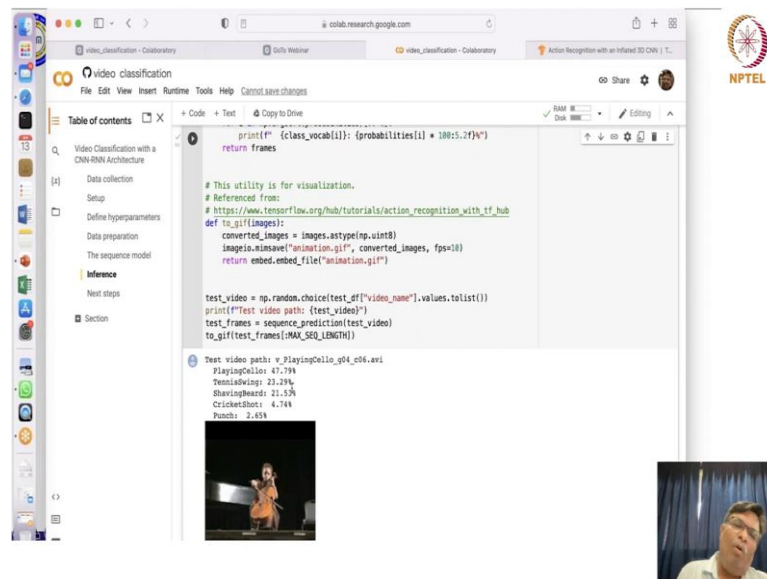
And then once this model is trained ok now this particular thing is basically is something like a sequence model with so, many trainable parameters right. And then using this UCF101 data set and then you basically can use this for inferencing ok.

(Refer Slide Time: 09:46)



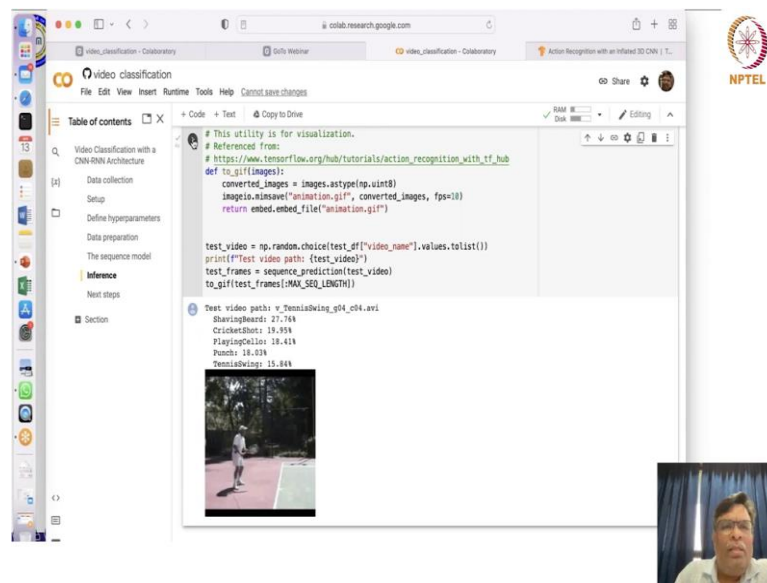
So, the idea is this that, if you run this particular thing ok it will randomly take certain images and then do the prediction.

(Refer Slide Time: 09:53)



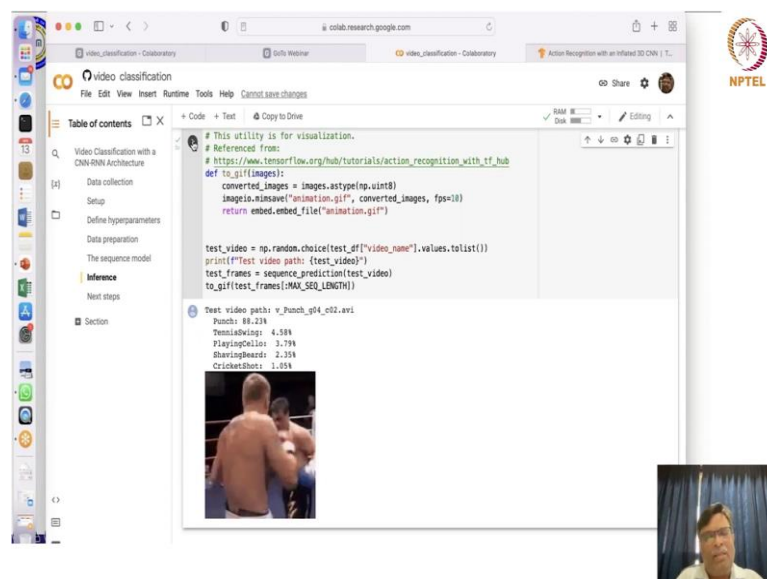
For example, this is the video path what is it doing? It is classifying it as playing a Cello 47.79 percent it is not surely related to tennis swing, but it is basically trying to give it that particular percentage right. And then you have to get shot and punch with respective values the basic idea is, you need to train it more right we trained it with this data set with certain specific number of epochs just to show you like how fast we can do this ok. Otherwise, you can do it for various settings of hyper parameters, then try to do it we will try to see again it will randomly choose certain videos and see here.

(Refer Slide Time: 10:40)



This is trying to detect that it is 27.76 percent as ShavingBeard and TennisSwing it gives me as 15.84 percent. So, this effectively is a wrong classification right. So, it is not necessary that every classification should be correct because of your training parameters, training values training epochs and all of that you understand that right. So, that is how it is.

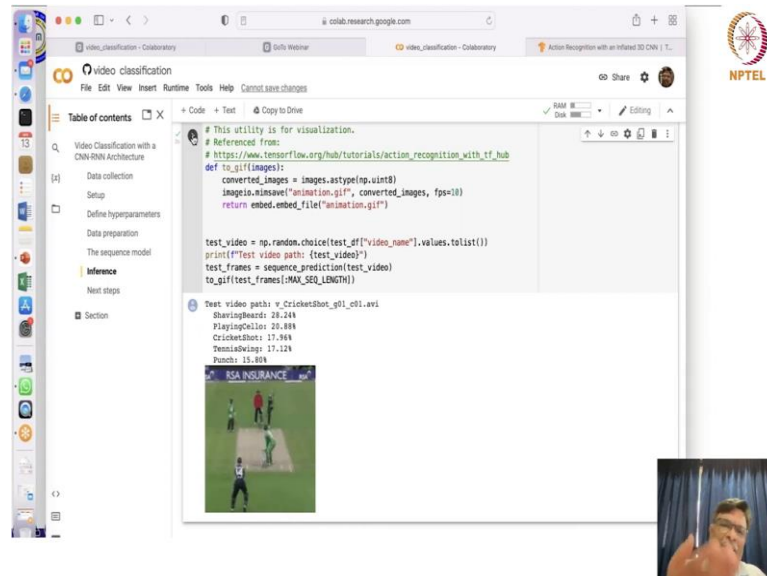
(Refer Slide Time: 11:20)



So, let us try to see one or two more videos and then we will see ok of what to be yes. So, this is randomly taking pictures ok which are not used for training and then this is

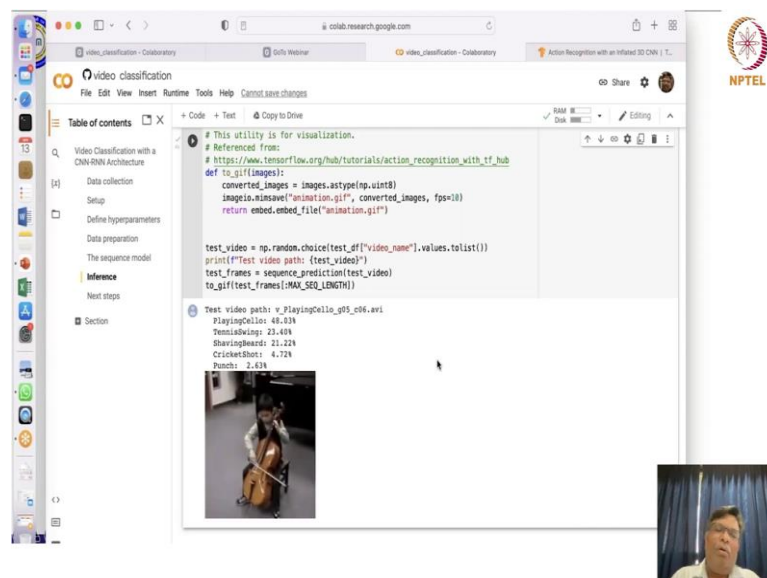
how it is going to do this right. So, what effectively is happening here is, these video classifications are not correct ok some of them let us try to see one or two more.

(Refer Slide Time: 11:50)



And let us try to understand like what is the accuracy see here. It is such a easy thing right, but still it is giving me shaving beard right.

(Refer Slide Time: 12:02)



(Refer Slide Time: 12:11)

video_classification

Table of contents

- Video Classification with a CNN-RNN Architecture
- Data collection
- Setup
- Define hyperparameters
- Data preparation
- The sequence model
- Inference
- Next steps
- Section

Next steps

- In this example, we made use of transfer learning for extracting meaningful features from video frames. You could also fine-tune the pre-trained network to notice how that affects the end results.
- For speed-accuracy trade-offs, you can try out other models present inside `tf.keras.applications`.
- Try different combinations of `MAX_SEQ_LENGTH` to observe how that affects the performance.
- Train on a higher number of classes and see if you are able to get good performance.
- Following [this tutorial](#), try a [pre-trained action recognition model](#) from DeepMind.
- Rolling-averaging can be useful technique for video classification and it can be combined with a standard image classification model to infer on videos. [This tutorial](#) will help understand how to use rolling-averaging with an image classifier.
- When there are variations in between the frames of a video not all the frames might be equally important to decide its category. In those situations, putting a [self-attention layer](#) in the sequence model will likely yield better results.
- Following [this book chapter](#), you can implement Transformers-based models for processing videos.

So, yes, so basically to remove all of this right, to remove all of this there is a concept are there is this particular approach right which is called as inflated 3D CNN usage right.

(Refer Slide Time: 12:36)

TensorFlow

Resources > Hub > Tutorials

Action Recognition with an Inflated 3D CNN

Run in Google Colab

This Colab demonstrates recognizing actions in video data using the `tfhub.dev/deepmind/3d_kinetics-400/1` module. More models to detect actions in videos can be found [here](#).

The underlying model is described in the paper "Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset" by Joao Carneiro and Andrew Zisserman. The paper was posted on arXiv in May 2017, and was published as a CVPR 2017 conference paper. The source code is publicly available on [github](#).

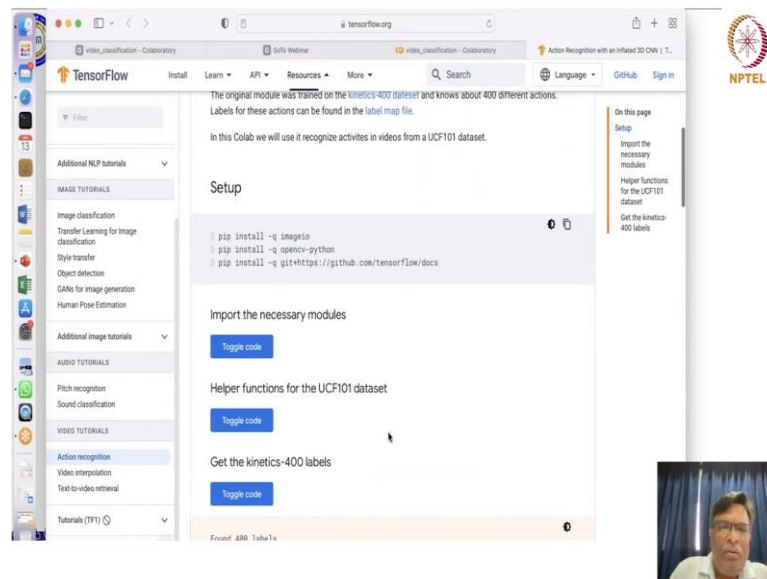
"Quo Vadis" introduced a new architecture for video classification, the Inflated 3D Convnet or I3D. This architecture achieved state-of-the-art results on the UCF101 and HMDB51 datasets from fine-tuning these models. I3D models pre-trained on Kinetics also placed first in the CVPR 2017 [Charades challenge](#).

The original module was trained on the [kinetics-400 dataset](#) and knows about 400 different actions. Labels for these actions can be found in the [label map file](#).

In this Colab we will use it to recognize activities in videos from a UCF101 dataset.

Setup

(Refer Slide Time: 12:41)

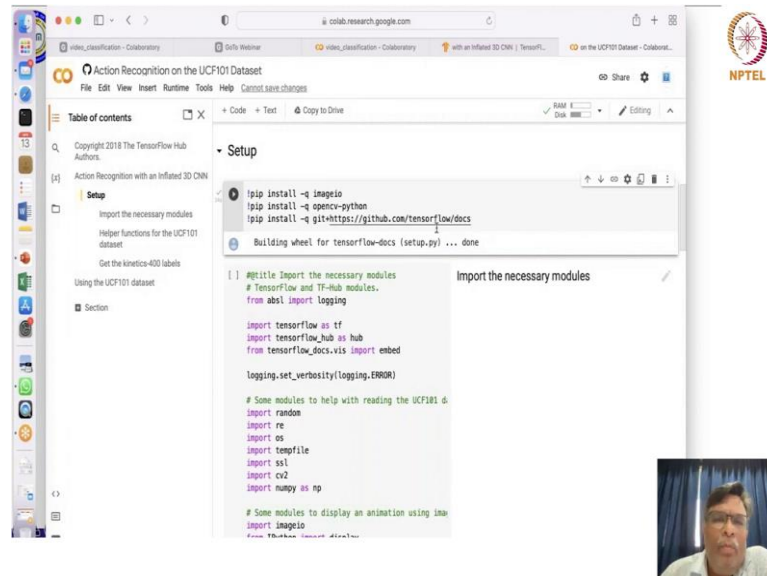


Now, what is basically inflated 3D CNN approach for action recognition is that it also does the same thing and you can use the kinetic data set right which will improve right your recognition capability. So, what is the difference? Here the input again is a video. So, the idea is it is a 3D input with a two dimensional frame with time as the third dimension ok.

So, technically speaking it is not a 3D model, but it is considered to be a three dimensional input it is considered to be a three dimensional input ok as a two dimensional image frame right and time as the third dimension. So, in this particular example we have tried to actually see right that these people have used a CNN with two strides or stride two after which you have a max pooling layer and multiple inception modules right. So, this is how it is.

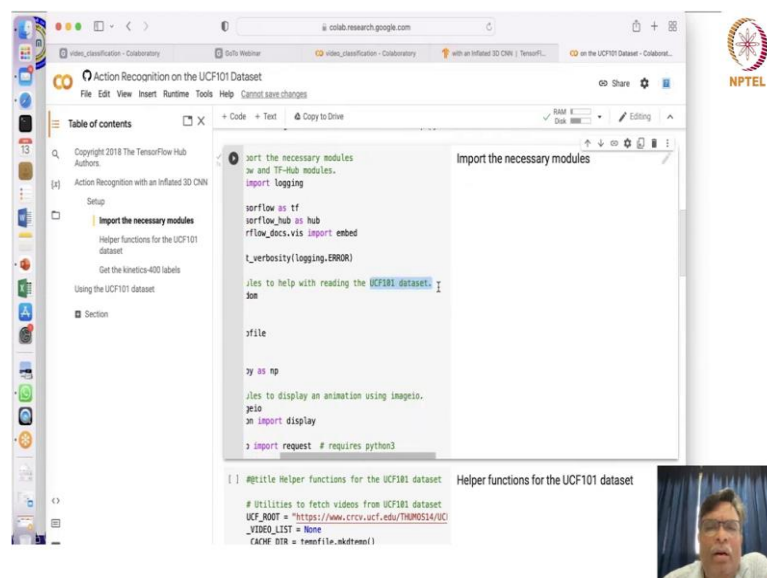
So, in the previous thing what I have done is, I had actually run it before, but now what I am going to do is I am going to run it so, that. So, let me just run in the colab thing now give me some time yes.

(Refer Slide Time: 14:59)

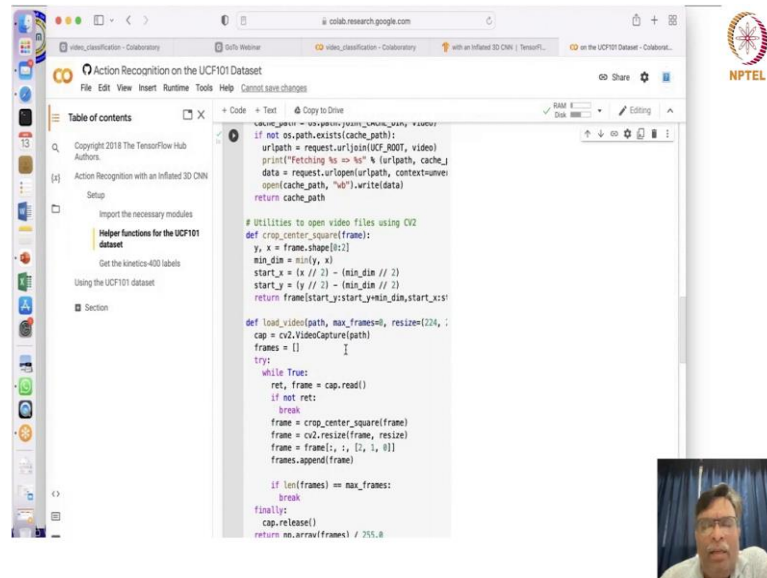


So, ok let me just close that session previous one ok. So, in this case this is basically a solution which was at just the first rank in CVPR 2017 charades challenge right. So, ok once we are able to install the various modules in the our environment we will just try to understand and import the various necessary modules and again here we are using UCF101.

(Refer Slide Time: 15:48)

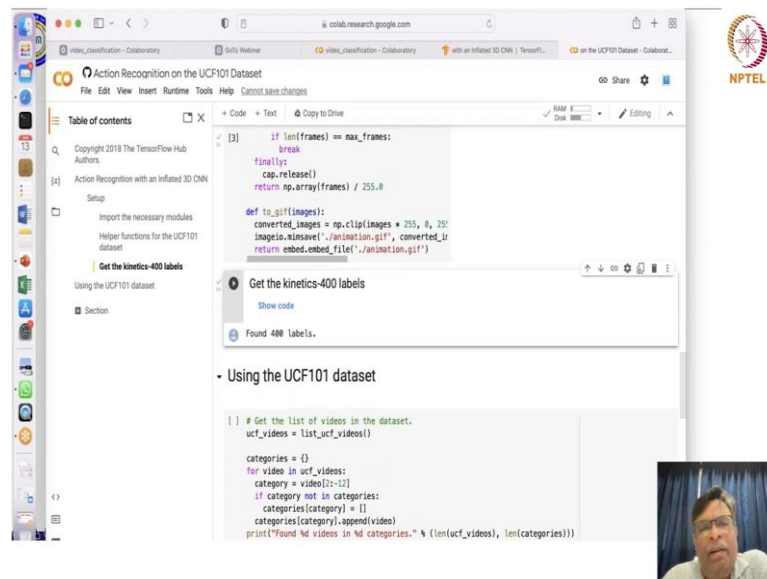


(Refer Slide Time: 16:04)



And then yeah again the same thing we are trying to open the video file using CV2 and then crop the center square frame and then trying to load everything right.

(Refer Slide Time: 16:18)



And then for each of these videos we are trying to get the kinetic right cameras labels.

(Refer Slide Time: 16:28)

Found 400 labels.

Using the UCF101 dataset

```
# Get the list of videos in the dataset.
ucf_videos = list_ucf_videos()

categories = {}
for video in ucf_videos:
    category = video[2:-12]
    if category not in categories:
        categories[category] = []
    categories[category].append(video)
print("Found %d videos in %d categories." % (len(ucf_videos), len(categories)))

for category, sequences in categories.items():
    summary = ", ".join(sequences[12:])
    print("%-20s %d videos (%s, ...)" % (category, len(sequences), summary))
```

125 videos (v_HulaHoop_g01_c01.avi, v_HulaHoop_g01_c02.avi, ...)

158 videos (v_Juggling_g01_c01.avi, v_Juggling_g01_c02.avi, ...)

117 videos (v_JavelinThrow_g01_c01.avi, v_JavelinThrow_g01_c02.avi, ...)

121 videos (v_JugglingBalls_g01_c01.avi, v_JugglingBalls_g01_c02.avi, ...)

144 videos (v_JumpRope_g01_c01.avi, v_JumpRope_g01_c02.avi, ...)

123 videos (v_JumpingJack_g01_c01.avi, v_JumpingJack_g01_c02.avi, ...)

141 videos (v_Kayaking_g01_c01.avi, v_Kayaking_g01_c02.avi, ...)

123 videos (v_Knitting_g01_c01.avi, v_Knitting_g01_c02.avi, ...)

131 videos (v_LongJump_g01_c01.avi, v_LongJump_g01_c02.avi, ...)

127 videos (v_Lunges_g01_c01.avi, v_Lunges_g01_c02.avi, ...)

174 videos (v_MilitaryParade_g01_c01.avi, v_MilitaryParade_g01_c02.avi, ...)

And then; so, what is the list of videos in the data set right?

(Refer Slide Time: 16:00)

JumpingJack 123 videos (v_JumpingJack_g01_c01.avi, v_JumpingJack_g01_c02.avi, ...)

Kayaking 141 videos (v_Kayaking_g01_c01.avi, v_Kayaking_g01_c02.avi, ...)

Knitting 123 videos (v_Knitting_g01_c01.avi, v_Knitting_g01_c02.avi, ...)

LongJump 131 videos (v_LongJump_g01_c01.avi, v_LongJump_g01_c02.avi, ...)

Lunges 127 videos (v_Lunges_g01_c01.avi, v_Lunges_g01_c02.avi, ...)

MilitaryParade 125 videos (v_MilitaryParade_g01_c01.avi, v_MilitaryParade_g01_c02.avi, ...)

Mixing 136 videos (v_Mixing_g01_c01.avi, v_Mixing_g01_c02.avi, ...)

MoppingFloor 110 videos (v_MoppingFloor_g01_c01.avi, v_MoppingFloor_g01_c02.avi, ...)

Nunchucks 132 videos (v_Nunchucks_g01_c01.avi, v_Nunchucks_g01_c02.avi, ...)

ParallelBars 114 videos (v_ParallelBars_g01_c01.avi, v_ParallelBars_g01_c02.avi, ...)

PizzaTossing 113 videos (v_PizzaTossing_g01_c01.avi, v_PizzaTossing_g01_c02.avi, ...)

PlayingCello 164 videos (v_PlayingCello_g01_c01.avi, v_PlayingCello_g01_c02.avi, ...)

PlayingGuitar 151 videos (v_PlayingGuitar_g01_c01.avi, v_PlayingGuitar_g01_c02.avi, ...)

PlayingHarp 164 videos (v_PlayingHarp_g01_c01.avi, v_PlayingHarp_g01_c02.avi, ...)

PlayingLute 155 videos (v_PlayingLute_g01_c01.avi, v_PlayingLute_g01_c02.avi, ...)

PlayingSitar 160 videos (v_PlayingSitar_g01_c01.avi, v_PlayingSitar_g01_c02.avi, ...)

PlayingTabla 185 videos (v_PlayingTabla_g01_c01.avi, v_PlayingTabla_g01_c02.avi, ...)

PlayingViolin 157 videos (v_PlayingViolin_g01_c01.avi, v_PlayingViolin_g01_c02.avi, ...)

PoleVault 111 videos (v_PoleVault_g01_c01.avi, v_PoleVault_g01_c02.avi, ...)

PommelHorse 180 videos (v_PommelHorse_g01_c01.avi, v_PommelHorse_g01_c02.avi, ...)

PullUps 140 videos (v_PullUps_g01_c01.avi, v_PullUps_g01_c02.avi, ...)

Punch 123 videos (v_Punch_g01_c01.avi, v_Punch_g01_c02.avi, ...)

PushUps 180 videos (v_PushUps_g01_c01.avi, v_PushUps_g01_c02.avi, ...)

Rafting 111 videos (v_Rafting_g01_c01.avi, v_Rafting_g01_c02.avi, ...)

RopeClimbingIndoor 144 videos (v_RopeClimbingIndoor_g01_c01.avi, v_RopeClimbingIndoor_g01_c02.avi, ...)

RopeClimbing 119 videos (v_RopeClimbing_g01_c01.avi, v_RopeClimbing_g01_c02.avi, ...)

Rowing 137 videos (v_Rowing_g01_c01.avi, v_Rowing_g01_c02.avi, ...)

SalsaSpin 133 videos (v_SalsaSpin_g01_c01.avi, v_SalsaSpin_g01_c02.avi, ...)

ShavingBoard 161 videos (v_ShavingBoard_g01_c01.avi, v_ShavingBoard_g01_c02.avi, ...)

Shotput 144 videos (v_Shotput_g01_c01.avi, v_Shotput_g01_c02.avi, ...)

Skateboarding 120 videos (v_Skateboarding_g01_c01.avi, v_Skateboarding_g01_c02.avi, ...)

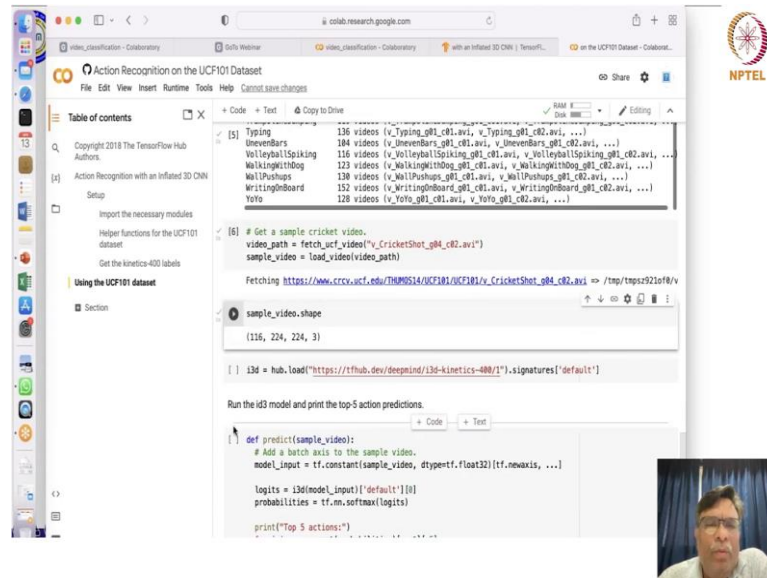
Skiing 135 videos (v_Skiing_g01_c01.avi, v_Skiing_g01_c02.avi, ...)

Skijet 180 videos (v_Skijet_g01_c01.avi, v_Skijet_g01_c02.avi, ...)

SkyDiving 110 videos (v_SkyDiving_g01_c01.avi, v_SkyDiving_g01_c02.avi, ...)

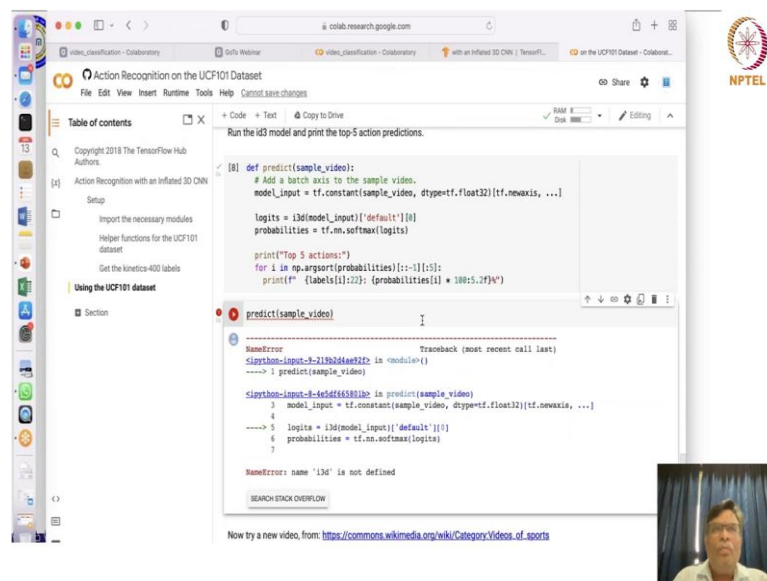
SoccerJuggling 147 videos (v_SoccerJuggling_g01_c01.avi, v_SoccerJuggling_g01_c02.avi, ...)

(Refer Slide Time: 16:39)



All of this videos in the initial this thing and then it is trying to show you certain things wherein.

(Refer Slide Time: 16:59)



Let us try to; miss this ok.

(Refer Slide Time: 17:32)

Table of contents

- Copyright 2018 The TensorFlow Hub Authors
- Action Recognition with an Inflated 3D CNN
 - Setup
 - Import the necessary modules
 - Helper functions for the UCF101 dataset
 - Get the kinetics-400 labels
 - Using the UCF101 dataset
 - Section

```

[12] # Add a batch axis to the sample video.
model_input = tf.constant(sample_video, dtype=tf.float32)[tf.newaxis, ...]

logits = t3d(model_input)[:, default][:10]
probabilities = tf.nn.softmax(logits)

print("Top 5 actions:")
for i in np.argsort(probabilities)[::-1][:5]:
    print(f" {labels[i]:22}: (probabilities[i] * 100):5.2f%%")

predict(sample_video)

Top 5 actions:
playing cricket      : 97.77%
skateboarding       :  0.71%
robot dancing       :  0.56%
roller skating      :  0.56%
golf putting        :  0.13%

Now try a new video, from: https://commons.wikimedia.org/wiki/Category:Videos\_of\_sports
How about this video by Patrick Gillett:

[ ] !curl -O https://upload.wikimedia.org/wikipedia/commons/9/96/End_of_a_jam.ogv

[ ] video_path = "End_of_a_jam.ogv"

[ ] sample_video = load_video(video_path)[:100]
sample_video.shape

```

So, this sample video which you choose will give you the probabilities right it is not displaying what it will give you the probabilities.

(Refer Slide Time: 18:19)

```

[14] predict(sample_video)

Top 5 actions:
playing cricket      : 97.77%
skateboarding       :  0.71%
robot dancing       :  0.56%
roller skating      :  0.56%
golf putting        :  0.13%

Now try a new video, from: https://commons.wikimedia.org/wiki/Category:Videos\_of\_sports
How about this video by Patrick Gillett:

[15] !curl -O https://upload.wikimedia.org/wikipedia/commons/9/96/End_of_a_jam.ogv

% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 55.0M  100 55.0M    0     0  25.0M      0     0  0:00:02  0:00:02 --:--:-- 25.0M

[ ] video_path = "End_of_a_jam.ogv"

[ ] sample_video = load_video(video_path)[:100]
sample_video.shape

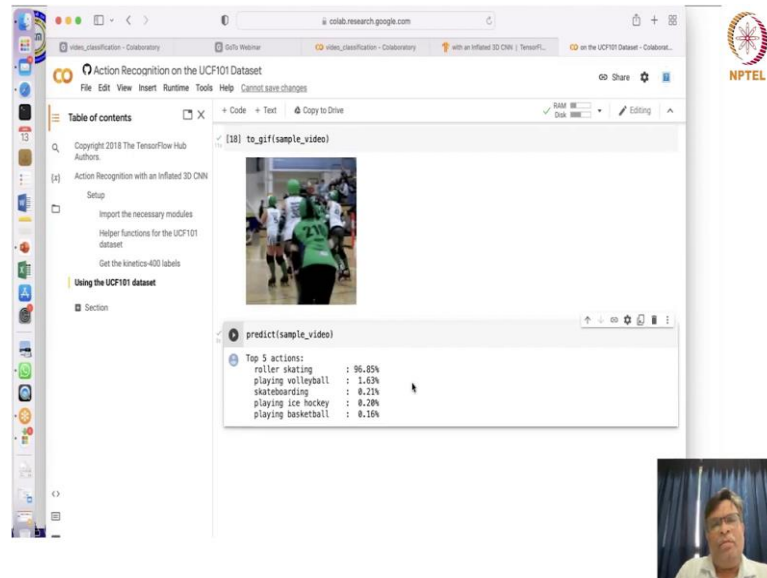
[ ] to_gif(sample_video)

[ ] predict(sample_video)

```

And then you can basically try to download certain videos ok trying to understand then.

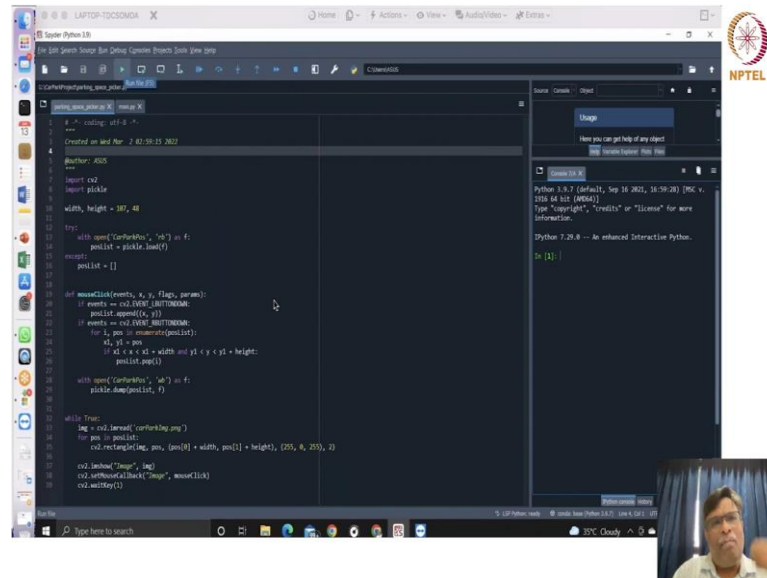
(Refer Slide Time: 18:59)



Yes. So, if you see this, this basically is trying to give you a prediction which is about 96.85 percent ok. Now, the idea here is that you have done almost the same thing, but then you have tried to actually link ok the kinetic label right with basically trying to use the kinetic data set ok along with the UCF101 and HMD basically data sets for fine tuning the models ok. And now these basic models which we are talking of a 3D convnet ok if you go to the 3D convnet model ok these are free trained models which are available to us.

So, 3D convnet model basically are available to us and then yeah. So, that is what basically is done in this particular thing I am not going into the technical details of 3D convnet and all you can basically refer the various links which are there here and then this thing which can happen right. So, now you can predict anything you can work on this in your leisure time and you can work basically on this, we have shared the links on our slides as well yeah. So, let me just show you some other demos one minute give me a sec.

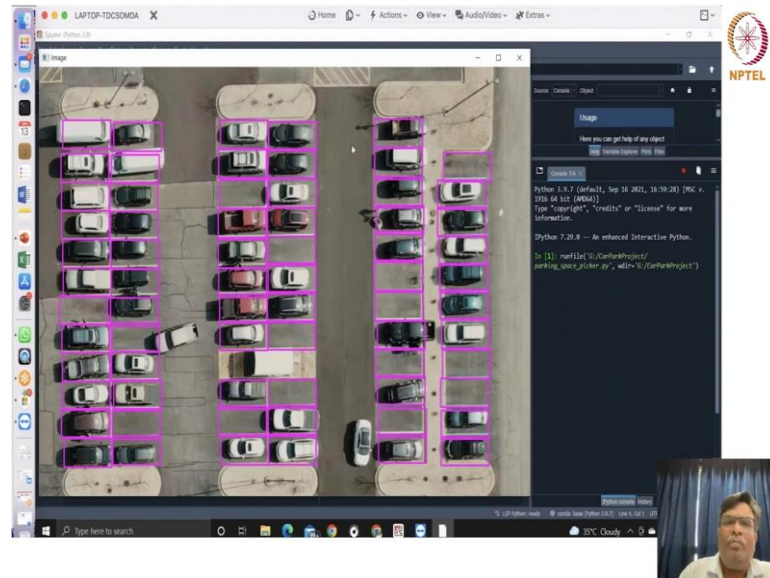
(Refer Slide Time: 20:54)



So, the idea is that this is a very simple program which we have been doing like everybody does it, but the idea is just to show you right how basically you can further automate it right which we will be doing it in the next class with deep stream right. So, the idea was how standard applications like this could be improved right using deep stream and GPU computing is what we are going to show you in the next class so, but we will show you how this is stand in the standard procedure done right.

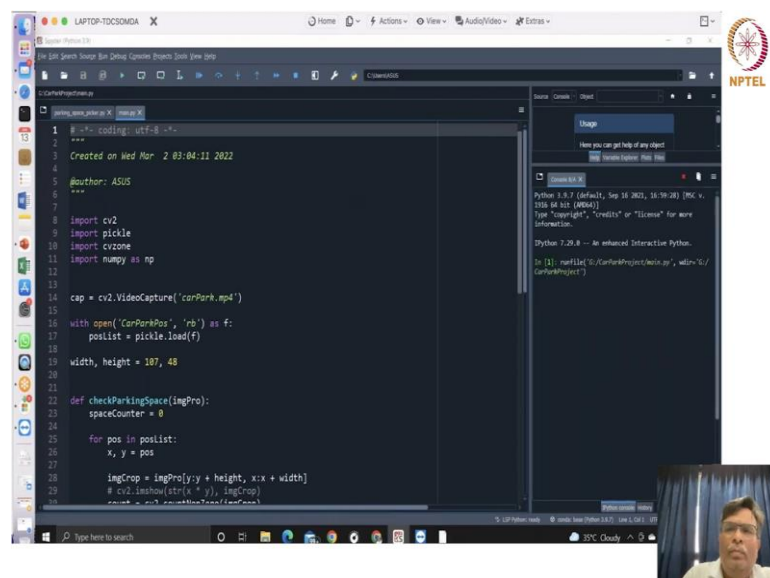
So, we have this width and height. So, trying to understand that and then trying to basically use mouse clicks to basically find out the areas which you are going to select. So, we will show you that and then we will see how this car parking thing works right. So, we will try to show you this yeah yes.

(Refer Slide Time: 21:55)



So, if you see this one minute. So, this is space for drawing yeah. So, this is how actually you are going to draw it ok yeah. So, this is how you are actually going to draw it and then now you can see like this we have put in and drawn ok for majority of this trying to understand this. And then trying to you can add in go on adding more areas and something like that that should not be a problem. So, this is the basic idea and now if you see and we start the next step of processing yes.

(Refer Slide Time: 22:52)



(Refer Slide Time: 22:57)



So, if you can see here it is actually going to show us the free slots the count of number of occupied slots and how basically the frames are getting processed ok and then thresholds are getting checked right. So, this is the standard computer vision application ok which might run on a GPU which might not run on a GPU. So, this is the standard application wherein you are trying to detect the free slots ok based on the features available using certain techniques of thresholding.

These are not very standard in the sense it is a standard technique, but it does not use any deep neural network type of approach right. So, how are we going to change this application ok? To a real time smart city application using something which is called as deep stream which is what we are going to discuss in the next session right.

So, I hope you understand the context with which this is told today so, that we can actually go into the details of certain specific applications which would be developed using things like deep stream ok which is a very very good set of libraries right which could be used for developing video analytics applications right. So, I suppose I am done.