VIDEO LINK:
https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing

Table of Contents

VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing

Abstract

In this project we have tried to simplify a very tough problem that is video classification. Video classification is tougher from image classification because here problem lies with the collection of dataset and training the model. The size of the video dataset is very large and also training the model with those dataset is a tough job as it consume time and not possible with normal daily use laptops with basic specs. So, for solving this problem we have designed a technique of implementing video classification through image classification. We know that videos are collection of thousands of frames (images) and therefore we have come out with a solution of converting videos into images and them forming image classification on them. This will reduce both the dataset size and also the model training time. We have designed a CNN model and used it in two application one is to classify sports video of cricket, football, hockey and another to classify hand shape using real human hand videos.

Ackonoweldgement

In this project we acknowledge difficulties of high time complexity and workload during the process of video classification and give counter technique to solve its issue. For solving this problem we have designed a technique of implementing video classification through image classification. We know that videos are collection of thousands of frames (images) and therefore we have come out with a solution of converting videos into images and them forming image classification on them. This will reduce both the dataset size and also the model training time.

1.Introduction

1.1 Motivation

We are motivated to make a video classification system which will be better in time load and searching videos of different contents and variety of world helping in better construct of this system.

1.2. Aim of the proposed Work

we acknowledge difficulties of high time complexity and workload during the process of video classification and give counter technique to solve its issue. For solving this problem we have designed a technique of implementing video classification through image classification. We know that videos are collection of thousands of frames (images) and therefore we have come out with a solution of converting videos into images and them forming image classification on them.

1.3. Objective of the proposed work

The purpose of this document is to classify the video, which ease the online website like youtube. Through which we can easily classify the nature of the video.

2. Literature Survey

Classifying video presents unique challenges for machine learning models. As we will covered here, video has the

VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing

added (and interesting) property of temporal features in addition to the spatial features present in 2D images. While this additional information provides us more to work with, it also requires different network architectures and, often, adds larger memory and computational demands.

Various video classification methods

1. Classifying one frame at a time with a ConvNet
2. Using a time-distributed ConvNet and passing the features to an RNN, in one network
3. Using a 3D convolutional network
4. Extracting features from each frame with a ConvNet and passing the sequence to a separate RNN
5. Extracting features from each frame with a ConvNet and passing the sequence to a separate MLP

Constraints:

Each of these methods could be its own blog post (or ten), so we'll impose a few constraints to help simplify things and also to keep down computational complexity for future applications in real-time systems:

1. We won't use any optical flow images. This reduces model complexity, training time, and a whole whackload of hyperparemeters we don't have to worry about.
2. Every video will be subsampled down to 40 frames. So a 41-frame video and a 500 frame video will both be reduced to 40 frames, with the 500-frame video essentially being fast-forwarded.
3. We won't do much preprocessing. A common preprocessing step for video classification is subtracting the mean, but we'll keep the frames pretty raw from start to finish.
4. Every model has to fit into the 12 GiB of memory provided to us in the GPU in the new AWS p2.xlarge instances.

With these constraints, we know we won't hit the ~94% state-of-the-art accuracy, but we'll see if we can go in that direction.

Dataset

VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing

We're going to use the popular UCF101 dataset. I find this dataset to have a great balance of classes and training data, as well as a lot of well-documented benchmarks for us to judge ourselves against. And unlike some of the newer video datasets (see YouTube-8M), the amount of data is manageable on modern systems.

UCF summarizes their dataset well:

With 13,320 videos from 101 action categories, UCF101 gives the largest diversity in terms of actions and with the presence of large variations in camera motion, object appearance and pose, object scale, viewpoint, cluttered background, illumination conditions, etc, it is the most challenging data set to date.

Challenge accepted!

Data preparation

The first thing we need to do is get the data in a format we can train on. We accomplish this in three steps:

1. Split all the videos into train/test folders
2. Extract jpegs of each frame for each video
3. Summarize the videos, their class, train/test status and frame count in a CSV we'll reference throughout our training.

One important note is that the training set has many videos from the same "group". This means there may be multiple videos of the same person from the same angle in the same setting performing the same action. It's crucial that we don't end up with videos from the same group in both the train and test groups, as we'd score unrealistically high on these classes.

UCF provides three train/test split recommendations that we can follow. For the sake of time, we use just split #1 for all of our experiments.

A note about the graphs below

Each graph includes three series:

1. The CNN-only top 1 accuracy in **red**, used as a baseline.

VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing

2. The top 5 categorical accuracy in **green**.

3. The top 1 categorical accuracy in **blue**.

I apologize for the lack of legend and the fugliness of the Matplotlib charts!

Method #0: Randomly guess or always choose the most common

This isn't a real classification method, but if our model can't beat random or the most used, we're definitely not on the right track!

Trying to randomly guess the best result gives us ~0.9% accuracy. This makes sense since there are 101 classes, and, well… math.

Always guessing the most common class, "TennisSwing", yields 1.32%. Also makes sense since TennisSwing labels are ~1.32% of our dataset. Okay, we're on the right track, and we have something to beat. Let's build some models.

Method #1: Classify one frame at a time with a CNN

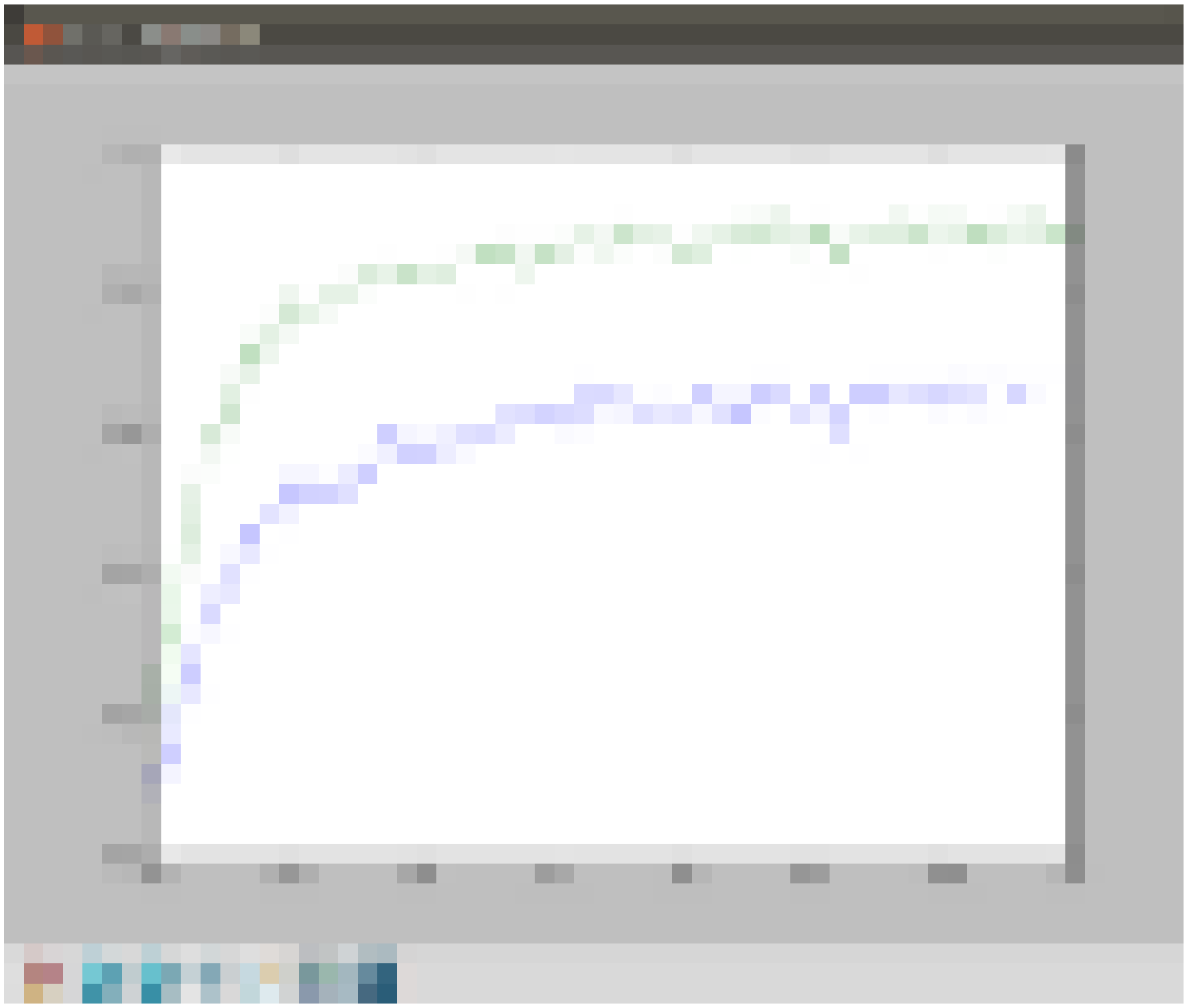
For our first method, we'll ignore the temporal features of video and attempt to classify each clip by looking at a single frame. We'll do this by using a CNN, AKA ConvNet. More specifically, we'll use Inception V3, pre-trained on ImageNet.

We'll use transfer learning to retrain Inception on our data. This takes two steps.

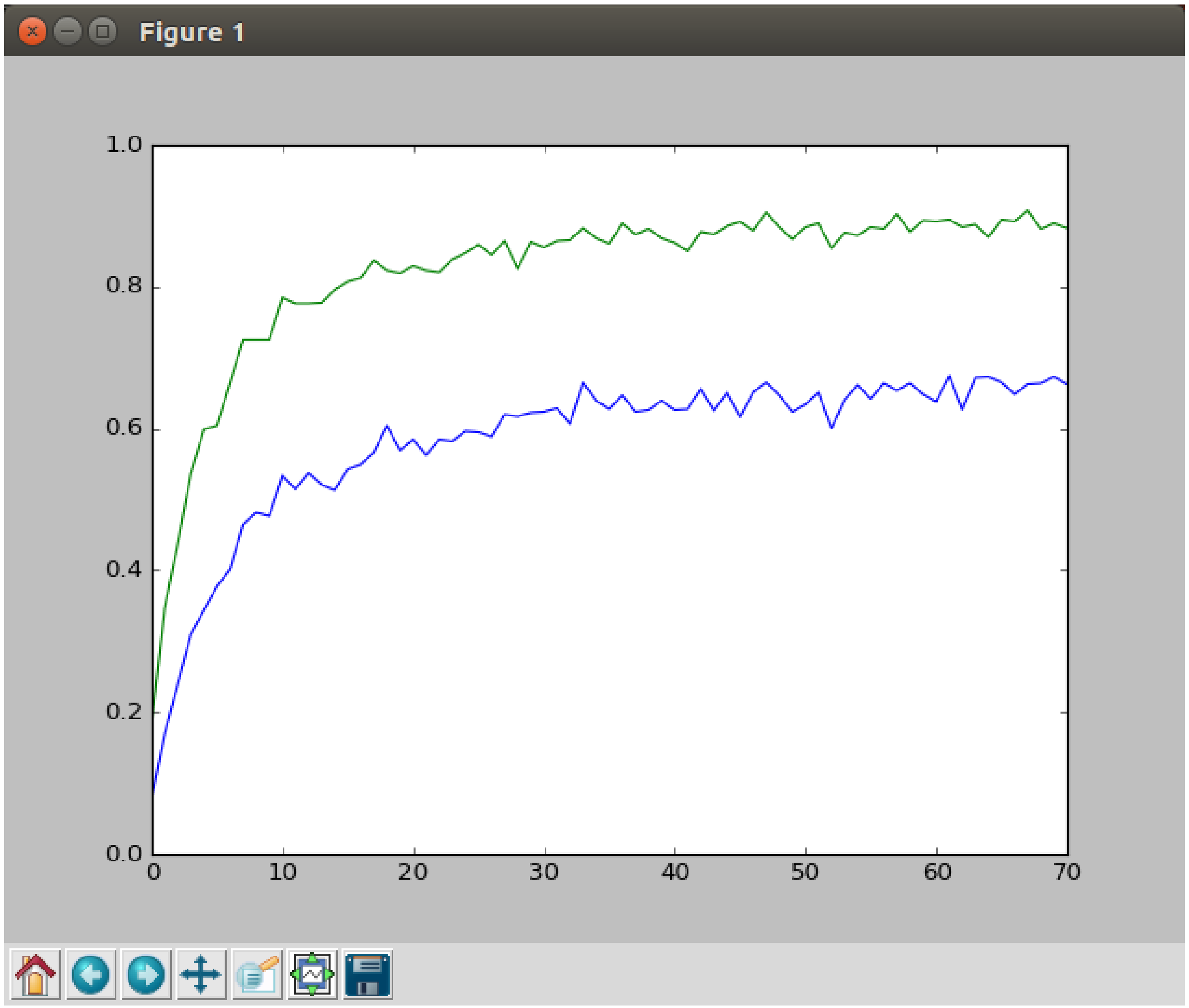
First, we fine-tune the top dense layers for 10 epochs (at 10,240 images per epoch) in an attempt to retain as much of the previous learning as possible. Fine tuning the top dense layers get us to ~52% top-1 validation accuracy, so it's a great shortcut!

Next, we retrain the top two inception blocks. Coming up on 70 epochs, we're looking really good, achieving a top 1 test accuracy of about 65%!

VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing



VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing



The blue line denotes the top 1 accuracy, while the green line denotes the top 5 accuracy.

It’s worth noting that we’re literally looking at each frame independently, and classifying the entire video based solely on that one frame. We aren’t looking at all the frames and doing any sort of averaging or max-ing.

Let’s spot check a random sample of images from the test set to see how we did:

VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing

Predictions: HighJump: 0.37, FloorGymnastics: 0.14, SoccerPenalty: 0.09. **Actual**: Nunchucks. **Result:** Fail

Predictions: WallPushups: 0.67, BoxingPunchingBag: 0.09, JugglingBalls: 0.08
. **Actual:** Wallpushups. **Result:** Top 1 correct!

VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing

Predictions: Drumming: 1.00. **Actual**: Drumming. **Result**: Top 1 correct!

VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing

Predictions: HandstandWalking: 0.32, Nunchucks: 0.16, JumpRope: 0.11 . **Actual**: JumpRope. **Result**: Top 5 correct!

Final test accuracy: ~65% top 1, ~90% top 5

Method #2: Use a time-distributed CNN, passing the features to an RNN, in one network

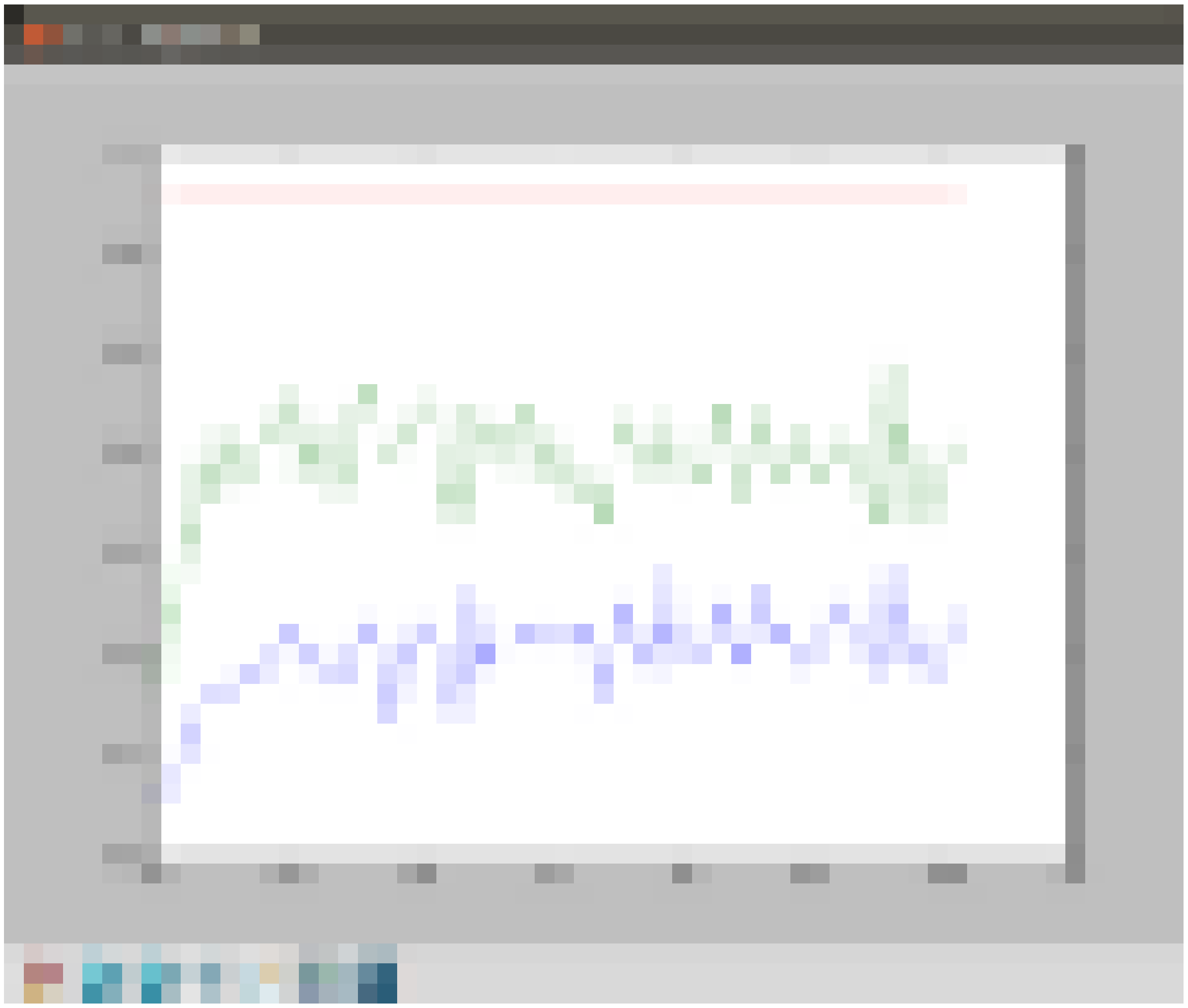
Now that we have a great baseline with Inception to try to beat, we'll move on to models that take the temporal features of video into consideration. For our first such net, we'll use Kera's awesome TimeDistributed wrapper, which allows us to distribute layers of a CNN across an extra dimension — time. Obviously.

For the ConvNet part of the model, we'll use a very small VGG16-style network. Ideally we'd use a deeper network for this part, but given that we have to load the whole thing into GPU memory, our options are fairly limited.

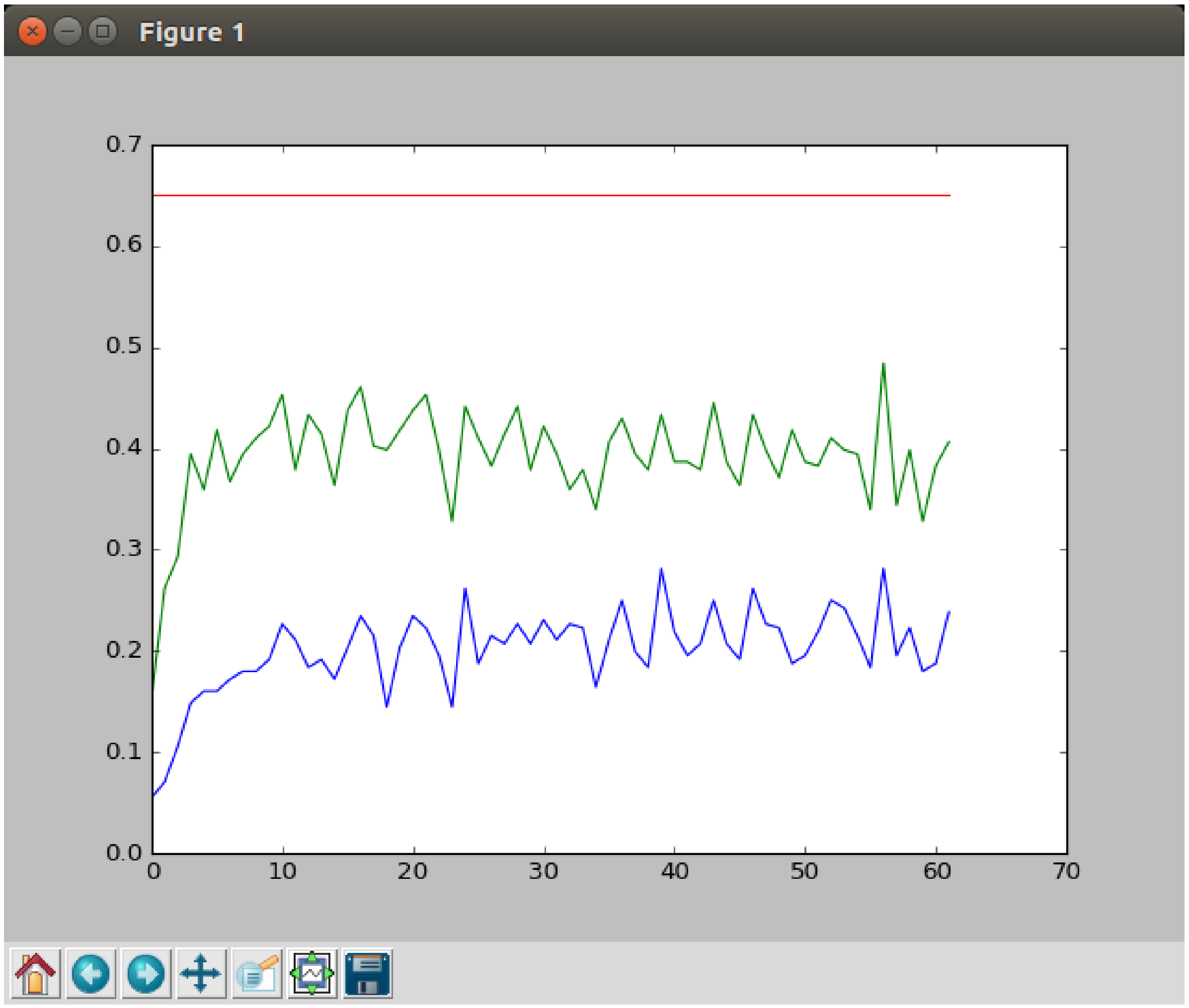
For the RNN part of the net, we'll use a three-layer GRU, each consisting of 128 nodes, and a 0.2 dropout between each layer.

Unlike with method #1, where we got to use the pre-trained ImageNet weights, we'll have to train the whole model on our data from scratch here. This could mean that we'll require too much data or a much bigger network to achieve the same level of accuracy as the Inception whopper produced. However, it also means that our CNN weights will be updated on each backprop pass along with the RNN. Let's see how it does!

VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing



VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing



In all charts, the red line is the Method #1 benchmark, green is the top 5 categorical accuracy, and blue is the top 1 categorical accuracy.

Yikes. How disappointing. Looks like we may need a more complex CNN to do the heavy lifting.

Final test accuracy: 20% top 1, 41% top 5

Method #3: Use a 3D convolutional network

Okay so training a CNN and an LSTM together from scratch didn't work out too well for us. How about 3D convolutional networks?

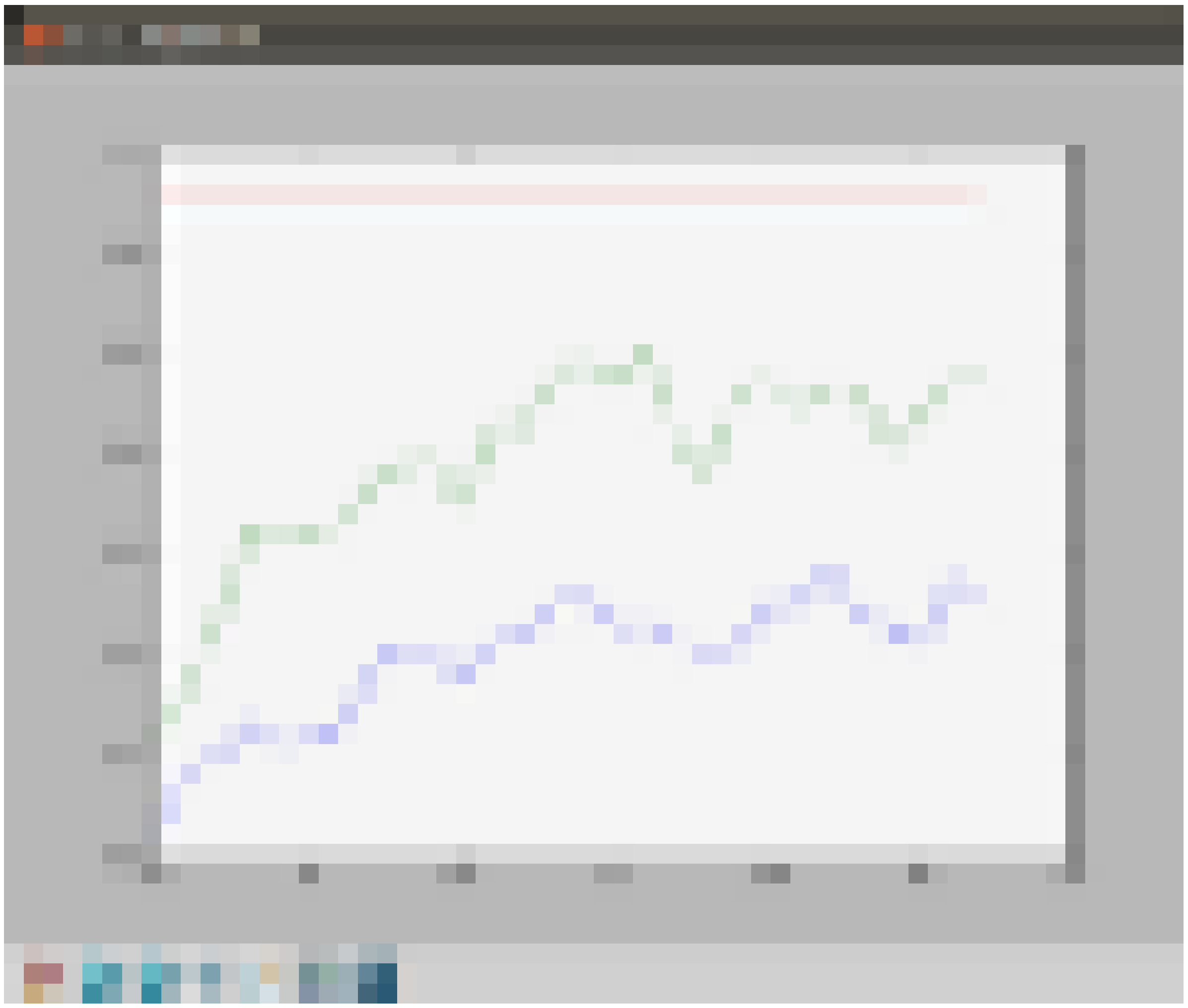
3D ConvNets are an obvious choice for video classification since they inherently apply convolutions (and max poolings) in the 3D space, where the third dimension in our case is time. (Technically speaking it's 4D, since our 2D images are represented as 3D vectors, but the net result is the same.)

However, they have the same drawback we ran into with method #2: memory! In Learning Spatiotemporal Features with 3D Convolutional Networks, the authors propose a network they call C3D that achieves 52.8% accuracy on

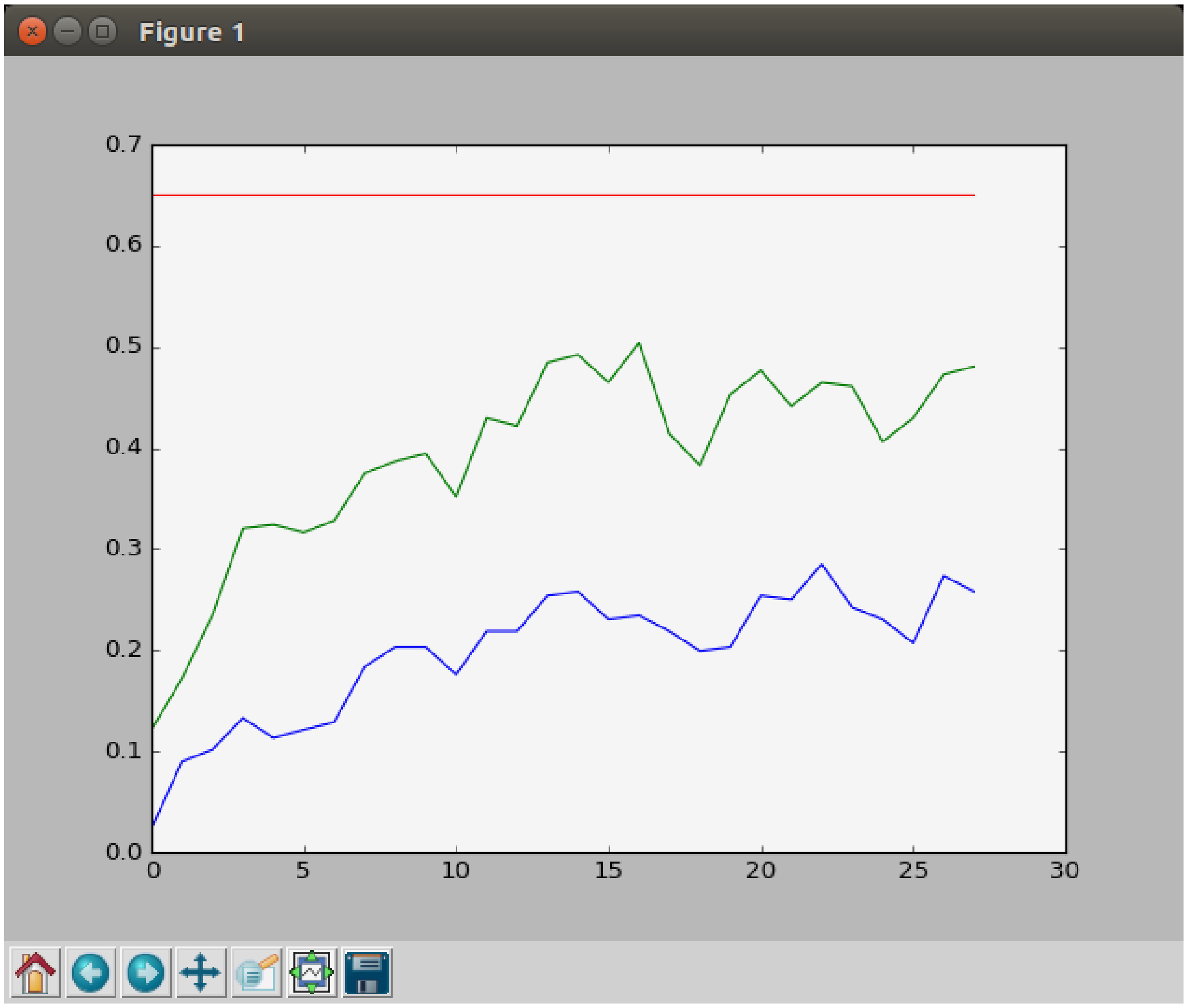
VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing

UCF101. I was excited to attempt to reproduce these results, but I was stalled out with memory limitations of the 12 GiB GPU in the P2. The C3D simply wouldn't run, even as I hacked off layer after layer.

As a plan B, I designed a smaller derivative, consisting of just three 3D convolutions, growing in size from 32 to 64 to 128 nodes. How'd we do?



VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing



After 28 epochs, we aren't even close to hitting the benchmark we set with Inception. I did reduce the learning rate from 5e-5 to 1e-6 and trained for another 30 epochs (not graphed), which got us a little better, but still not in the ballpark.

Maybe training from top-to-bottom isn't the way to go. Let's go another direction.

Final test accuracy: 28% top 1, 51% top 5

Method #4: Extract features with a CNN, pass the sequence to a separate RNN

Given how well Inception did at classifying our images, why don't we try to leverage that learning? In this method, we'll use the Inception network to extract features from our videos, and then pass those to a separate RNN.

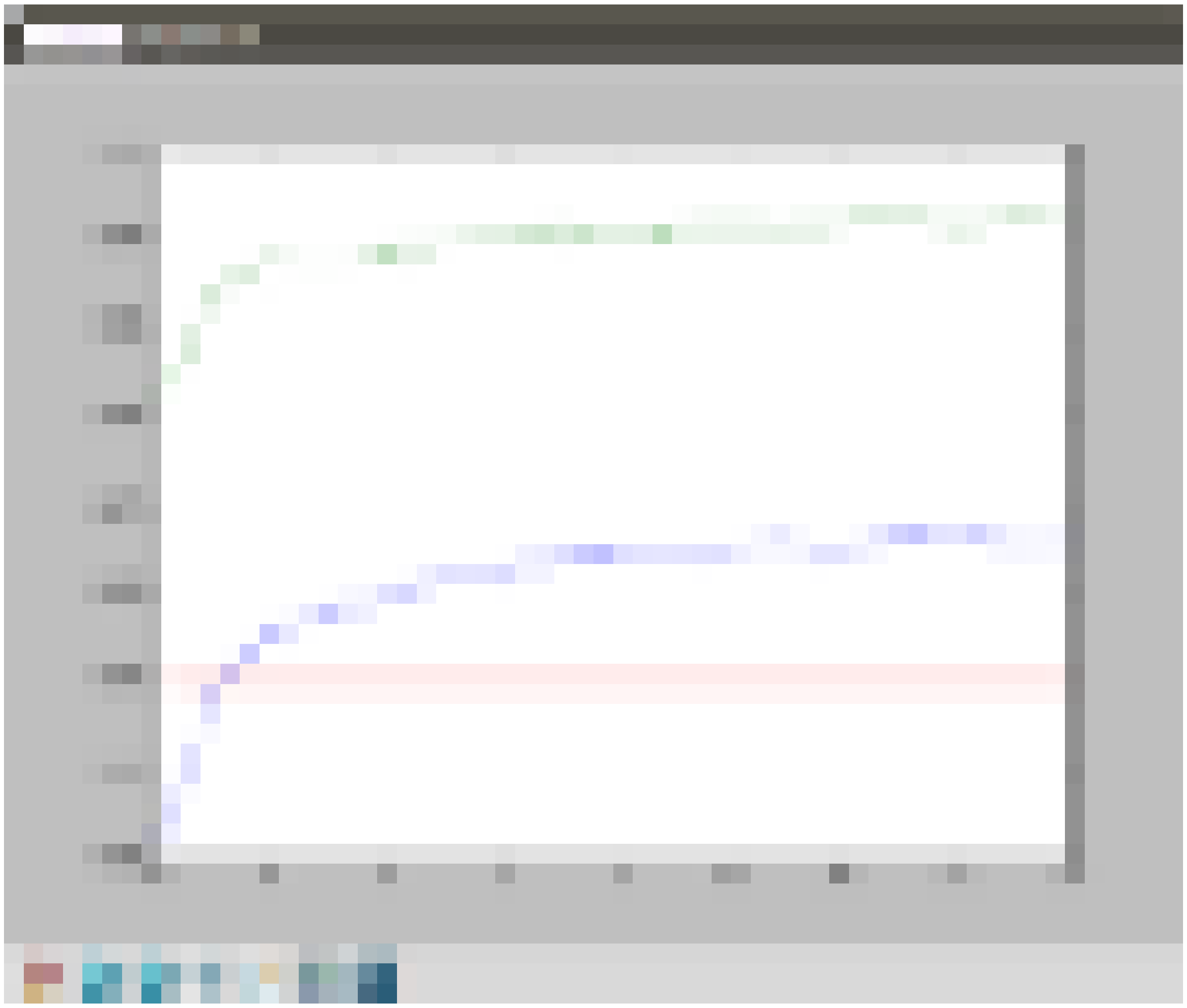
This takes a few steps.

VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing

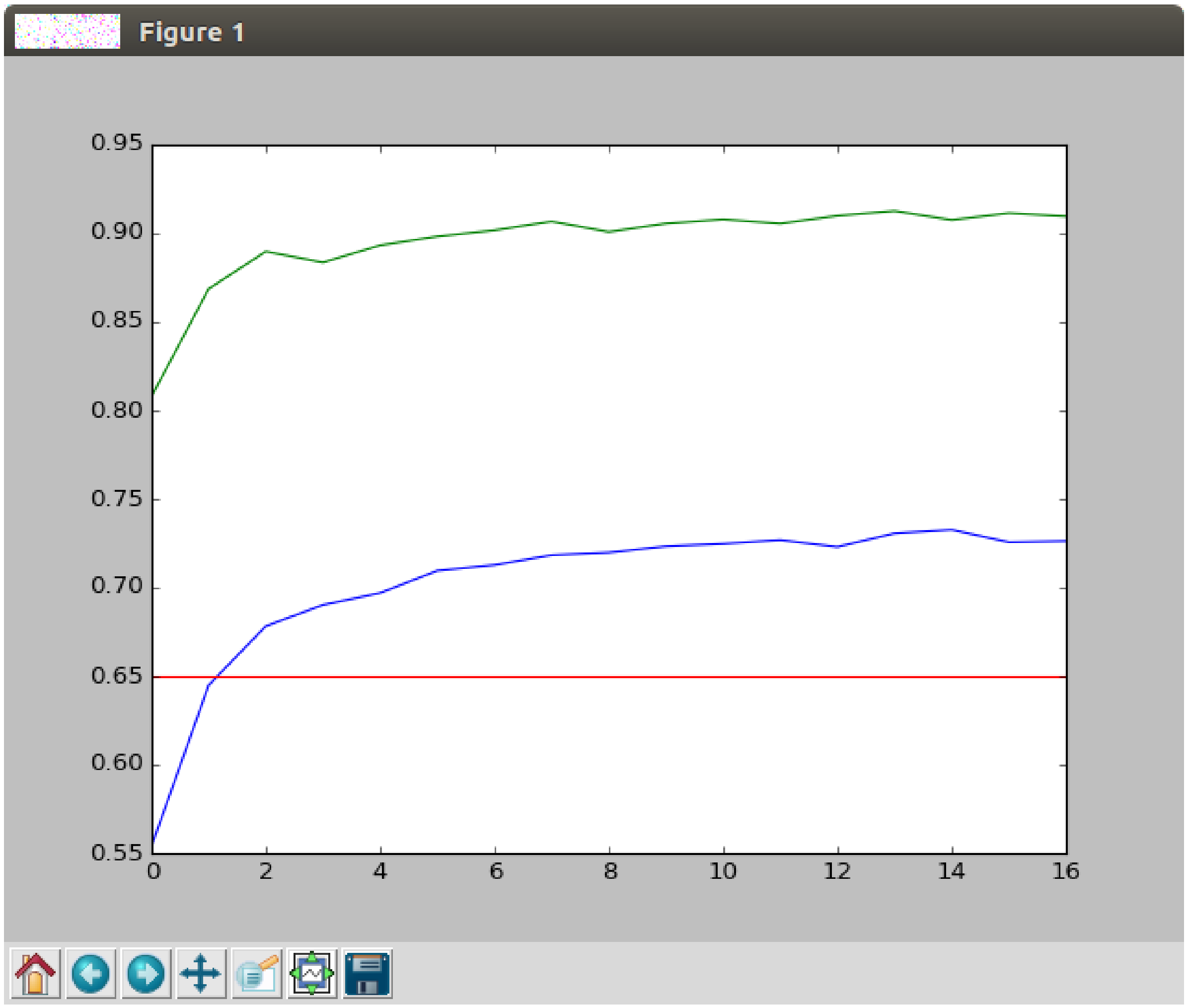
First, we run every frame from every video through Inception, saving the output from the final pool layer of the network. So we effectively chop off the top classification part of the network so that we end up with a 2,048-d vector of features that we can pass to our RNN. For more info on this strategy, see my previous blog post on continuous video classification.

Second, we convert those extracted features into *sequences* of extracted features. If you recall from our constraints, we want to turn each video into a 40-frame sequence. So we stitch the sampled 40 frames together, save that to disk, and now we're ready to train different RNN models without needing to continuously pass our images through the CNN every time we read the same sample or train a new network architecture.

For the RNN, we use a single, 4096-wide LSTM layer, followed by a 1024 Dense layer, with some dropout in between. This relatively shallow network outperformed all variants where I tried multiple stacked LSTMs. Let's take a look at the results:



VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing



The RNN handily beats out the CNN-only classification method.

Hey that's pretty good! Our first temporally-aware network that achieves better than CNN-only results. And it does so by a significant margin.

Final test accuracy: 74% top 1, 91% top 5

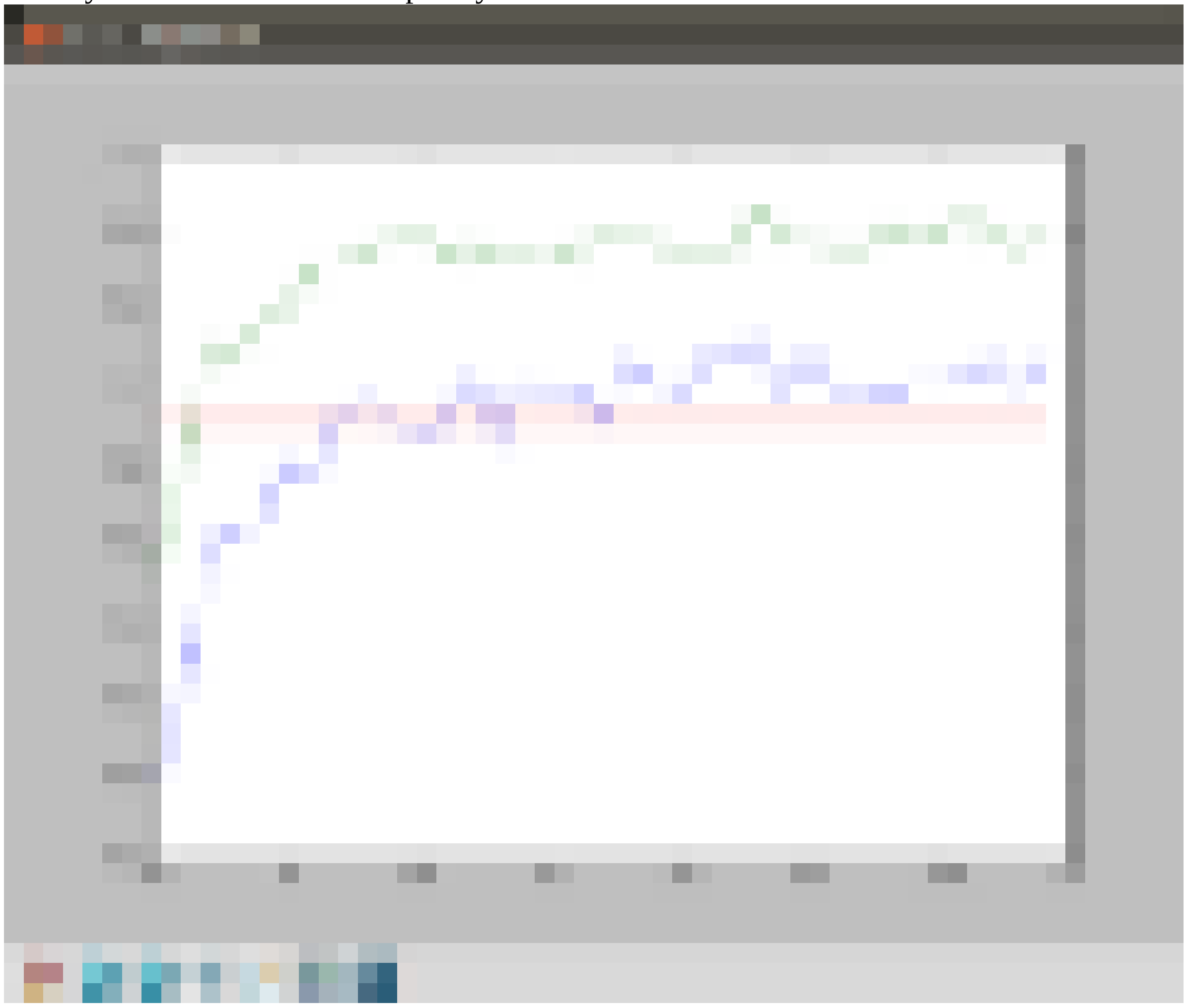
Method #5: Extract features from each frame with a CNN and pass the sequence to an MLP

Let's apply the same CNN extraction process as in the previous method, but instead of sending each piece of the sequence to an RNN, we'll flatten the sequence and pass the new (2,048 x 40) input vector into a fully connected network, AKA a multilayer perceptron (MLP). The hypothesis is that the MLP will be able to infer the temporal features from the sequence organically, without it having to know it's a sequence at all.

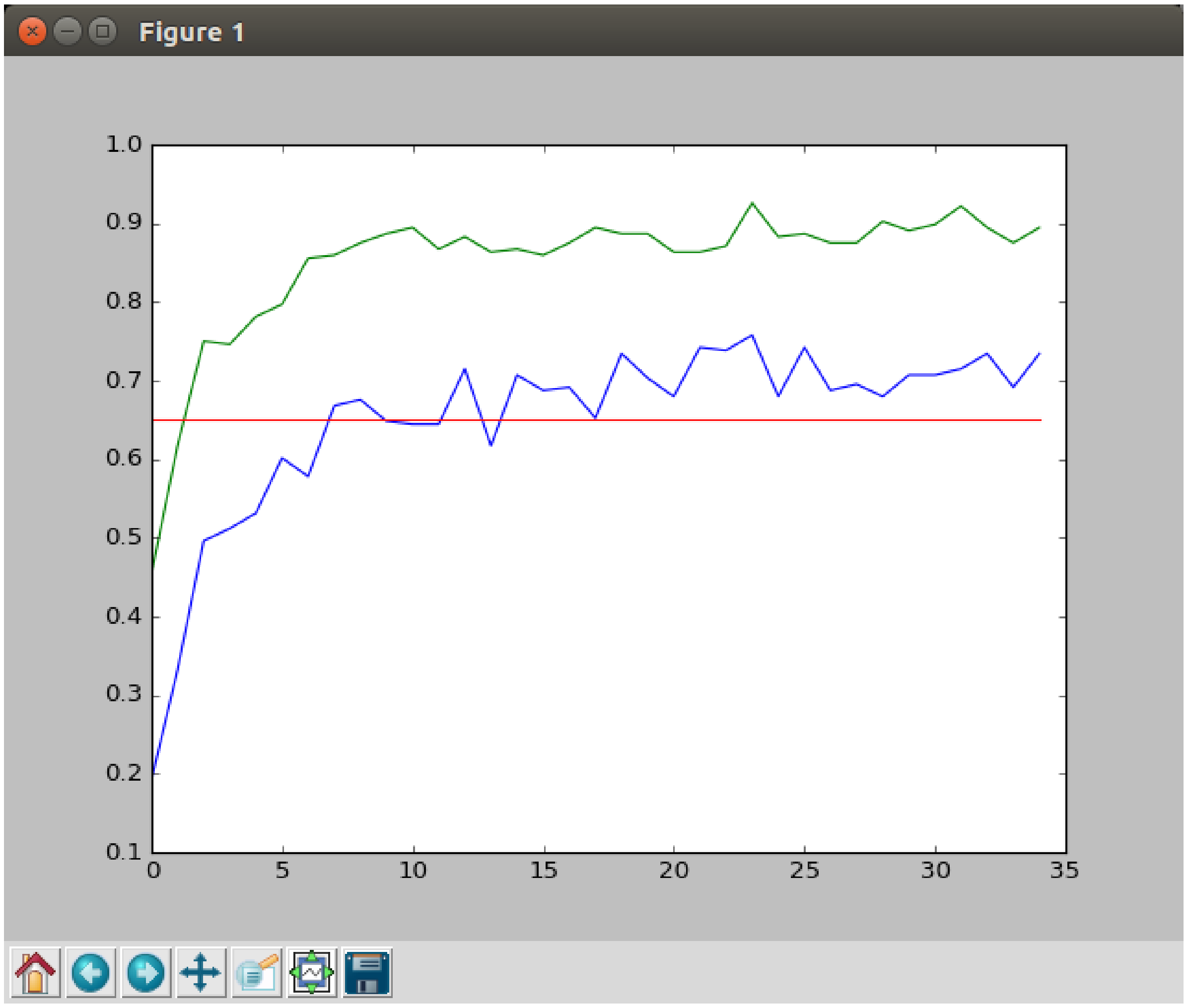
(Plus, "multilayer perceptron" is one of the coolest terms in data science.)

VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing

After trying quite a few deep, shallow, wide and narrow networks, we find that the most performant MLP is a simple two-layer net with 512 neurons per layer:



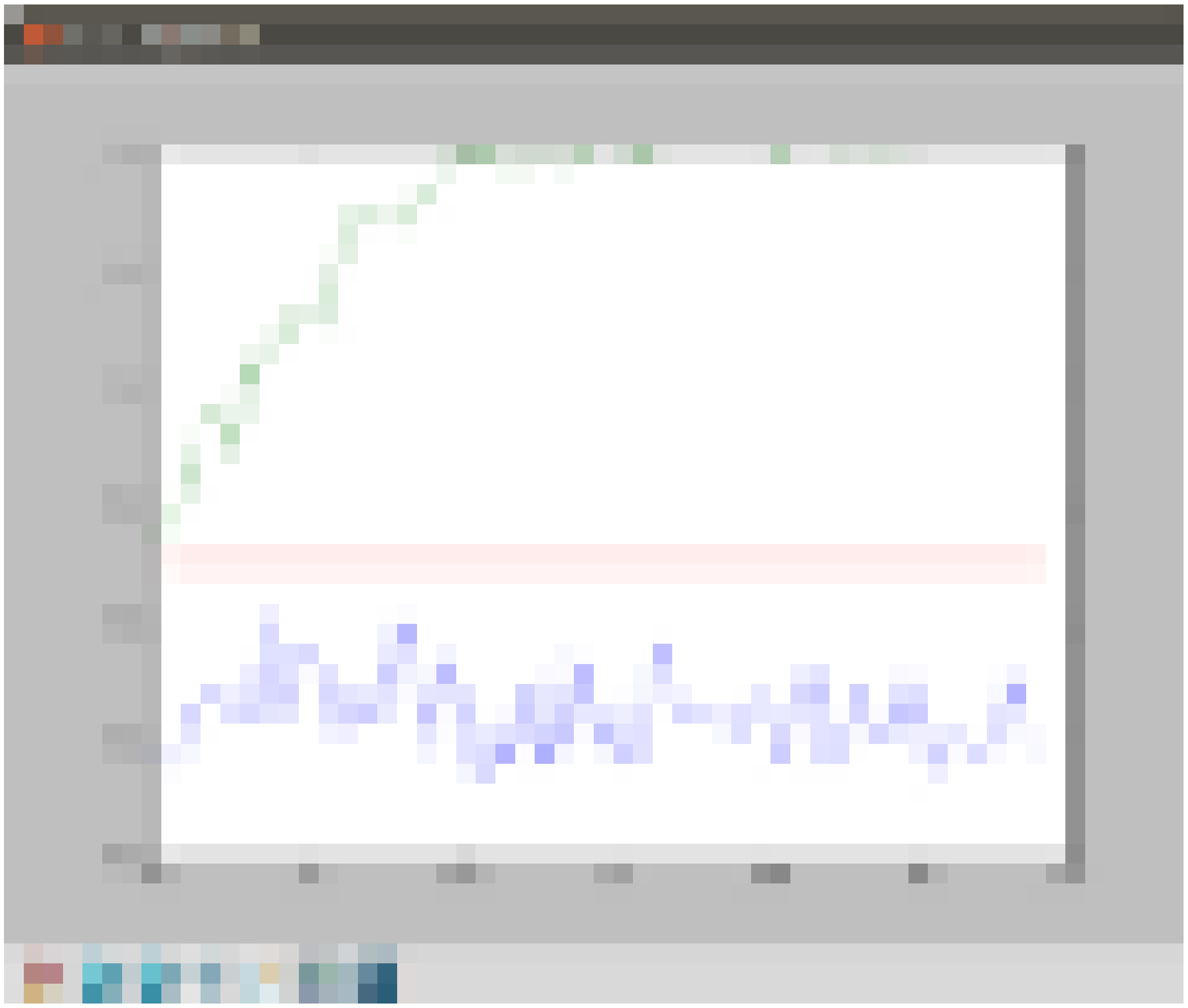
VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing



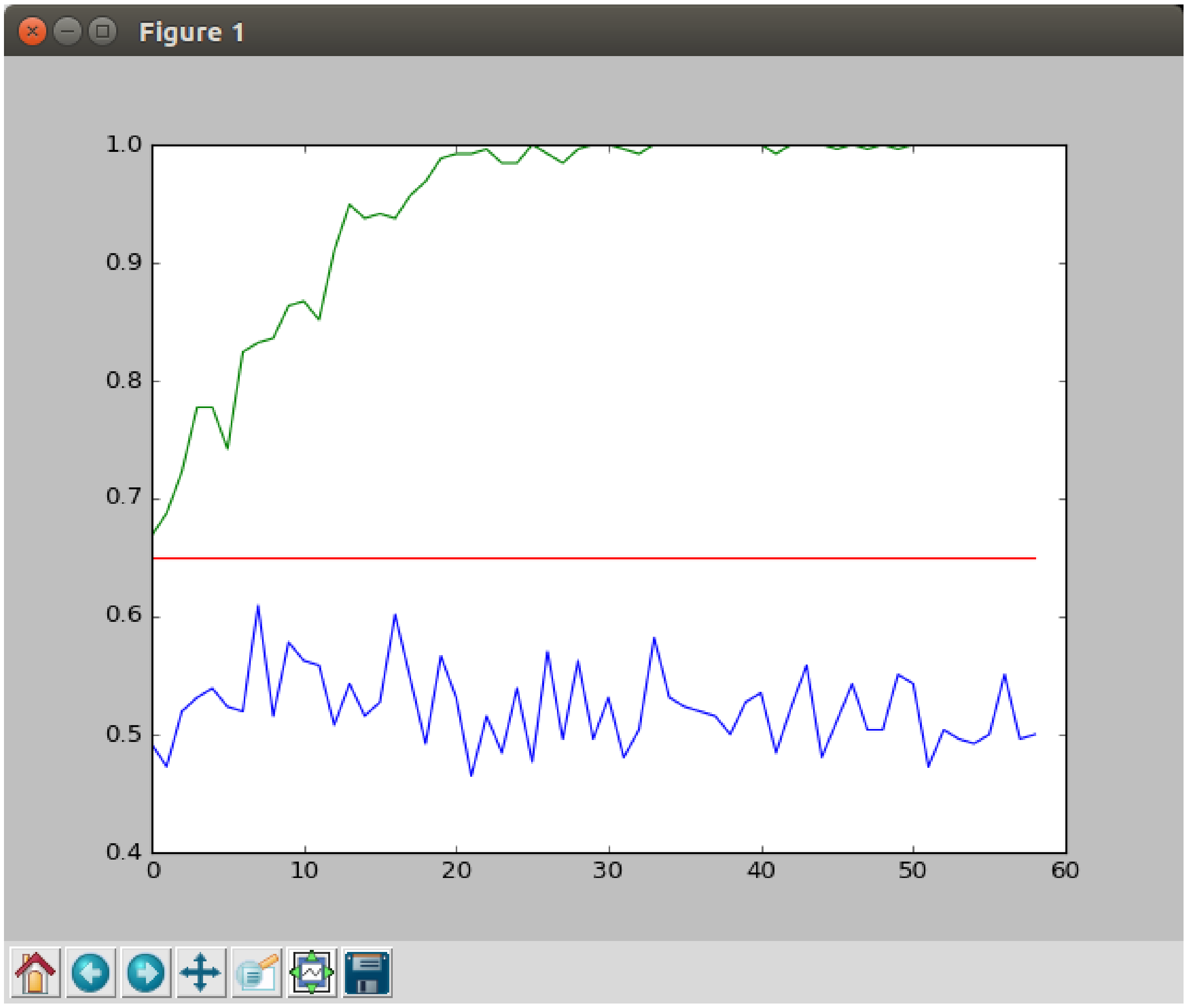
Another method that beats the CNN-only benchmark! But not nearly as impressively as the RNN did. My gut tells me there's room for parameter tuning on this to do better.

For most of our methods, I'm only showing results for our top performing network. However, with the MLP, something interesting happened when we tried deeper and wider networks: The top 1 accuracy floundered, but the top 5 accuracy went off the charts! Here's a four-layer, 2,048-wide MLP:

VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing



VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing



That is basically perfect top 5 classification accuracy. But I digress…

Final test accuracy: 70% top 1, 88% top 5

the Inception ConvNet to extract features followed by a single-layer LSTM RNN!

VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing

3.Proposed System Requirements Analysis and Design

3.1 Introduction
Product Perspective is A Video Retrievals system stores the following information. Database It stores the videos through which user can easily retrieve the real or original content through search engine. All categories of the video are stored in this database. It searches the video by using the concept of ANN or CNN using Python program.

3.2 Requirement Analysis
User Classes and Characteristics Users of the system should be able to get the real content, which they have searched through search engine from the database. Only the users those who can type can use this. Disabled people like the one not having hands cannot use because in this database we can only retrieve the data by using search engine.
Software: Windows 10
Hardware: PC
Platform: Python

3.2.1. Stakeholder Identification
Aniket kumar (18BCE0486)
Abhishek Shah(17BCE2394)
Shivam Sah(17BCE2386)

3.2.2. Functional Requirements
User Classes and Characteristics Users of the system should be able to get the real content, which they have searched through search engine from the database. Only the users those who can type can use this

3.2.3. Non Functional Requirements
Any computer device to access the program , Disabled people like the one not having hands cannot use because in this database we can only retrieve the data by using search engine.

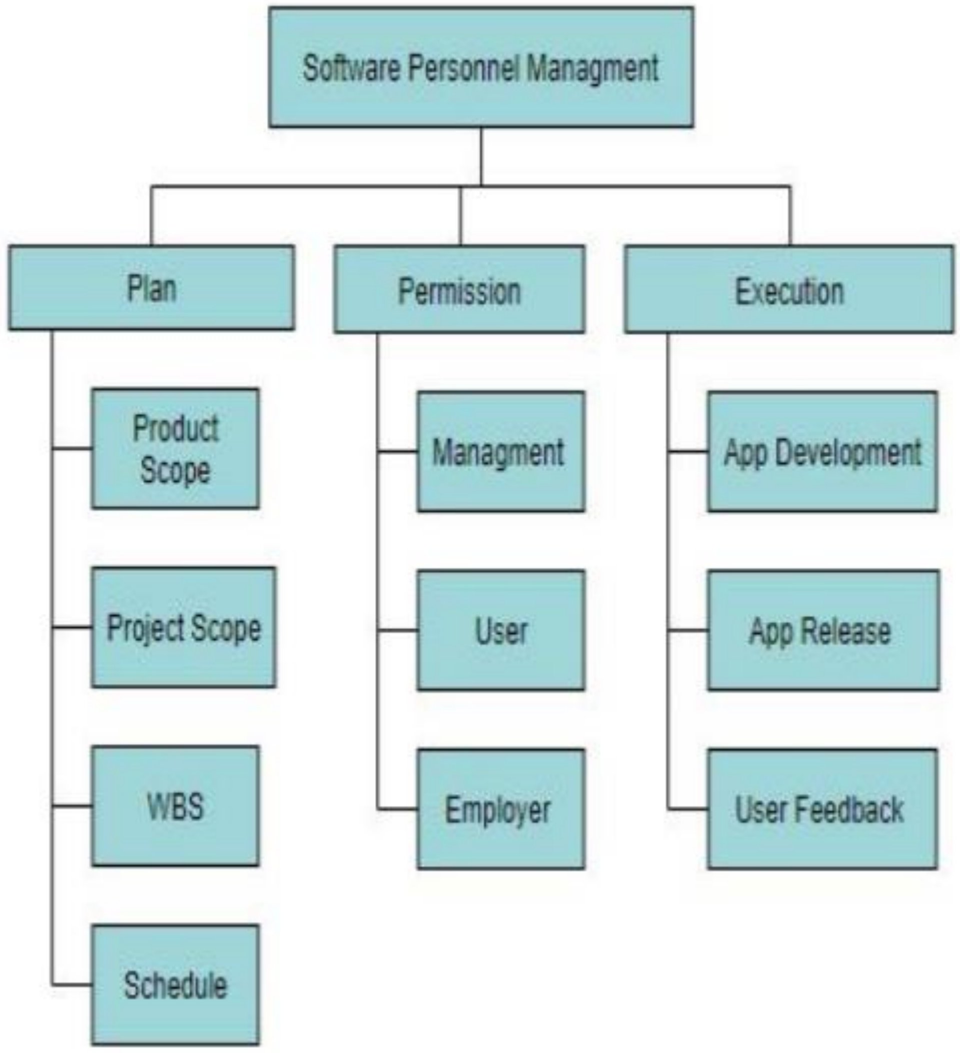
3.2.4. System Requirements
Software: Windows 10
Hardware: PC
Platform: Python

3.2.5 Software requirement specification
Platform: Python, Python Gui,Tensorflow
Software: Windows 10
Hardware: PC

VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing

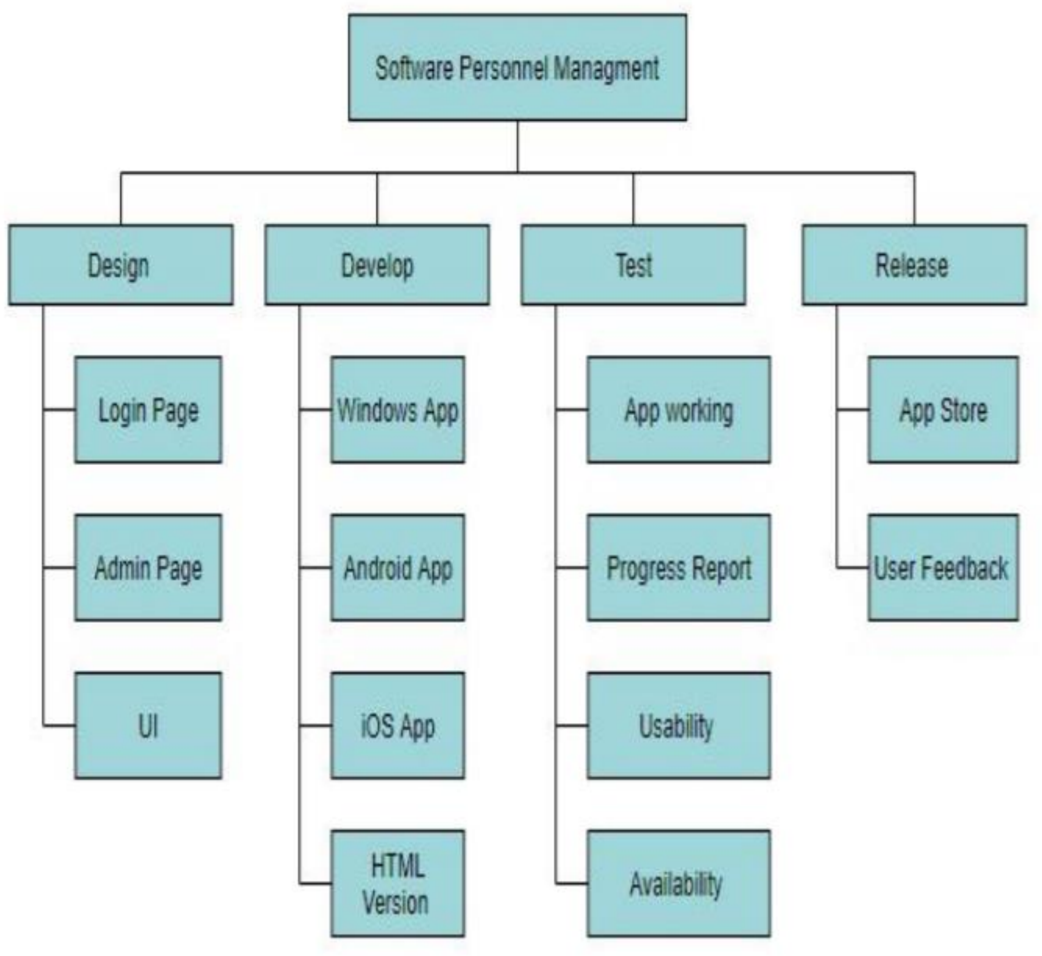
3.3 Work breakdown structure

3.3.1 NOUN ORIENTED WBS



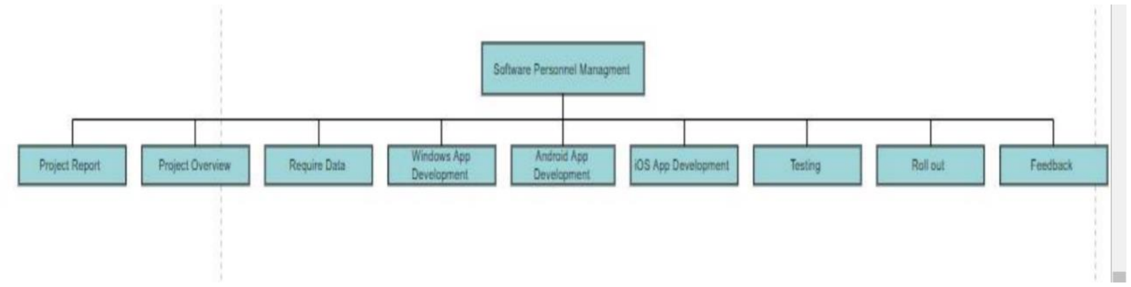
VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing

3.3.2 VERB ORIENTED WBS



VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing

3.3.3. TIME ORIENTED WBS



4. Design of the Proposed System

4.1. Introduction

The proposed approach can be used to classify hand shape images using videos as input data. A Convolutional Neural Network model has been developed which classifies the hand shape images into categories whether the hand is in fully open state, fully closed state or partially open. For training any CNN model thousands of images are needed and the same for testing and it requires lot of hard work. The proposed model tries to reduce the work by collecting videos instead of images. Since, videos are made by the combination of thousands of frames (images) and therefore can be used as an alternative. At the time of training a CNN model videos can be converted into frames and these generated frames can be used as an input for the model. By using this technique any CNN model can be trained effectively and efficiently just by using few numbers of videos.

4.2. High level Design (Framework, Architecture or Module for the Proposed System(with explanation))

The proposed model consists of three stages. In the first stage, videos of hand gesture are recorded as for training and testing the CNN model. The video count can be kept low as one video of 15sec length can generate approximately 15k images if extracted at every 1millisecond. In the second stage, some Image Pre-processing techniques are applied on the extracted frames and make them ready to be fed inside CNN model. In the third stage, the predictions are made and the accuracy graphs are plotted by using the testing data.

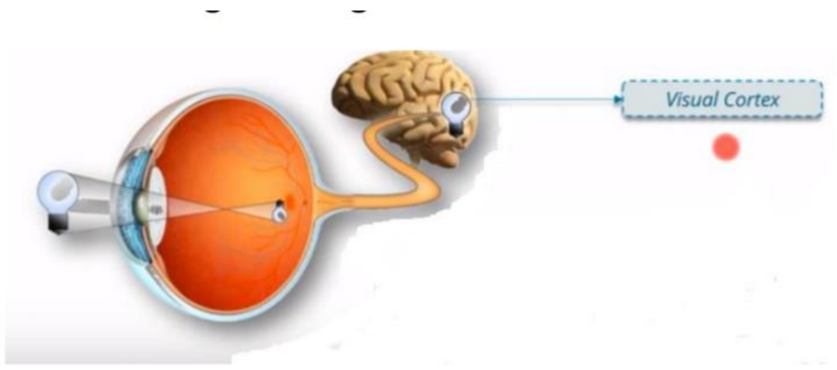
Algorithm:

```
 1: Start.
 2: Collect the videos for the train and the test data.
 3: Extract the frames form the videos.
 4: Design the CNN model.
5: Use training data to train the CNN model.
6: Use test data for testing and making predictions.
7: Stop.
```

4.2.1.Deep Convolutional Neural Network Model:

Convolutional Neural Network (CNN, or Conv-Net) is a type of feed-forward artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex. The visual cortex has small regions of cells that are sensitive to specific regions of the visual field. Some individual neuronal cells in the brain respond (or fires) only in the presence of edges of a certain orientation. For example, some neurons fires when exposed to vertical edges and some when shown horizontal or diagonal edges.

VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing

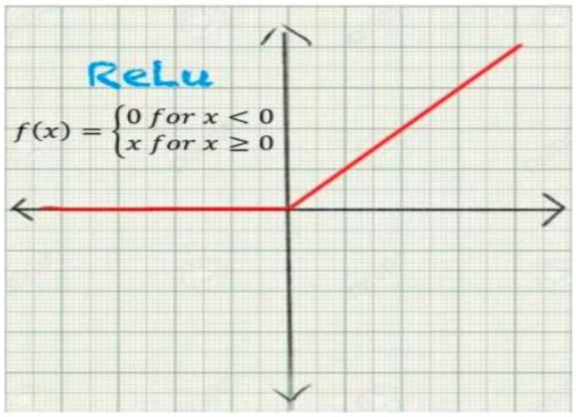


CNN compares the images pieces by pieces. The pieces that it looks for are called features. By finding rough features matches, in roughly the same position in two images, CNN gets a lot better at seeing similarity than whole-image matching schemas. We choose a feature and put it on the input image, if it matches then the images are classified correctly

A Typical CNN consists of four layers:

1). Convolution layer:
Here the features/filters are moved to every possible position of the image. Steps involved in this layer are:
Step1: Line up the feature and the image.
Step2: Multiply each image pixel by the corresponding feature pixel.
Step3: Add then up.
Step4: Divide the total number of pixels in the feature. Now to keep track of where that feature was, a map is created and the values of the filter are mapped. Similarly, the features are moved to every other positions of the image and detect how the feature matches that part of image. A similar procedure is performed with every other filter.

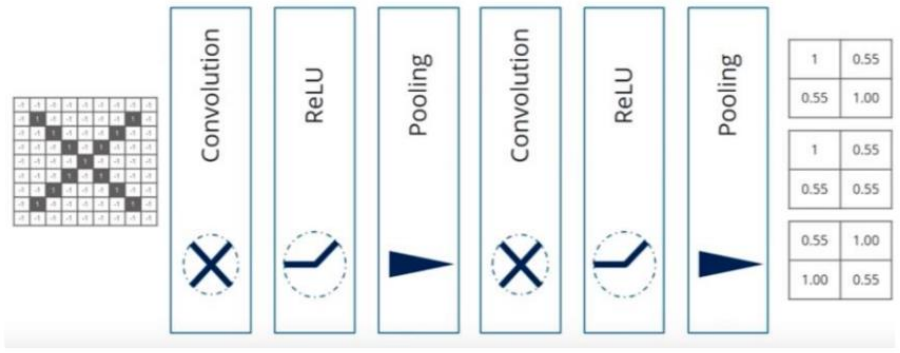
2). ReLU layer:
In this layer all the negative values are removed from the filtered images and are replaced with zeros. This is done to avoid the values from summing up to zero. Rectified Linear Unit (ReLU) transform function only activates a node if the input is above a certain quantity while the input is below zero, the output is zero, but when the input rises above certain threshold it has a linear relationship with the dependent variable



3). Pooling:

VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing

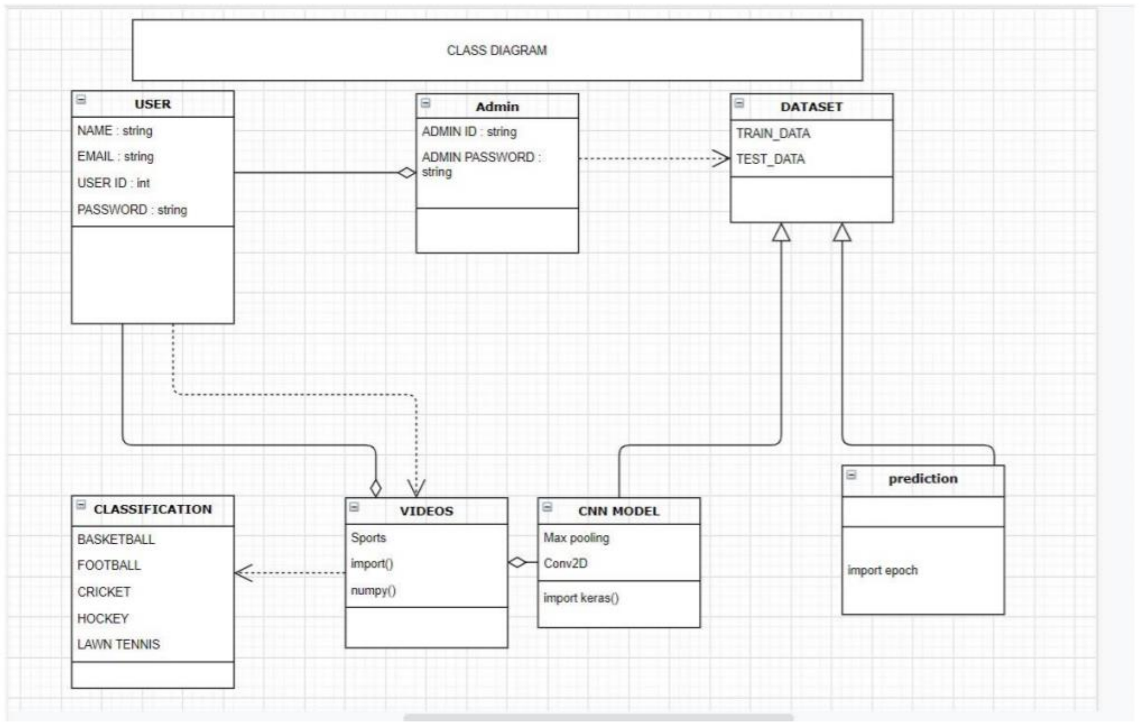
In this layer image stack are made to shrink into a smaller size.
Steps involved in this layer:
Step1: Pick a window of size (usually 2 or 3).
Step2: Pick a stride (usually 2).
Step3: Walk the window across filtered images.
Step4: From each window, take the maximum value



4). Fully Connected:
This is final layer where the actual classification happens. Here the filtered images are put in a single list and are made ready to classify newly inputted images.

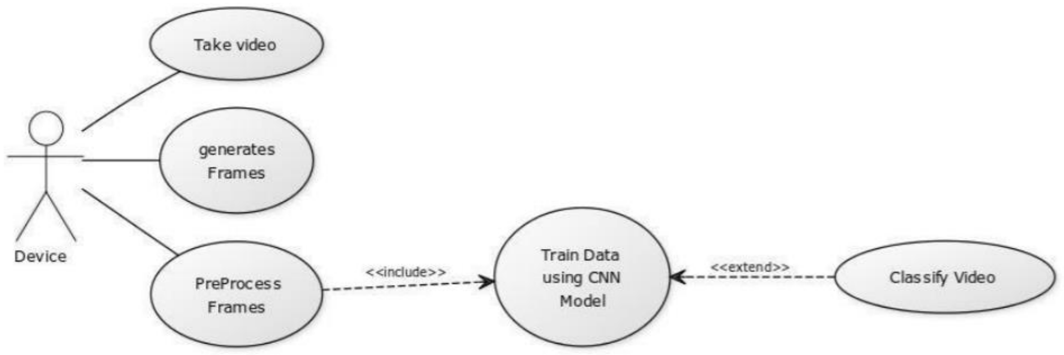
4.3. Detailed Design (ER Diagram/UML Diagram/Mathematical Modeling)

CLASS DIAGRAM:

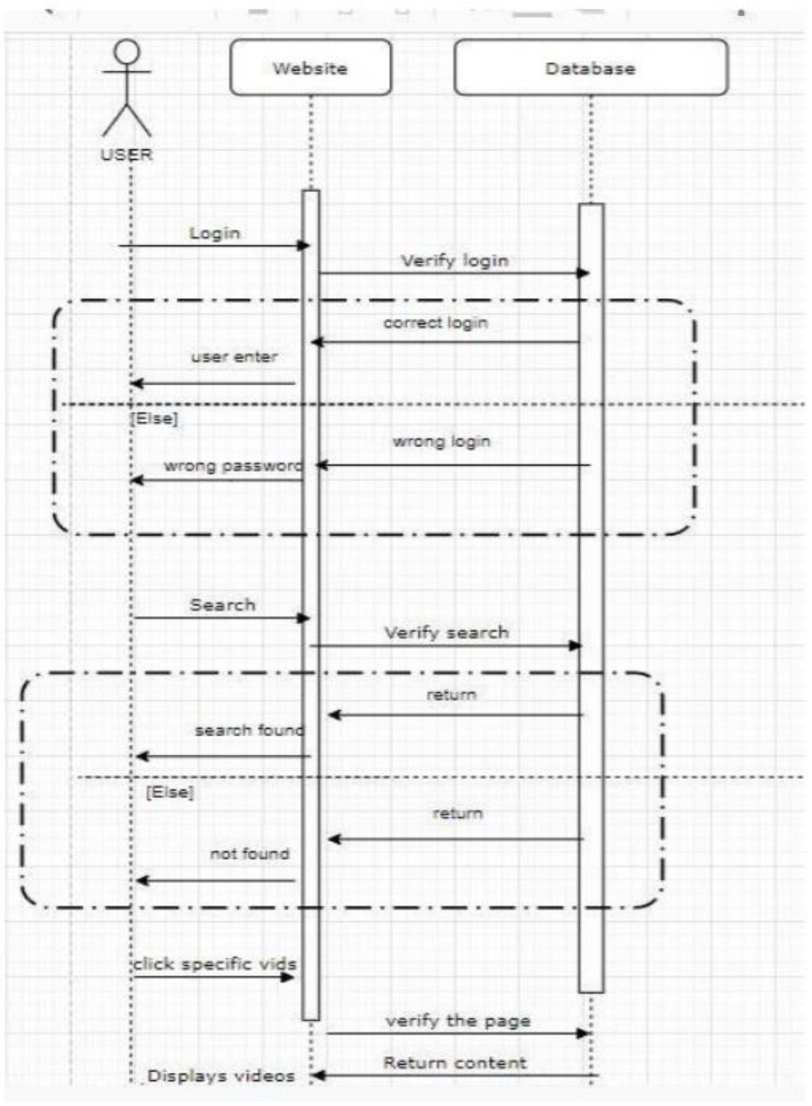


VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing

USE CASE DIAGRAM:

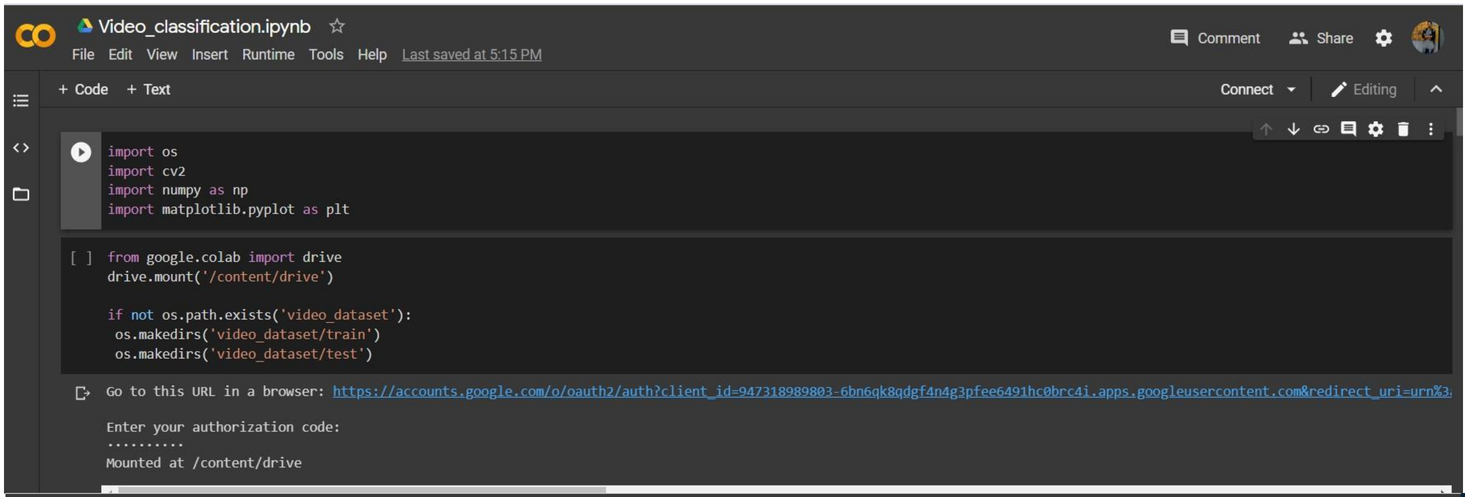


SEQUENCE DIAGRAM:



VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing

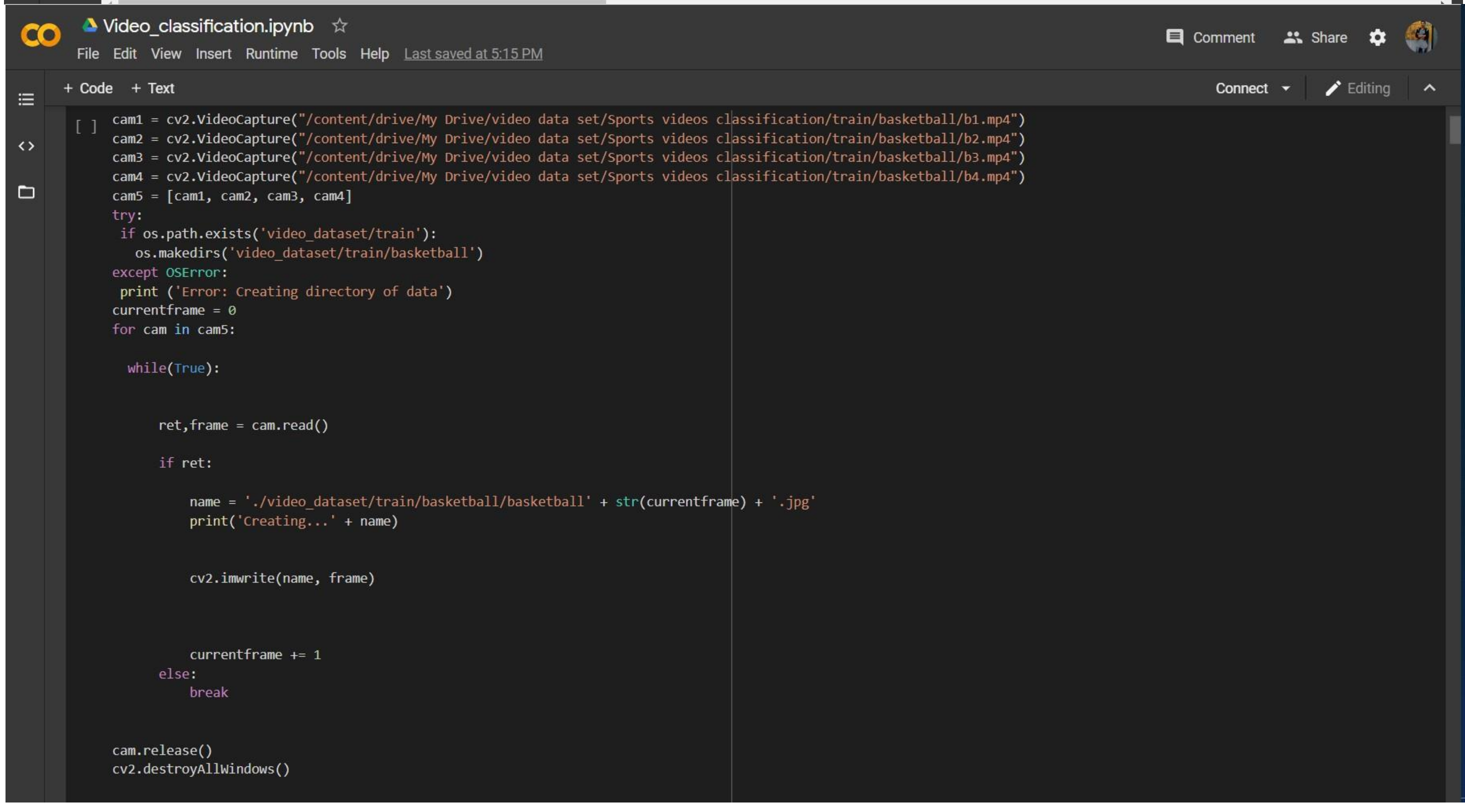
5.CODE IMPLEMENTATION AND TESTING


Video_classification.ipynb
File Edit View Insert Runtime Tools Help Last saved at 5:15 PM
Comment Share
+ Code + Text
Connect Editing

```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

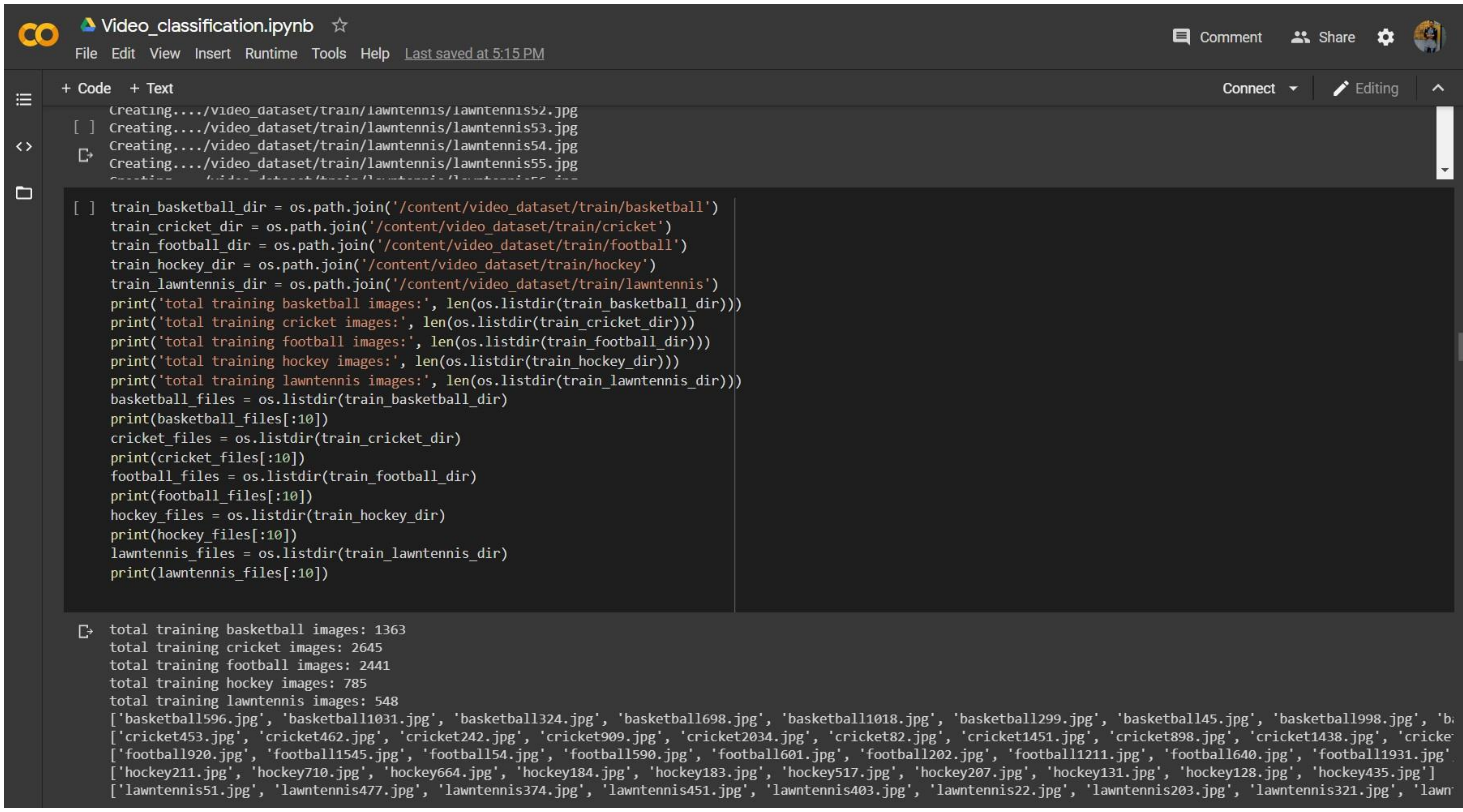
```
from google.colab import drive
drive.mount('/content/drive')

if not os.path.exists('video_dataset'):
 os.makedirs('video_dataset/train')
 os.makedirs('video_dataset/test')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3

Enter your authorization code:
··········
Mounted at /content/drive
```


Video_classification.ipynb
File Edit View Insert Runtime Tools Help Last saved at 5:15 PM
Comment Share
+ Code + Text
Connect Editing

```
cam1 = cv2.VideoCapture("/content/drive/My Drive/video data set/Sports videos classification/train/basketball/b1.mp4")
cam2 = cv2.VideoCapture("/content/drive/My Drive/video data set/Sports videos classification/train/basketball/b2.mp4")
cam3 = cv2.VideoCapture("/content/drive/My Drive/video data set/Sports videos classification/train/basketball/b3.mp4")
cam4 = cv2.VideoCapture("/content/drive/My Drive/video data set/Sports videos classification/train/basketball/b4.mp4")
cam5 = [cam1, cam2, cam3, cam4]
try:
 if os.path.exists('video_dataset/train'):
   os.makedirs('video_dataset/train/basketball')
except OSError:
 print ('Error: Creating directory of data')
currentframe = 0
for cam in cam5:

  while(True):


      ret,frame = cam.read()

      if ret:

          name = './video_dataset/train/basketball/basketball' + str(currentframe) + '.jpg'
          print('Creating...' + name)


          cv2.imwrite(name, frame)



          currentframe += 1
      else:
          break


cam.release()
cv2.destroyAllWindows()
```

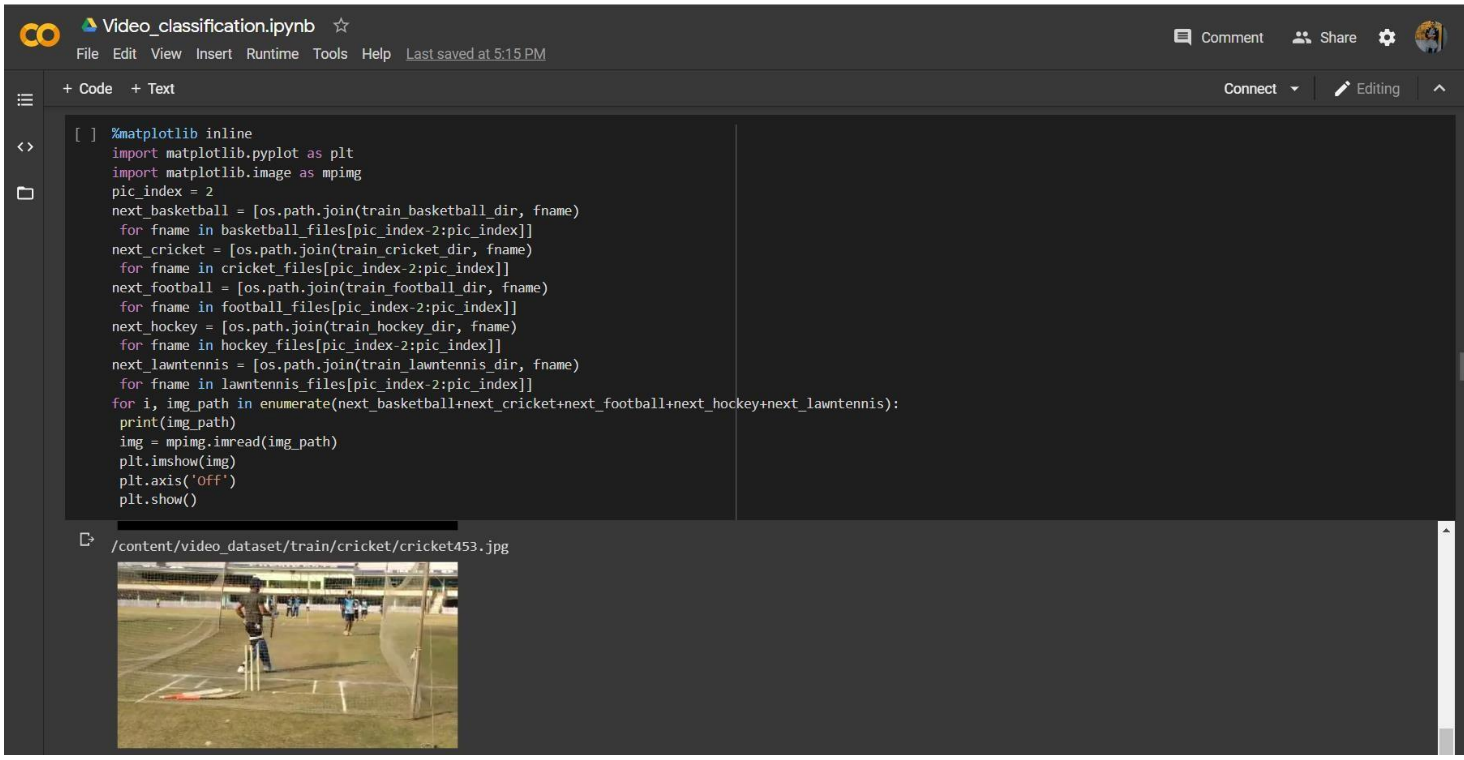
VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing


Video_classification.ipynb
File Edit View Insert Runtime Tools Help Last saved at 5:15 PM
Comment Share
+ Code + Text
Connect Editing

```
Creating..../video_dataset/train/lawntennis/lawntennis52.jpg
Creating..../video_dataset/train/lawntennis/lawntennis53.jpg
Creating..../video_dataset/train/lawntennis/lawntennis54.jpg
Creating..../video_dataset/train/lawntennis/lawntennis55.jpg
```

```
train_basketball_dir = os.path.join('/content/video_dataset/train/basketball')
train_cricket_dir = os.path.join('/content/video_dataset/train/cricket')
train_football_dir = os.path.join('/content/video_dataset/train/football')
train_hockey_dir = os.path.join('/content/video_dataset/train/hockey')
train_lawntennis_dir = os.path.join('/content/video_dataset/train/lawntennis')
print('total training basketball images:', len(os.listdir(train_basketball_dir)))
print('total training cricket images:', len(os.listdir(train_cricket_dir)))
print('total training football images:', len(os.listdir(train_football_dir)))
print('total training hockey images:', len(os.listdir(train_hockey_dir)))
print('total training lawntennis images:', len(os.listdir(train_lawntennis_dir)))
basketball_files = os.listdir(train_basketball_dir)
print(basketball_files[:10])
cricket_files = os.listdir(train_cricket_dir)
print(cricket_files[:10])
football_files = os.listdir(train_football_dir)
print(football_files[:10])
hockey_files = os.listdir(train_hockey_dir)
print(hockey_files[:10])
lawntennis_files = os.listdir(train_lawntennis_dir)
print(lawntennis_files[:10])
```

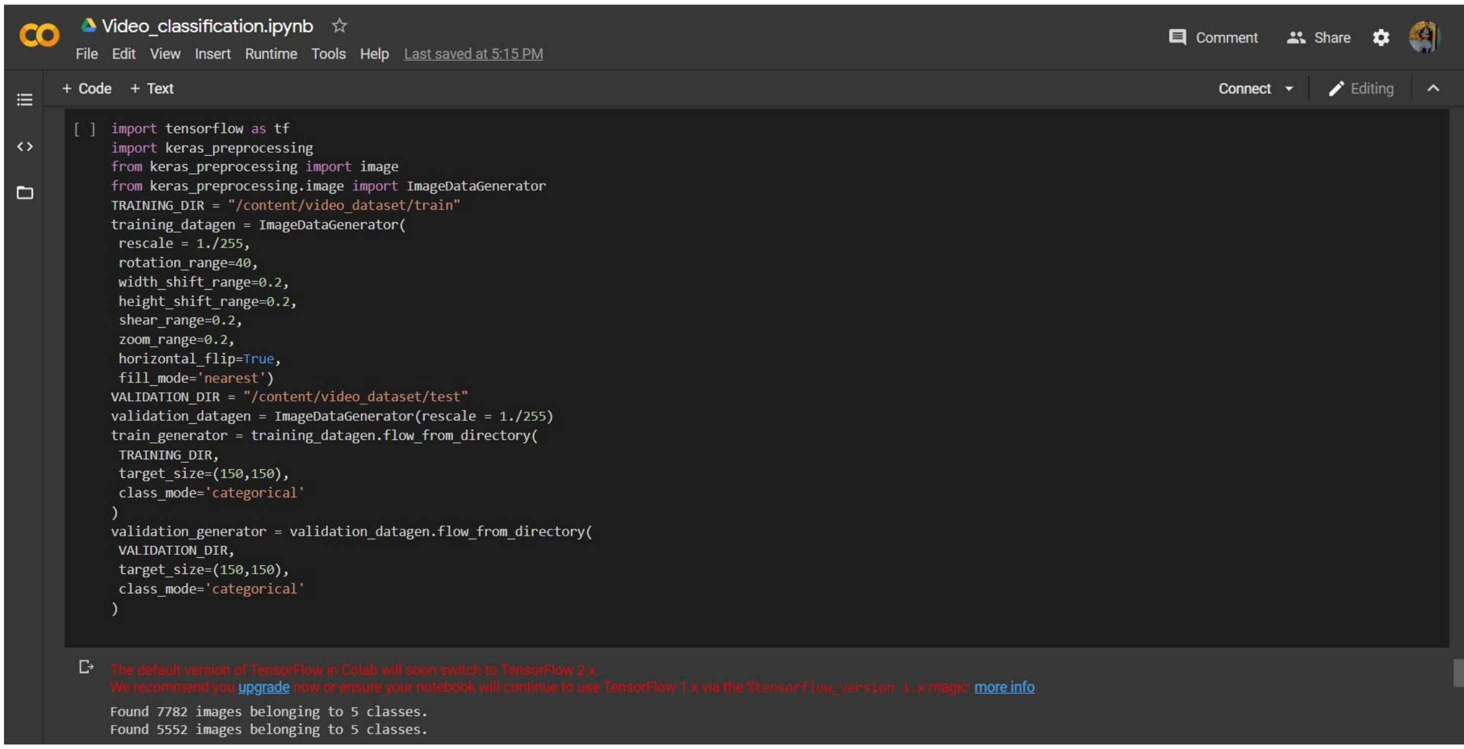
```
total training basketball images: 1363
total training cricket images: 2645
total training football images: 2441
total training hockey images: 785
total training lawntennis images: 548
['basketball596.jpg', 'basketball1031.jpg', 'basketball324.jpg', 'basketball698.jpg', 'basketball1018.jpg', 'basketball299.jpg', 'basketball45.jpg', 'basketball998.jpg', 'ba
['cricket453.jpg', 'cricket462.jpg', 'cricket242.jpg', 'cricket909.jpg', 'cricket2034.jpg', 'cricket82.jpg', 'cricket1451.jpg', 'cricket898.jpg', 'cricket1438.jpg', 'cricke
['football920.jpg', 'football1545.jpg', 'football54.jpg', 'football590.jpg', 'football601.jpg', 'football202.jpg', 'football1211.jpg', 'football640.jpg', 'football1931.jpg'
['hockey211.jpg', 'hockey710.jpg', 'hockey664.jpg', 'hockey184.jpg', 'hockey183.jpg', 'hockey517.jpg', 'hockey207.jpg', 'hockey131.jpg', 'hockey128.jpg', 'hockey435.jpg']
['lawntennis51.jpg', 'lawntennis477.jpg', 'lawntennis374.jpg', 'lawntennis451.jpg', 'lawntennis403.jpg', 'lawntennis22.jpg', 'lawntennis203.jpg', 'lawntennis321.jpg', 'lawn
```


Video_classification.ipynb
File Edit View Insert Runtime Tools Help Last saved at 5:15 PM
Comment Share
+ Code + Text
Connect Editing

```
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
pic_index = 2
next_basketball = [os.path.join(train_basketball_dir, fname)
 for fname in basketball_files[pic_index-2:pic_index]]
next_cricket = [os.path.join(train_cricket_dir, fname)
 for fname in cricket_files[pic_index-2:pic_index]]
next_football = [os.path.join(train_football_dir, fname)
 for fname in football_files[pic_index-2:pic_index]]
next_hockey = [os.path.join(train_hockey_dir, fname)
 for fname in hockey_files[pic_index-2:pic_index]]
next_lawntennis = [os.path.join(train_lawntennis_dir, fname)
 for fname in lawntennis_files[pic_index-2:pic_index]]
for i, img_path in enumerate(next_basketball+next_cricket+next_football+next_hockey+next_lawntennis):
 print(img_path)
 img = mpimg.imread(img_path)
 plt.imshow(img)
 plt.axis('Off')
 plt.show()
```

```
/content/video_dataset/train/cricket/cricket453.jpg
```

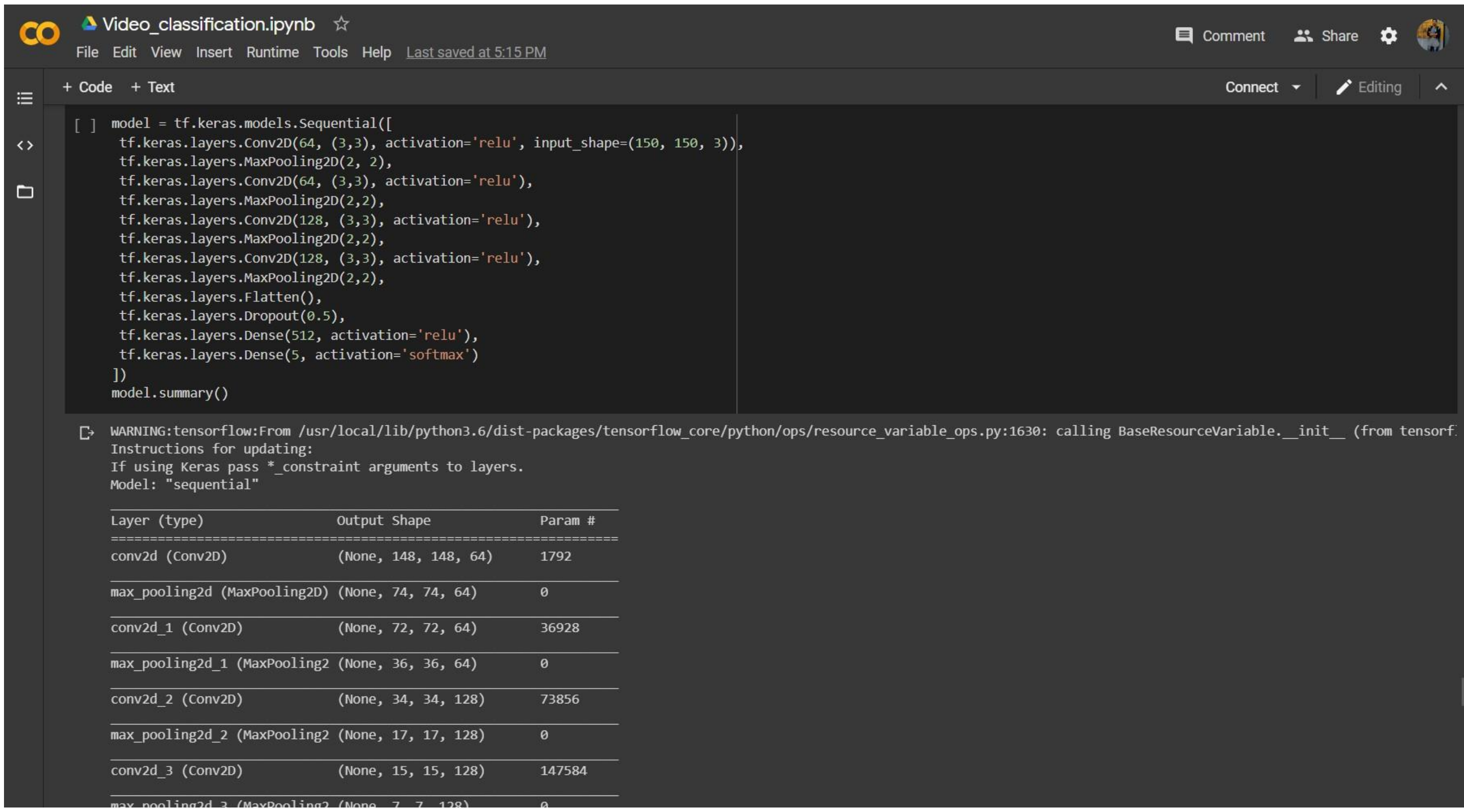
VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing



Video_classification.ipynb
File Edit View Insert Runtime Tools Help Last saved at 5:15 PM
Comment Share
+ Code + Text
Connect Editing

```
import tensorflow as tf
import keras_preprocessing
from keras_preprocessing import image
from keras_preprocessing.image import ImageDataGenerator
TRAINING_DIR = "/content/video_dataset/train"
training_datagen = ImageDataGenerator(
 rescale = 1./255,
 rotation_range=40,
 width_shift_range=0.2,
 height_shift_range=0.2,
 shear_range=0.2,
 zoom_range=0.2,
 horizontal_flip=True,
 fill_mode='nearest')
VALIDATION_DIR = "/content/video_dataset/test"
validation_datagen = ImageDataGenerator(rescale = 1./255)
train_generator = training_datagen.flow_from_directory(
 TRAINING_DIR,
 target_size=(150,150),
 class_mode='categorical'
)
validation_generator = validation_datagen.flow_from_directory(
 VALIDATION_DIR,
 target_size=(150,150),
 class_mode='categorical'
)
```

```
The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the %tensorflow_version 1.x magic: more info
Found 7782 images belonging to 5 classes.
Found 5552 images belonging to 5 classes.
```

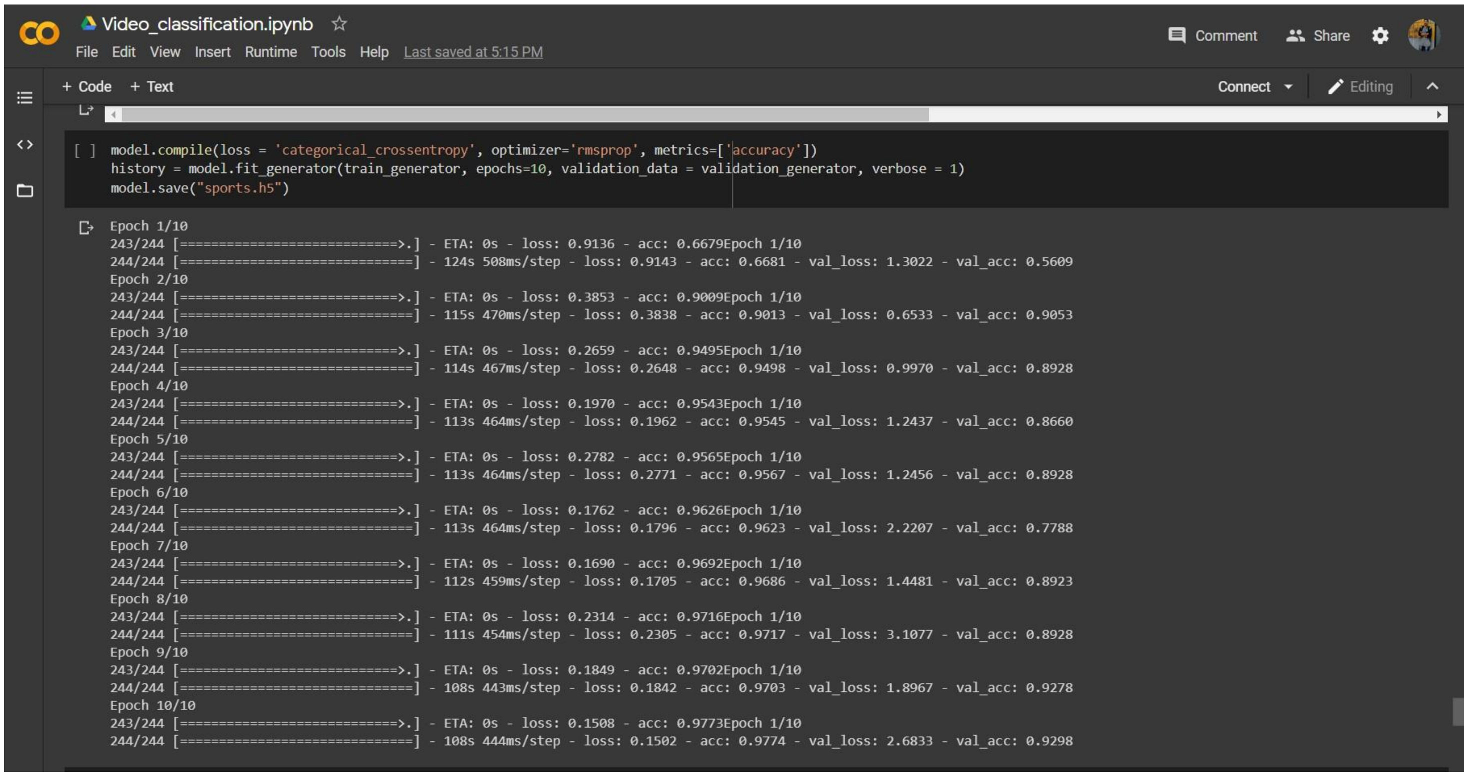


Video_classification.ipynb
File Edit View Insert Runtime Tools Help Last saved at 5:15 PM
Comment Share
+ Code + Text
Connect Editing

```
model = tf.keras.models.Sequential([
 tf.keras.layers.Conv2D(64, (3,3), activation='relu', input_shape=(150, 150, 3)),
 tf.keras.layers.MaxPooling2D(2, 2),
 tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
 tf.keras.layers.MaxPooling2D(2,2),
 tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
 tf.keras.layers.MaxPooling2D(2,2),
 tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
 tf.keras.layers.MaxPooling2D(2,2),
 tf.keras.layers.Flatten(),
 tf.keras.layers.Dropout(0.5),
 tf.keras.layers.Dense(512, activation='relu'),
 tf.keras.layers.Dense(5, activation='softmax')
])
model.summary()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/ops/resource_variable_ops.py:1630: calling BaseResourceVariable.__init__ (from tensorf
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 64)      1792
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 128)         0
```

VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing

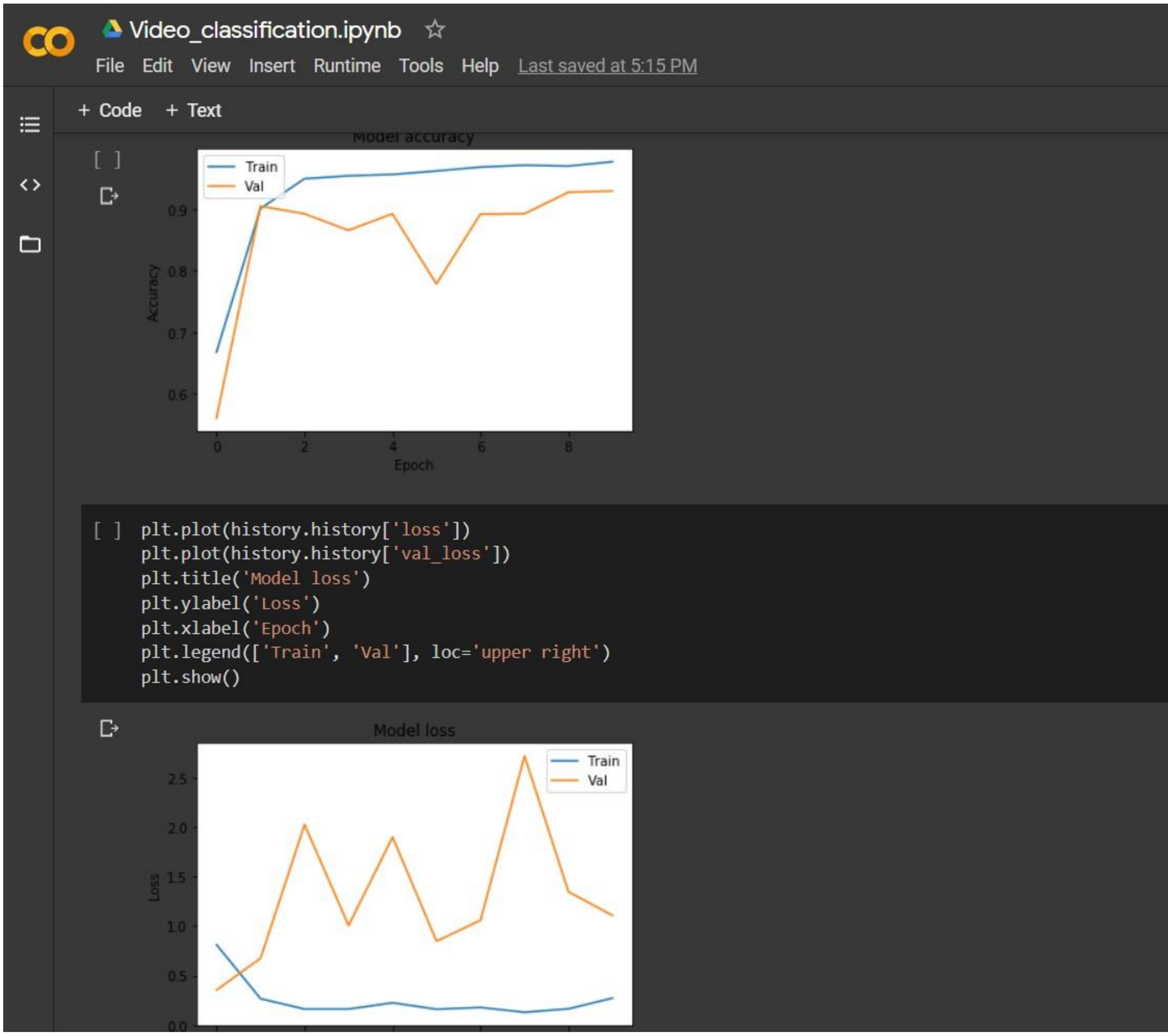


Video_classification.ipynb
File Edit View Insert Runtime Tools Help Last saved at 5:15 PM
Comment Share
+ Code + Text
Connect Editing

```
model.compile(loss = 'categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
history = model.fit_generator(train_generator, epochs=10, validation_data = validation_generator, verbose = 1)
model.save("sports.h5")
```

```
Epoch 1/10
243/244 [===========================>.] - ETA: 0s - loss: 0.9136 - acc: 0.6679Epoch 1/10
244/244 [==============================] - 124s 508ms/step - loss: 0.9143 - acc: 0.6681 - val_loss: 1.3022 - val_acc: 0.5609
Epoch 2/10
243/244 [===========================>.] - ETA: 0s - loss: 0.3853 - acc: 0.9009Epoch 1/10
244/244 [==============================] - 115s 470ms/step - loss: 0.3838 - acc: 0.9013 - val_loss: 0.6533 - val_acc: 0.9053
Epoch 3/10
243/244 [===========================>.] - ETA: 0s - loss: 0.2659 - acc: 0.9495Epoch 1/10
244/244 [==============================] - 114s 467ms/step - loss: 0.2648 - acc: 0.9498 - val_loss: 0.9970 - val_acc: 0.8928
Epoch 4/10
243/244 [===========================>.] - ETA: 0s - loss: 0.1970 - acc: 0.9543Epoch 1/10
244/244 [==============================] - 113s 464ms/step - loss: 0.1962 - acc: 0.9545 - val_loss: 1.2437 - val_acc: 0.8660
Epoch 5/10
243/244 [===========================>.] - ETA: 0s - loss: 0.2782 - acc: 0.9565Epoch 1/10
244/244 [==============================] - 113s 464ms/step - loss: 0.2771 - acc: 0.9567 - val_loss: 1.2456 - val_acc: 0.8928
Epoch 6/10
243/244 [===========================>.] - ETA: 0s - loss: 0.1762 - acc: 0.9626Epoch 1/10
244/244 [==============================] - 113s 464ms/step - loss: 0.1796 - acc: 0.9623 - val_loss: 2.2207 - val_acc: 0.7788
Epoch 7/10
243/244 [===========================>.] - ETA: 0s - loss: 0.1690 - acc: 0.9692Epoch 1/10
244/244 [==============================] - 112s 459ms/step - loss: 0.1705 - acc: 0.9686 - val_loss: 1.4481 - val_acc: 0.8923
Epoch 8/10
243/244 [===========================>.] - ETA: 0s - loss: 0.2314 - acc: 0.9716Epoch 1/10
244/244 [==============================] - 111s 454ms/step - loss: 0.2305 - acc: 0.9717 - val_loss: 3.1077 - val_acc: 0.8928
Epoch 9/10
243/244 [===========================>.] - ETA: 0s - loss: 0.1849 - acc: 0.9702Epoch 1/10
244/244 [==============================] - 108s 443ms/step - loss: 0.1842 - acc: 0.9703 - val_loss: 1.8967 - val_acc: 0.9278
Epoch 10/10
243/244 [===========================>.] - ETA: 0s - loss: 0.1508 - acc: 0.9773Epoch 1/10
244/244 [==============================] - 108s 444ms/step - loss: 0.1502 - acc: 0.9774 - val_loss: 2.6833 - val_acc: 0.9298
```

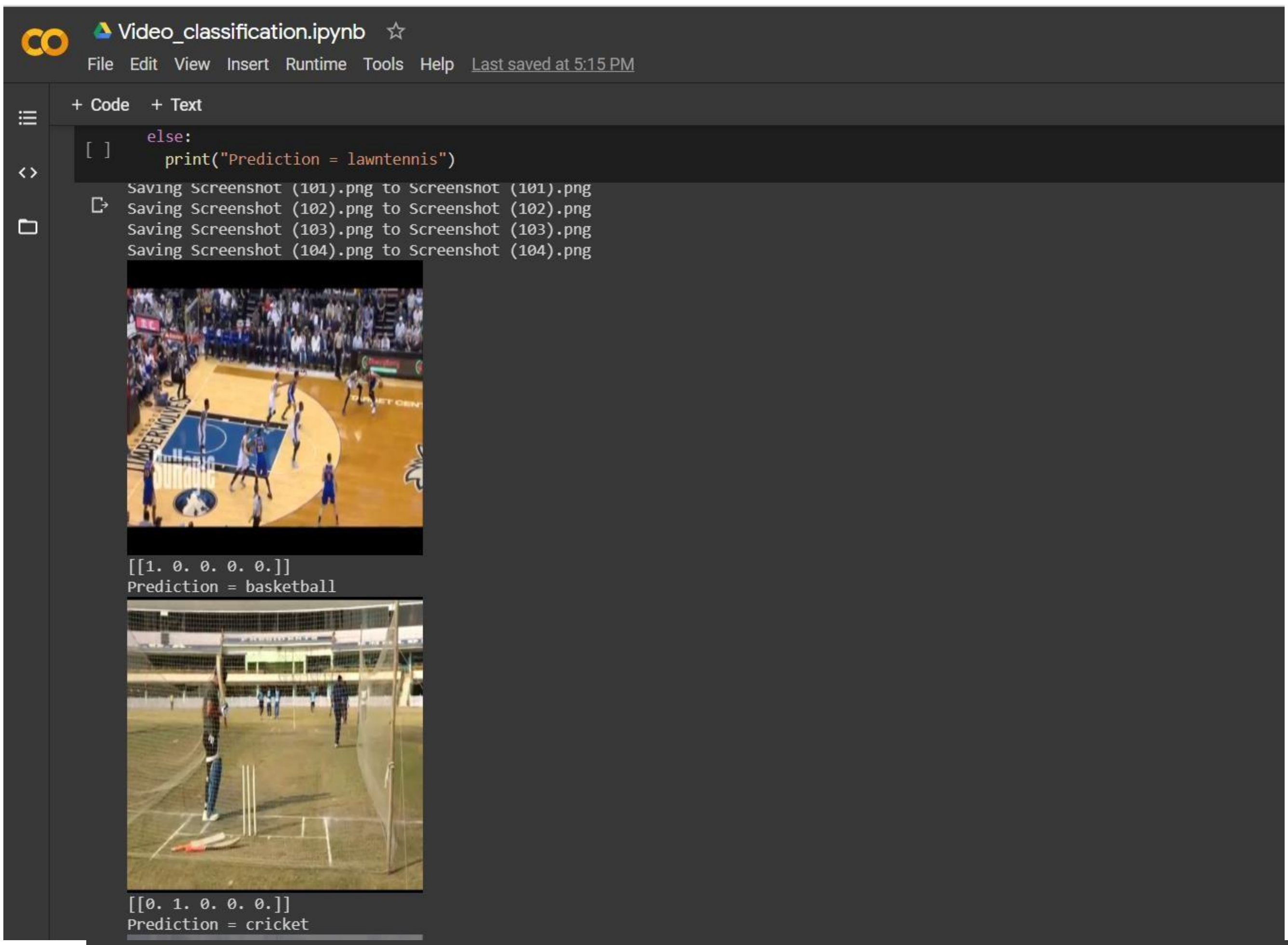
VIDEO LINK: https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing



```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```

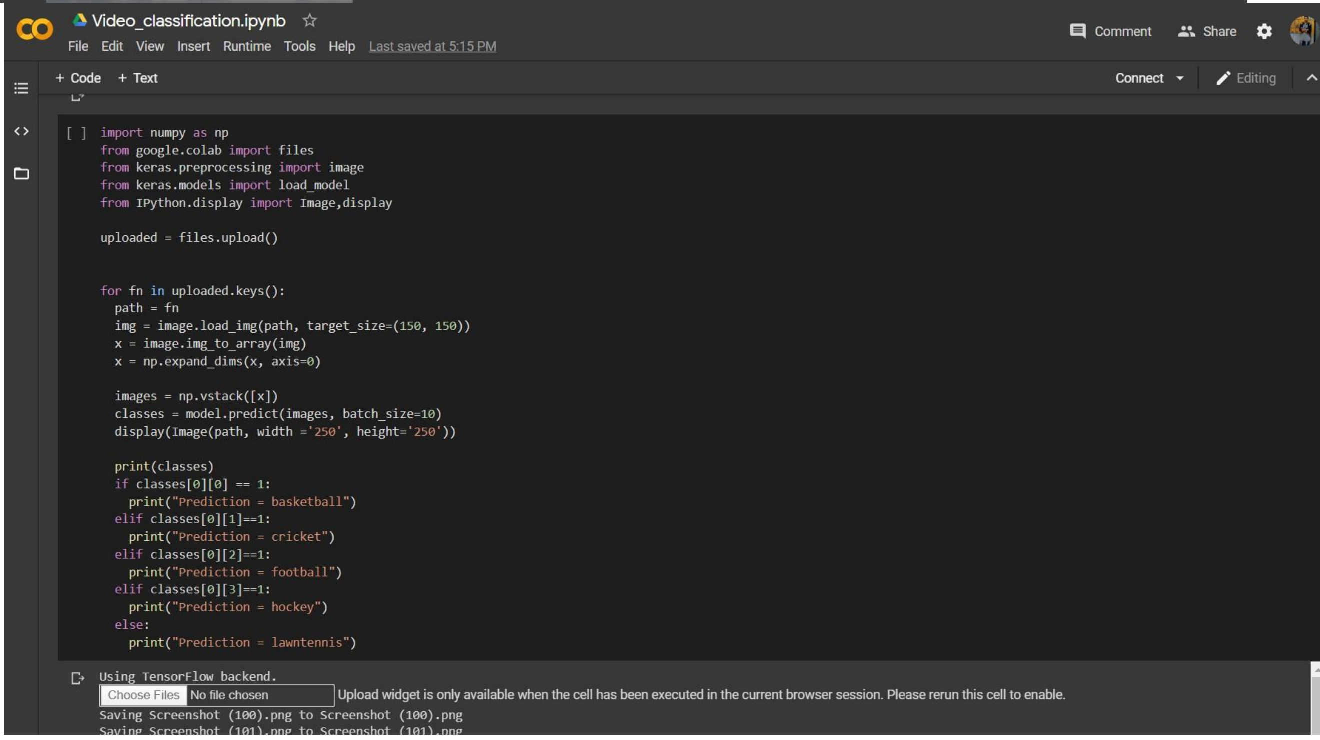
VIDEO LINK:
https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing

6.


Video_classification.ipynb
File Edit View Insert Runtime Tools Help Last saved at 5:15 PM
+ Code + Text

```
  else:
    print("Prediction = lawntennis")
```

```
Saving Screenshot (101).png to Screenshot (101).png
Saving Screenshot (102).png to Screenshot (102).png
Saving Screenshot (103).png to Screenshot (103).png
Saving Screenshot (104).png to Screenshot (104).png
[[1. 0. 0. 0. 0.]]
Prediction = basketball
[[0. 1. 0. 0. 0.]]
Prediction = cricket
```


Video_classification.ipynb
File Edit View Insert Runtime Tools Help Last saved at 5:15 PM
Comment Share
+ Code + Text
Connect Editing

```
import numpy as np
from google.colab import files
from keras.preprocessing import image
from keras.models import load_model
from IPython.display import Image,display

uploaded = files.upload()


for fn in uploaded.keys():
  path = fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=10)
  display(Image(path, width ='250', height='250'))

  print(classes)
  if classes[0][0] == 1:
    print("Prediction = basketball")
  elif classes[0][1]==1:
    print("Prediction = cricket")
  elif classes[0][2]==1:
    print("Prediction = football")
  elif classes[0][3]==1:
    print("Prediction = hockey")
  else:
    print("Prediction = lawntennis")
```

```
Using TensorFlow backend.
Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving Screenshot (100).png to Screenshot (100).png
Saving Screenshot (101).png to Screenshot (101).png
```

VIDEO LINK:
https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing

6.Conclusion, Limitations and Scope for future Work

In this project we have tried to perform video classification by converting videos to images and then performing image classification. In development of this project we have learned many concepts of machine learning like image preprocessing, working with numpy array, keras library, open cv, tensorflow, how to construct convolutional neural network, how to train a model, how to make predictions with the model and how to extract frames from the videos. This project can be used in many areas like in many mobile applications it can be used as a filter where any content uploaded on application will pass through this filter and it will confirm whether that content will be posted on the application or not. It can also be used in self driving car where it can help in finding the appropriate path for the vehicle to move safely.

The problems are that we still need to build and learn machine learnig models continusly to make it (video classification) efficient , we have used one of the best strategy as to take a video and change it to frames then evaluate .Video classification is tougher from image classification because here problem lies with the collection of dataset and training the model. The size of the video dataset is very large and also training the model with those dataset is a tough job as it consume time and not possible with normal daily use laptops with basic specs . disadvantage is also that the amount of operations of such a model is comparatively huge. It takes a lot of time not only to train a model, but also to use it. Unless you can speed the computation up using the future scope of improvement would be more efficient coding and decreasing the time complexity to build and train machine learning models

This project can be used in many areas like in many mobile applications it can be used as a filter where any content uploaded on application will pass through this filter and it will confirm whether that content will be posted on the application or not. It can also be used in self driving car where it can help in finding the appropriate path for the vehicle to move safely. Also can be used in content filtering which can hellp social website companies to be more ethical.

VIDEO LINK:
https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing

7.References:

1. https://medium.com/@tifa2up/image-classification-using-deep-neural-networks-a-beginnerfriendly-approach-using-tensorflow-94b0a090ccd4

2. https://www.sciencedirect.com/topics/computer-science/image-classification

3. https://www.analyticsvidhya.com/blog/2019/01/build-image-classification-model-10-minutes/

4. http://cs231n.github.io/classification/

5. https://www.tensorflow.org/lite/models/image_classification/overview

6. https://blog.coast.ai/five-video-classification-methods-implemented-in-keras-and-tensorflow-99cad29cc0b5#:~:text=The%20video%20classification%20methods&text=Classifying%20one%20frame%20at%20a,sequence%20to%20a%20separate%20RNN

7. https://www.eecs.wsu.edu/~cook/pubs/smc08.pdf

8. LINK OF VIDEO FOR THIS PROJECT IMPLEMENTATION
https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing

VIDEO LINK:
https://drive.google.com/file/d/10bSRQmrcnZL77egAHrF2vhq_qemyA3PU/view?usp=sharing