FabBits : A video cataloguer

A

Seminar Report

Submitted

In partial fulfillment

For the award of the Degree of

Bachelor of Technology

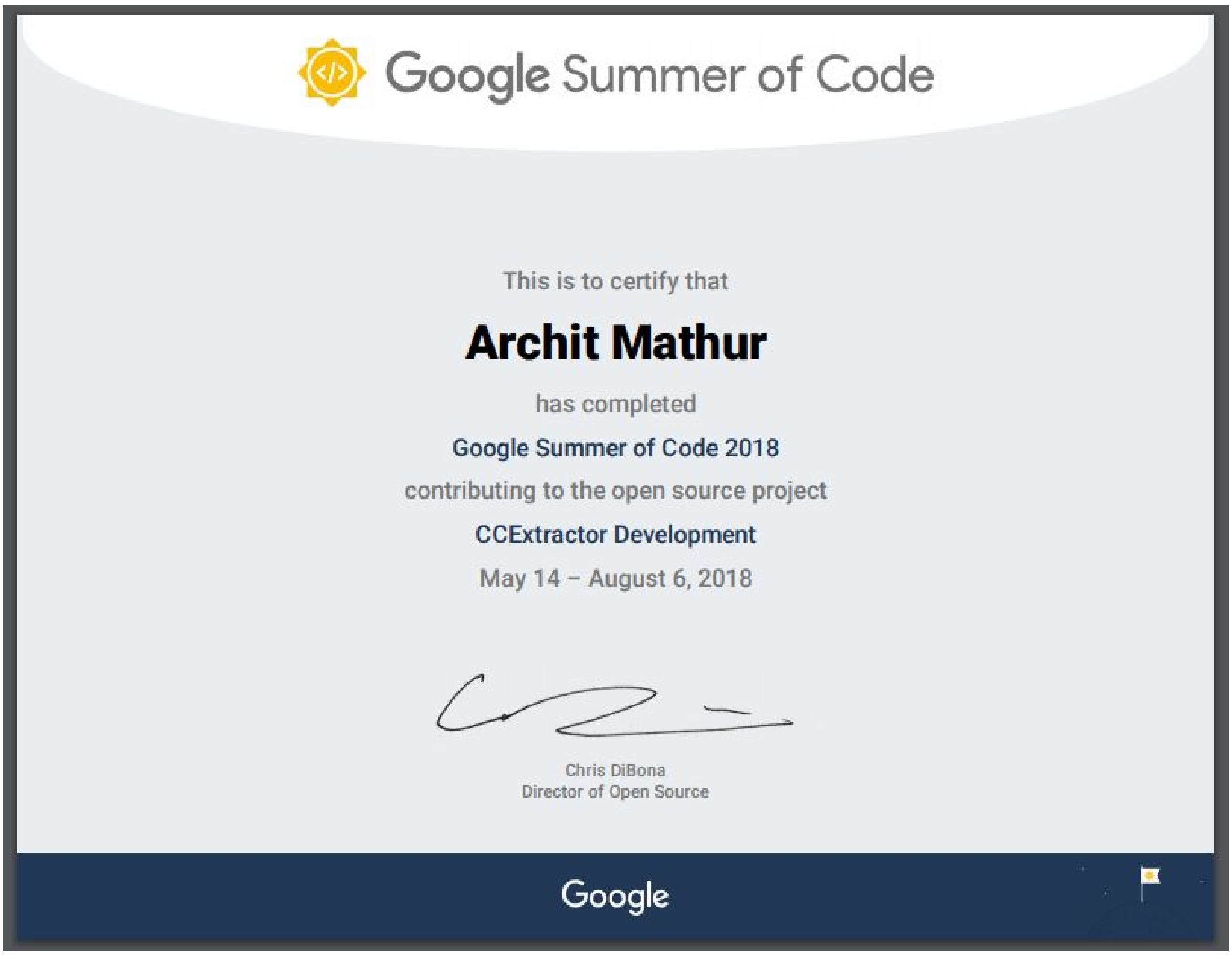
In Computer Science

Session 2018 - 19

Submitted to -	**Submitted by -**
Mr Sunil Dhankar	Archit Mathur
Associate Professor 1	15ESKCS033

Department of Computer Science
Swami Keshvanand Institute of Technology, Management & Gramothan
Rajasthan Technical University, Kota

October, 2018



Google Summer of Code

This is to certify that

Archit Mathur

has completed

Google Summer of Code 2018

contributing to the open source project

CCExtractor Development

May 14 – August 6, 2018

[illegible]

Chris DiBona
Director of Open Source

Google

Certificate of successful completion

DECLARATION

I hereby declare that the work, which is being presented in the Summer Internship Report, entitled "**FabBits : A video cataloguer**" in partial fulfilment for the award of Degree of Bachelor of Technology in the **Department of Computer Science submitted to Swami Keshvanand Institute of Technology, Management & Gramothan, Jaipur, Rajasthan Technical University** is a record of my own investigations carried under the guidance of **Mr. Carlos Fernandez, CCExtractor.org**. I have not submitted the matter presented in this report elsewhere for the award of any other degree.

Archit Mathur
BTech, Computer Science
Roll Number - 15ESKCS033
Swami Keshvanand Institute of Technology, Management & Gramothan, Jaipur

ACKNOWLEDGMENT

I feel immense pleasure in expressing my regards to the Chairman **Mr. Surja Ram Meel,** Director **Mr. Jaipal Meel**, Registrar **Mrs. Rachana Meel**, Director (Academics) **Prof. (Dr.) S. L. Surana**, Director (D&W) **Prof. (Dr.) S. K. Calla** & Principal **Prof. (Dr.) Ramesh Prachar** Swami Keshvanand Institute of Technology, Management & Gramothan, Jaipur for providing me necessary facilities during the various stages of this work.

I would like to thank **Dr. C. M. Choudhary**, Professor & Head, Department of Computer Science & Engineering and **Dr. Anil Chaudhary**, Professor & Head, Department of Information Technology for provide me opportunity to work in consistent direction and providing their valuable suggestions to improve Seminar Report.

I would like to thank my esteemed guide **Mr Sunil Dhankar**, Associate Professor, Department of Computer Science & Engineering, Swami Keshvanand Institute of Technology, Management & Gramothan, Jaipur for his valuable guidance, keen interest, constant encouragement, incessant inspiration and continuous help throughout this work. Specially I acknowledge his support when I was stuck and him suggesting me new ideas to solve the problems. His vast experience and realistic approach have been of great help to me. I am honored having Mr Sunil Dhankar as my seminar report supervisor. Her excellent guidance has been instrumental in making this work a success.

I would also like to express my thanks to my parents for their support and blessings. In addition, a very special thanks to all my colleagues and friends for their support in the completion of this work.

Archit Mathur

BTech, Computer Science

Roll Number - 15ESKCS033

CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

S No	Abbreviation	Meaning
1	DNN	Deep Neural Network
2	LBPH	Local Binary Pattern Histogram
3	HOG	Histogram of Gradients
4	CNN	Convolutional Neural Network
5	SMStd	Slow Motion Standard
6	SMHigh	Slow Motion High Speed
7	GUI	Graphical User Interface
8	API	Application Programming Interface
9	GNU	GNU’s Not Unix

ABOUT THE ORGANISATION

Google Summer of Code

Google Summer of Code is a global program focused on **bringing more student developers into open source software development**. Students work with an open source organization on a 3 month programming project during their break from school.

The idea for the Summer of Code came directly from **Google's founders**, Sergey Brin and Larry Page. From 2007 until 2009 Leslie Hawthorn, who has been involved in the project since 2006, was the program manager. From 2010 until 2015, Carol Smith was the program manager. In 2016, Stephanie Taylor took over management of the program.

CCExtractor.org

I worked with CCExtractor.org which is originally an organization about subtitles and accessibility. However, they're doing lots of other things that are related, including tinkering with open video hardware (JokerTV), imaging (OCR), and more. They have projects that cover a range of interests, despite our core tool being the most amazing subtitle extractor.

They mainly work on a tool, CCExtractor, that **analyzes video files and produces independent subtitle files from the closed captions data**. CCExtractor is the one tool that is free, portable, open source and community managed that can take a recording from a TV show and generate an external subtitle file for it. It is portable, small, and very fast. It works in Linux, Windows, and OSX.

ABSTRACT

Videos contain plethora of contextual information. For example, in a movie there are fighting scenes, sentimental scenes, romantic scenes, and many others. In a cricket match, there are wickets, sixes, fours et cetera. With the advent of the data-driven age, amateurs, researchers, and organisations alike require some specific part of this contextual information for their needs; maybe for creating a highlights reel of a sports match or mining data from movies for their machine learning models. This makes parts of certain types of videos very useful. **FabBits, the software I built, tries to automate** finding these parts.

Following are the things it is able to detect -

- Action sequences in movies/shows

  Fighting scenes, car chases etc
- Summary of movies/shows

  A frame by frame summary of the video
- Actor-specific scenes in movies/shows

  Scenes of 30+ supported actors can be extracted
- Jokes in sitcoms

  Any content having canned laughter can have jokes extracted
- Goals in Soccer

  Works by processing just the scoreboard area
- Three pointers in Basketball

  Similar to Goals in Soccer but uses OCR too

Chapter I

INTRODUCTION

1.1 Purpose

This era in tech, and the world in general, has been dominated by data. All those organisations, tech-based or not, that utilise the data they own or have access to have been thriving. For example, Facebook. A big chunk of their revenue comes from leveraging the information of the users for gaining insights about their behaviour.

However, **insights from data should not be a privilege to only a few rich and powerful** but rather to the whole community as one. FabBits uses videos to extract specific contextual information. As it is understood, videos contain a gigantic amount of contextual information. For example, in a movie there are fighting scenes, sentimental scenes, romantic scenes, and many others. In a cricket match, there are wickets, sixes, fours et cetera. With the advent of the data-driven age, amateurs, researchers, and organisations alike **require some specific part of this contextual information for their needs**; maybe for creating a highlights reel of a sports match or mining data from movies for their machine learning models. This makes parts of certain types of videos very useful. FabBits helps by automating the search for these parts.

1.2 Scope

There is always going to be a need for getting contextual information from videos. **FabBits has been developed very flexibly, both as a software and as a cataloguer**. Some use-cases it can be extended for, that are closely related to the work I have done, are -

- Slow motion clips/sequences in sports or movies

- Goal misses in soccer

1.3 Technologies used

The software was built using in Python3 using the following libraries -

- **PyQt5 : Used for building the GUI**

  Qt is set of cross-platform C++ libraries that implement high-level APIs for accessing many aspects of modern desktop and mobile systems. These include location and positioning services, multimedia, NFC and Bluetooth connectivity, a Chromium based web browser, as well as traditional UI development.

  *PyQt5 is a comprehensive set of Python bindings for Qt v5***.** It is implemented as more than 35 extension modules and enables Python to be used as an alternative application development language to C++ on all supported platforms including iOS and Android. It is a copyright of Riverbank Computing Limited and is released under two licenses - the GPL v3 license and a commercial license that allows development of proprietary applications.

- **OpenCV : Used for video and image processing**

  OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

  The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database etc.

- **Moviepy : Used for video and audio handling**

MoviePy is a Python module for video editing, which can be used for basic operations (like cuts, concatenations, title insertions), video compositing (a.k.a. non-linear editing), video processing, or to create advanced effects. It can read and write the most common video formats, including GIF.

It is open source and was originally written by Zulko and released under the MIT licence. It works on Windows, Mac, and Linux, with Python 2 or Python 3. The code is hosted on Github, where you can push improvements, report bugs and ask for help. There is also a MoviePy forum on Reddit and a mailing list on librelist .

- **Scipy : Used for audio processing**

SciPy is a free and open-source Python library used for scientific computing and technical computing.It contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering.

SciPy builds on the NumPy array object and is part of the NumPy stack which includes tools like Matplotlib, pandas and SymPy, and an expanding set of scientific computing libraries. This NumPy stack has similar users to other applications such as MATLAB, GNU Octave, and Scilab. The SciPy library is currently distributed under the BSD license, and its development is sponsored and supported by an open community of developers. It is also supported by Numfocus which is a community foundation for supporting reproducible and accessible science.

- **Tesserocr : Used for Optical Character Recognition**

Tesserocr simple, Pillow-friendly, wrapper around the tesseract-ocr API for Optical Character Recognition (OCR). It integrates directly with Tesseract's C++ API using Cython which allows for a simple Pythonic and easy-to-read source code. It enables real concurrent execution when used with Python's threading module by releasing the GIL while processing an image in tesseract.

Tesserocr is designed to be Pillow-friendly but can also be used with image files instead. It is licensed under MIT licence and is available for Python 2.7, 3.3, 3.4, 3.5, 3.6, and 3.7.

- **Pillow : Used to preprocess images for OCR**

Pillow is a fork of the ever famous PIL (Python Imaging Library) ever since it’s developed discontinued. Python Imaging Library was a free library for the Python programming language that had support for opening, manipulating, and saving many different image file formats. It is available for Windows, Mac OS X and Linux. The latest version of PIL is 1.1.7, was released in September 2009 and supports Python 1.5.2–2.7.

Since PIL’s development was discontinued with the last commit to the PIL repository coming in 2011, Pillow, a successor project, forked the PIL repository and added Python 3.x support. This fork has been adopted as a replacement for the original PIL in Linux distributions including Debian and Ubuntu (since 13.04).

Chapter II

PROJECT DESCRIPTION

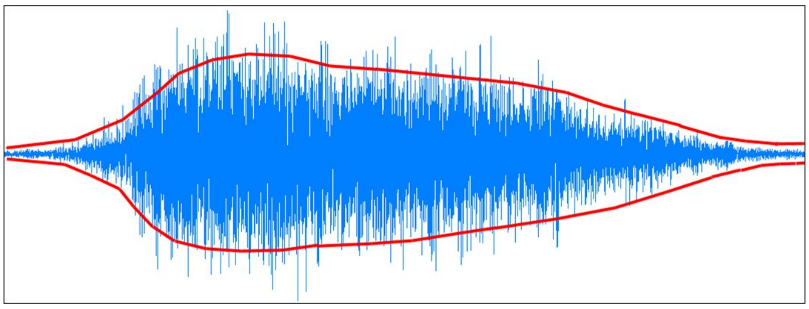
2.1 Use - Cases

Fabbits is able to catalogue the following type of clips from videos -

1. **Jokes in Sitcoms**

   Procedure -

   Funny jokes in sitcoms are always accompanied by canned laughter. Canned laughter tracks have a distinguishable structure in the time domain of the corresponding audio/video. There is an "envelope" of sorts, shown below, which signifies the abrupt starting of a laugh, during or after a joke, and its slow, scattered dying down.



Signal envelope of a laughter track

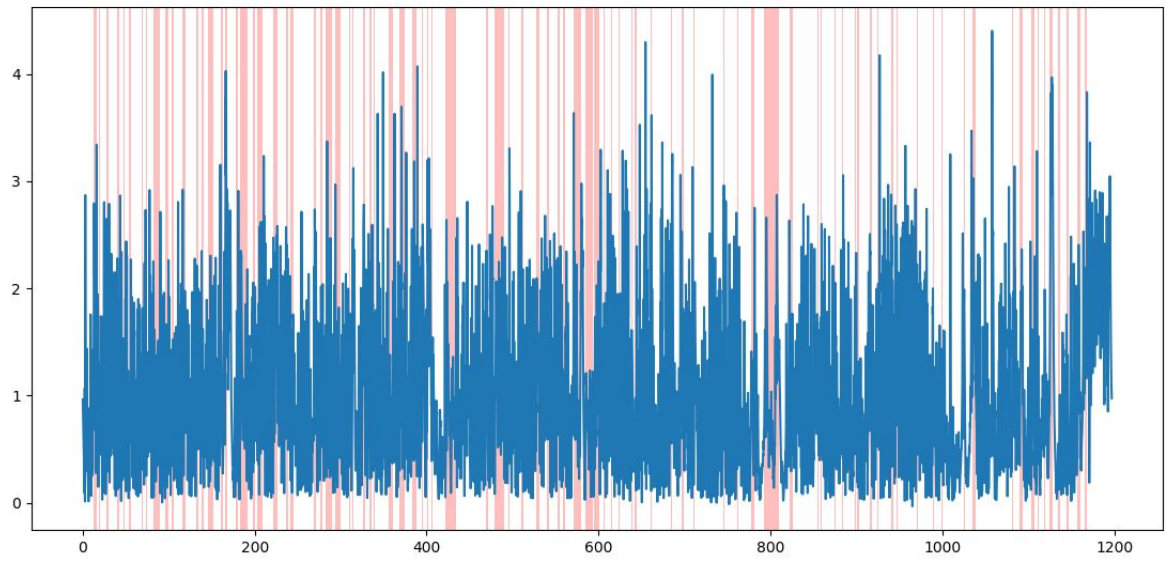
This distinct characteristic is further augmented by doing a Hilbert transform and calculating its absolute value. The signal so obtained is searched through for two thresholds -

1. Finding a threshold A.
2. Marking frames till another threshold B (<A) is reached

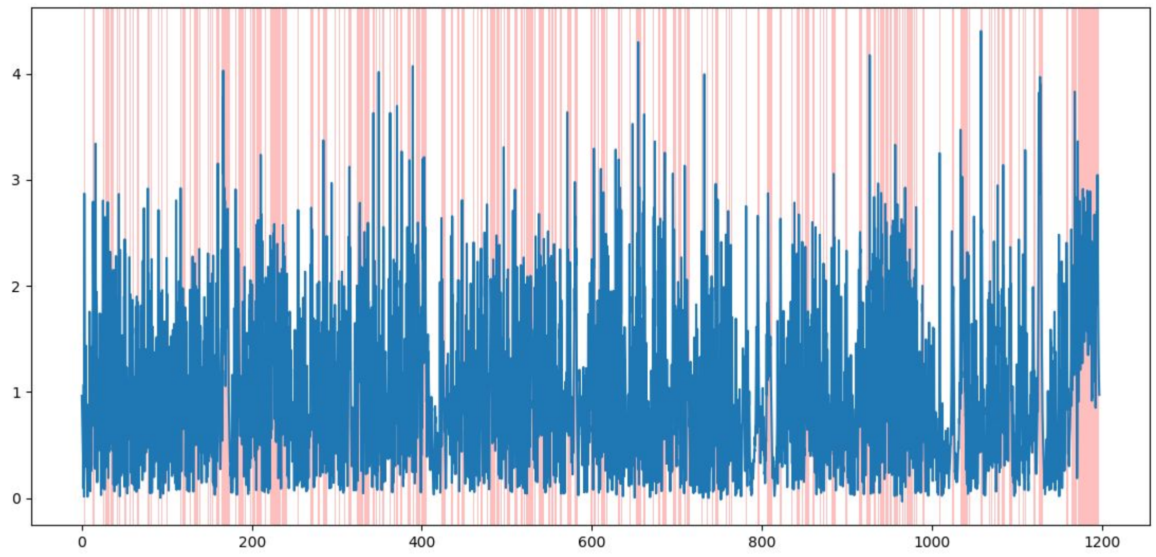
These marked frames constitute a scene that has laughter, implying the occurrence of a joke.

Example -

The red areas show occurrence of canned laughter.



Scenes with ***considerable, > 1s,*** *canned laughter in* ***Two and a Half Men S01E13***



Scenes found by the script to have canned laughter

2. Action scenes in movies/shows

Procedure -

Action scenes have quite a few features that differentiate them from other types of scenes. The following features are some of them -

Visual

a. High motion activity
b. High camera activity
c. Greater frequency of shot cuts etc

Auditory

a. Gunshot/Weapons' sounds
b. Screeches etc

After extracting these features from a set of handpicked movies and shows and labelling whether they depict an action sequence or not, I **trained a Support Vector Machine(SVM)** which will be able to able to classify whether a scene has an action sequence.

To actually find action sequences in the user provided video, the following needs to be done-

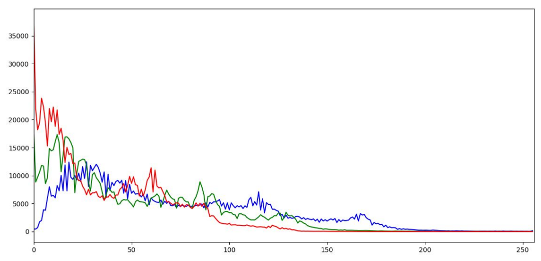
Scenes are made up of 'shots' which may be scattered and not in continuation for the whole scene. These shots need to be grouped together, using RGB histograms, to get the whole scene.

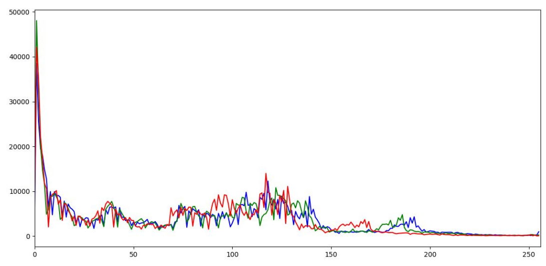
1. After grouping shots together, essentially segmenting different scenes, the above mentioned features will be extracted.
2. These "feature vectors", representing different scenes, will be passed through the trained SVM to classify whether they have an action sequence or not.

3. Summary of movies/shows

Procedure -

Every movie/show has different places, or sets, where the scene is filmed. Different scenes, or settings, have different RGB histograms which give the number of pixels of different shades/colours present in a frame. Thus, owing to the different ambience of different scenes, RGB histograms open a new pathway to compare scenes.





Significant distinction in the RGB histograms

Using this histogram-based comparison whenever a value of over some predetermined threshold is obtained, it would mean that the compared scenes are different, allowing the extraction of different settings in the provided video.

4. Goals in Soccer

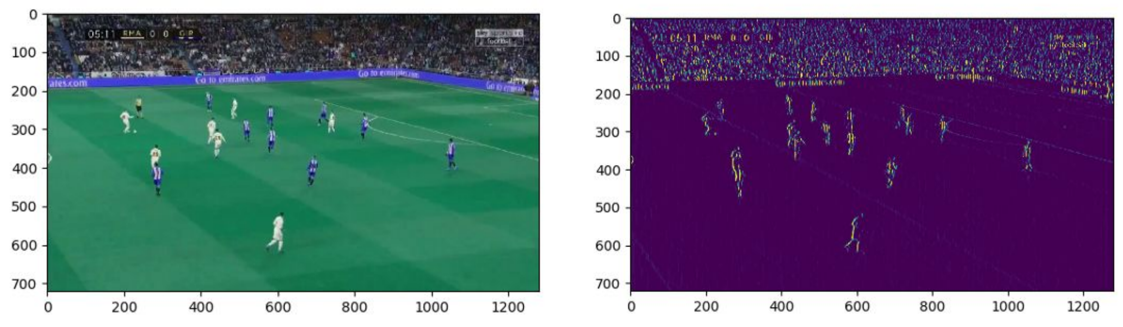
Procedure -

My approach for soccer goal detection was basically this -

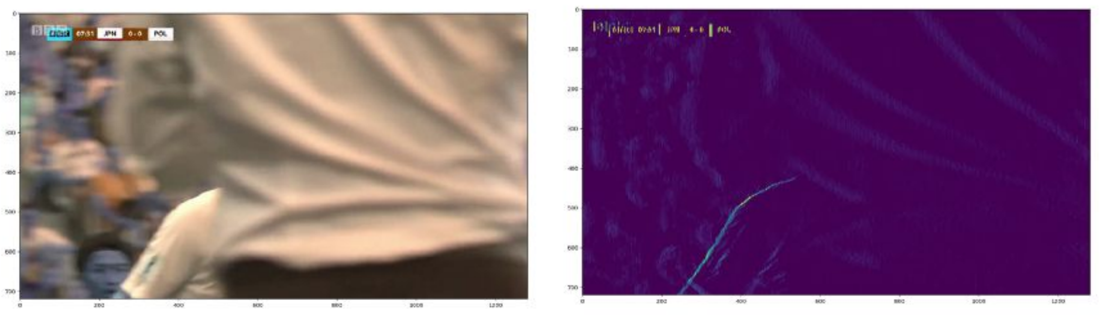
- Segment the scoreboard.
- Monitor the changes in scoreboard for detecting goals.

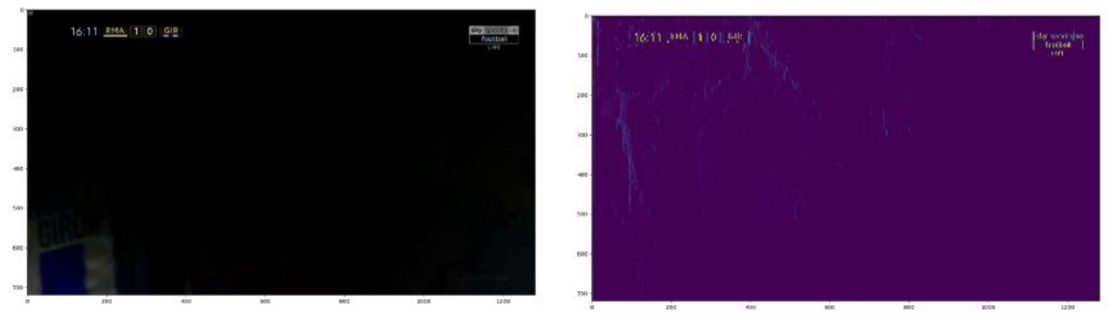
Turns out the scoreboard area has quite a bit of *edge content* owing to the textual regions it has. Due to this, Sobel/Scharr filter outputs show a better imprint of the scoreboard area and also weeds out unnecessary details. So, basically, what I had to do was pass a random frame, given it has the scoreboard, through Sobel filter and get the contours. Right? Actually, no.

This is what happened when I passed a random frame in -



I got around this by searching a sub clip, ~10 min, of the video for frames with least edge content and used the best one for further processing. Below are some frames with low edge content.

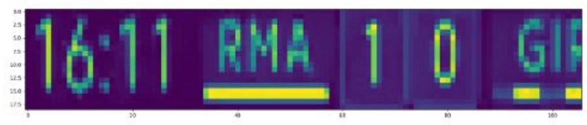


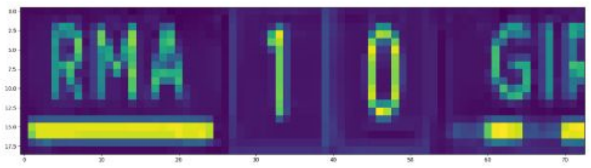


Then, after blurring and morphing, I found the contours in the least-edges, LE henceforth, frame, got their bounding boxes, and marked the one as the scoreboard that satisfied a certain criteria for area and aspect ratio.

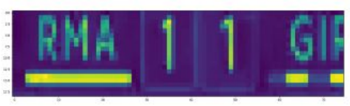
However, there is still an unnecessary thing that can pose problems later — the timer. Since my approach is based on comparing changes in scoreboards, the fact that there is something in there changing every second is pretty bad.

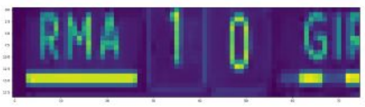
So, to remove the timer, I calculated the *structural similarity* between the LE frame's scoreboard and the scoreboard of the frame that comes after a second to the LE frame. And since the only thing that changes in such a short while is the timer, I was able to remove the timer side from the scoreboard.

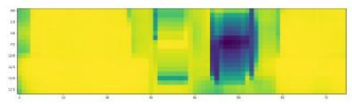




The next step is monitoring the score-only area for changes. I used structural similarity again for getting the differences between two scoreboards. This is how it looks when the score changes -







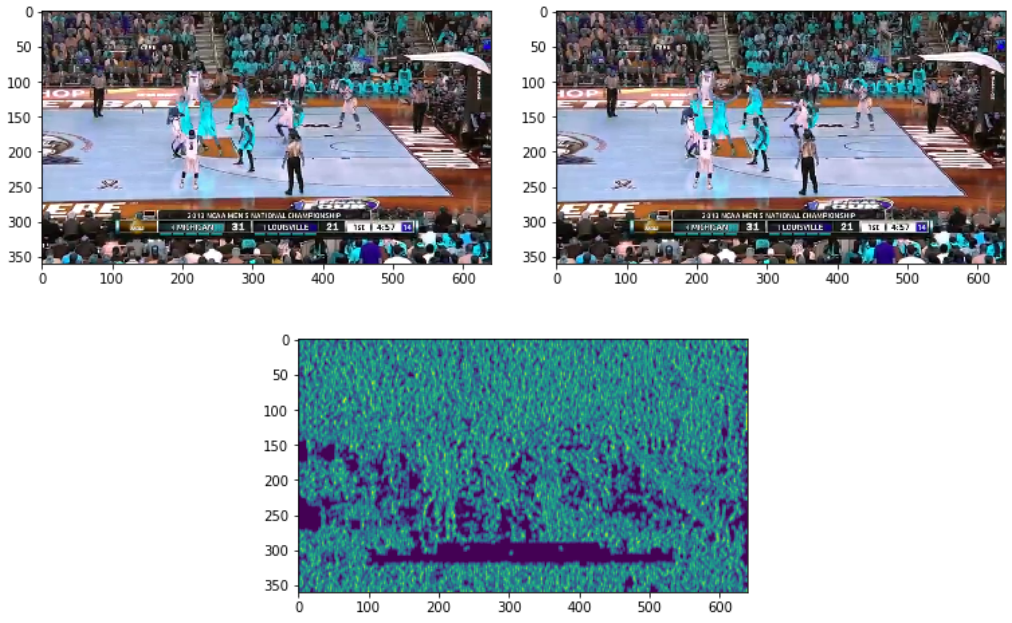
Then it was just a matter of iterating through the match and comparing the score-only area for significant changes. Although, choosing a threshold for similarity, whose breach would mean a goal, is another problem by itself, based on four matches from different studios, I found out that 87.5% is a good bet.

5. Three pointers in Basketball

Going about detecting 3-pointers was very smooth since I had done goal detection before.

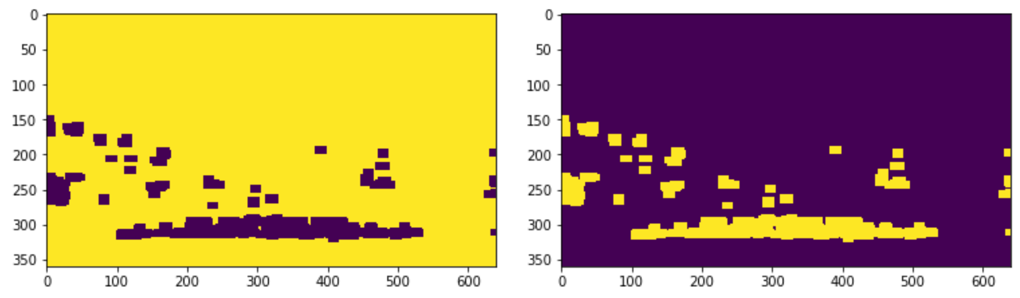
To detect the scoreboard region, I took 2 consecutive frames every second for 2 min. I then subtracted one from the other and applied Sobel filter and blurring on the result. At this point, the scoreboard area was distinct from its surroundings

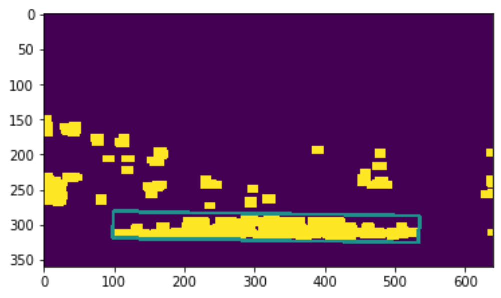
.



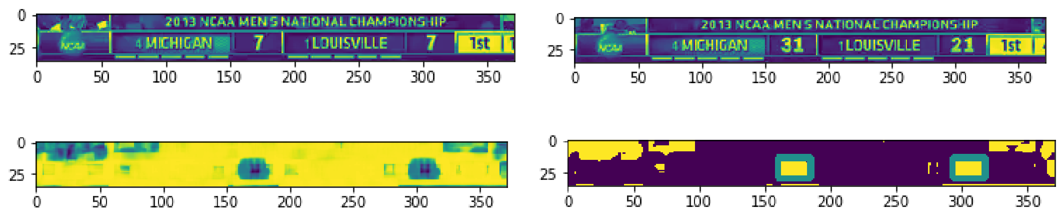
The consecutive frames with their difference operated on Sobel filter and blurred.

Since the main aim was getting the contour of the scoreboard, I then applied the closing morphology transform, reversed the pixel intensity, and calculated all the contours. If any contour satisfied the criteria for the shape and size of a typical scoreboard, it would become a candidate for the scoreboard. Remember I did this 120 times? I then sorted all the candidates based on their heights and chose the top candidate as the scoreboard.

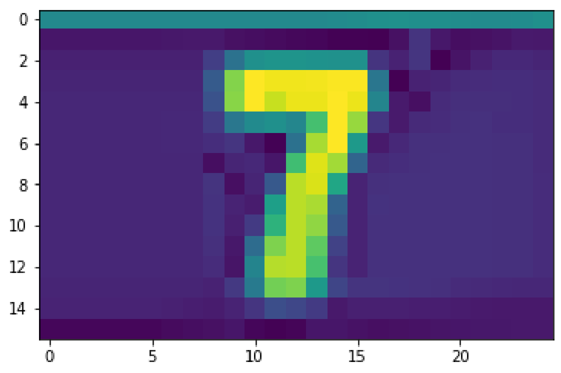


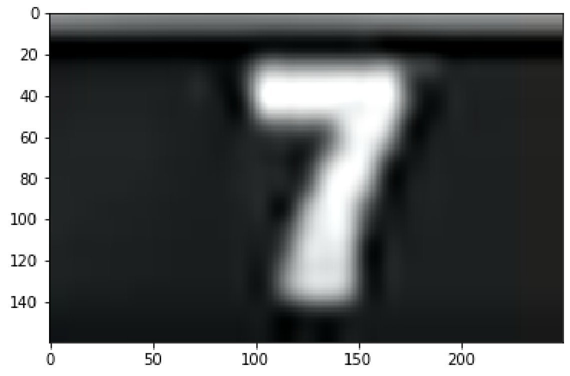


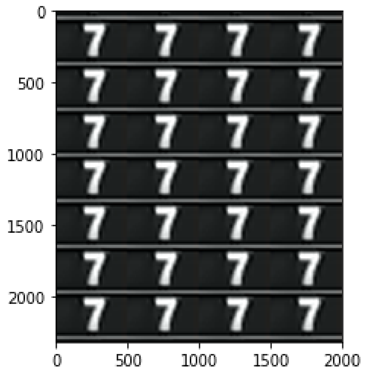
Since the scoreboard region was found, it was time for detecting the two score regions of teams. For this, I first found the initial scoreboard — by iterating from 5 min onwards and structurally comparing with the scoreboard region from before. Once a certain threshold, I chose 85%, was crossed, I found the contours on the SSIM image of this initial scoreboard and the scoreboard from before. Based on the shape and size of digits, two contours were chosen that depicted the scores of teams.


2013 NCAA MEN'S NATIONAL CHAMPIONSHIP
4 MICHIGAN
7
1 LOUISVILLE
7
1st
2013 NCAA MEN'S NATIONAL CHAMPIONSHIP
4 MICHIGAN
31
1 LOUISVILLE
21
1st

Then it was all Tesseract magic — or so I thought! Since these contours were very small, OCR did very poorly. I tried resizing and using anti-aliasing on the two score contour images with Pillow but Tesseract was still adamant. However, I went around this by padding the image with, well, itself (cv2.BORDER_WRAP) and resized it. Turns out OCR really liked that!







So now that I had access to scores at every moment, although I kinda decided to do this check after every 5s for less computations, I just had to check if any score would increase by 3. I also threw in some conditions in case OCR confused similar looking digits (2–7, 4–9). And this is how I found 3-pointers!

6. Slow-motion from soccer videos

First up, it's important to know that there are two types of slow motions — one consists of repeated and inserted frames and the other that comes from cameras that record at high speed/fps. Taking aside their differences from the normal speed clips, both of them differ from each other as well. Let's call the first one SMStd and the second SMHigh.

The thing with slomos in sports is that they are always preceded and succeeded by a gradual transition effect; usually an animation of the sponsor's/studio's logo. This makes things easier since now only the clips that are between gradual transitions need to be checked.

Once the *candidate shots* between gradual transitions have been found, HSV frame difference is calculated between consecutive frames.

$$HD(i,j) = H_t(i,j) - H_{t-1}(i,j)$$

$$SD(i,j) = S_t(i,j) - S_{t-1}(i,j)$$

$$DF[n] = \frac{1}{M * N}\sum_{i=1}^{M}\sum_{j=1}^{N}(HD(i,j)^2 + SD(i,j)^2 + VD(i,j)^2) \qquad VD(i,j) = V_t(i,j) - V_{t-1}(i,j)$$

X_t(i, j) is the H/S/V value of a pixel (i, j) on frame t. M and N correspond to the height and width.

Using this, repeated and inserted frames can be found; like if DF[n] is less than a threshold, n will be a repeated frame and since inserted frames are a combination of actual frames, their DF pattern is like this —

$$DF[p] > DF[p+1] > \cdots > DF[n]$$
$$DF[n] < \cdots < DF[q] < DF[q+1]$$

Thus, based on criteria revolving around DF, repeated and inserted frames are found and a clip is marked as SMStd only if in every 30 consecutive frames there are more than 10 repeated and inserted frames among them.

Then comes the turn of SMHigh. The authors introduced a lot of notations that help determine whether a clip is SMHigh.

- *AverDiff[s]*: The average frame difference of candidate shot *s* containing frames *i* to *j*.

$$AverDiff[s] = \frac{1}{j-i+1-r}\sum_{k=i}^{j} DF'[k], \tag{13}$$

$$DF[k]=\begin{cases} 0 & \text{if frame } k \text{ is a repeating or inserted frame} \\ 1 & \text{otherwise} \end{cases} \tag{14}$$

Where r is the sum of repeating and inserted frames number in s.

- *AverDiffstd*: The average frame difference in all SMStds.

$$AverDiffst\ d = \frac{1}{NumofSMstd}\sum_{s} Averdiff\ [s] \tag{15}$$

Where s is an SMStd, and *NumofSMStd* is the number of SMStds.

- *FDC[n]*: The dominant color of frame *n*.

$$FDC[n] = (FDH[n], FDS[n], FDV[n]) \tag{16}$$

$$(FDH[n], FDS[n], FDV[n]) = \underset{h,s,v}{\operatorname{argmax}} H[n][h][s][v] \tag{17}$$

- *FDCP[n]*: The dominant color percentage of frame *n*.

$$FDCP[n] = \frac{\sum_{i=-1}\sum_{j=-1}\sum_{k=-1} H[n][DH[n]+i][DS[n]+j][DV[n]+k]}{PN} \tag{18}$$

Where *PN* is the total pixels number of frame *n*.

- *SDC[s]*: The dominant color of shot *s*.

$$SDC[s] = (SDH[s], SDS[s], SDV[s]) = \frac{1}{FN}\sum_{n} FDC[n] \tag{19}$$

Where *FN* is the total frames number of shot *s*.

- *SDCP[s]*: The dominant color percentage of shot *s*.

$$SDCP\ [s] = \frac{1}{FN}\sum_{n} FDCP\ [n] \tag{20}$$

The test for SMHigh is done on those candidate shots that were discarded by the SMStd criteria — the *non-SMStd*. The main idea behind SMHigh's classification is that since high speed cameras are used, there are less/no repeated+inserted frames which makes their frame differences, typically, greater than those of SMStd. Thus, the average frame difference is calculated for all SMStds and is compared with all of non-SMStds'. The non-SMStds whose average frame difference is higher are kept for further processing.

The paper also talks about how a close-up shot can be mislabelled as a SMHigh. To rectify this, the dominant color % of the non-SMStd is found and is compared with those of SMStds. For almost all SMStds the dominant color is green but that's not really the case with a close-up shot. Also, since SMHighs show the same event as SMStds, they should have similar dominant color.

So based on this, the selected non-SMStds that have similar dominant color as SMStds are chosen to be SMHigh; essentially discarding closeup shots.

Once the processing with all the non-SMStd is completed, the neighbouring shots of SMStds also go through the same processing. This is due to the fact that most of the times a SMHigh succeeds a SMStd.

7. Actor-specific scenes

Procedure -

To find actors in videos, I trained a LBPH (Local Binary Pattern Histogram) classifier on a dataset of actors scraped from IMDb. I also used the MSRA-CFW, CelebA, and PubFig datasets. These datasets were cleaned such that only the facial part remained.

.

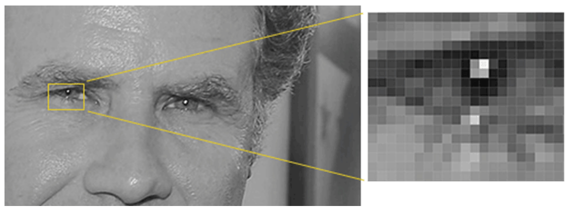
After the training, detecting scenes with the user specified actor consisted of the following steps -

1. Segmenting all the people/faces in the frame using HOG.

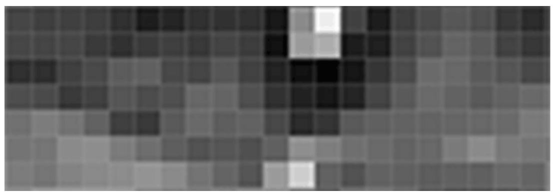
2. Recognition of the detected faces using the trained LBPH classifier.
3. Marking the scene if any on-screen actors match with the user's requirement.
4. Compiling all the marked scenes.

Among many other techniques, faces in a frame can be detected using a method called **Histogram of Oriented Gradient** or HOG. Following is how it's done -

a. After converting the image to grayscale, every single pixel in the image is looked at one at a time. For every single pixel, the pixels directly surrounding it are considered:



b. The goal is to figure out how dark the current pixel is compared to the pixels directly surrounding it. Then an "arrow is drawn" showing in which direction the image is getting darker

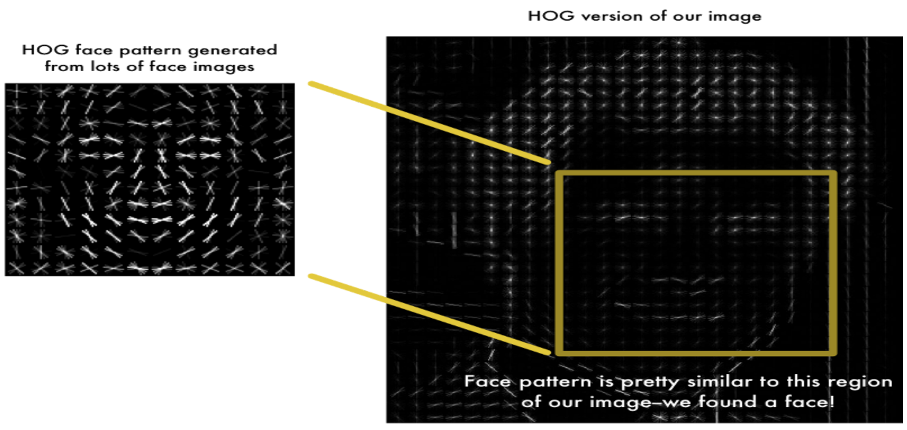


c. If this process is repeated for every single pixel in the image, each pixel is replaced by an arrow. These arrows are called *gradients* and they show the flow from light to dark across the entire image

d. Saving the gradient for every single pixel gives way too much detail. To get around this, the image is broken up into small squares of 16x16 pixels each. In each square, gradients that point in each major direction are counted (how many point up, point up-right, point right et cetera). Then that square in the image with the strongest gradient is replaced.

.

e. This results in a basic structural representation of the face


HOG face pattern generated from lots of face images
HOG version of our image
Face pattern is pretty similar to this region of our image—we found a face!
.

f. To find faces in HOG image, all that is to be done is to find the part of the image that looks the most similar to a known HOG pattern that was extracted from a bunch of other training faces.

Thus, after detecting faces in the frame, they will be **passed through the LBPH model** to get the identities of different actors. If one them is the user desired actor, the scene is added for compilation

2.2 Development Environment

PC Specs

My workstation had the following specifications -

- Intel 7th generation Core i7 octa-core processor
- NVIDIA Geforce GTX 1050 Ti (4GB) GPU
- 8 GB DDR4 RAM
- 1TB 7200RPM HDD
- Ubuntu 16 LTS and Windows 10 (Dual boot)

Project Dependencies

Dependencies can be installed by running -

```
pip3 install scipy
pip3 install opencv-python
pip3 install moviepy
pip3 install pyqt5
pip3 install Pillow
pip3 install tesserocr
```

or if you are the Anaconda kind -

```
conda install -c conda-forge scipy
conda install -c conda-forge opencv
conda install -c conda-forge moviepy
conda install -c anaconda pyqt
conda install -c conda-forge pillow
conda install -c simonflueckiger tesserocr
```

2.3 Project Itinerary

I followed this schedule -

- May 14 - May 21

  Work Done : Due to the last leg of end-terms during this period, I only designed the GUI and modeled its workflow. I also tried to modularise the subroutines I had already written to achieve reusability.

- May 22 - June 2

  Task : Making GUI and completing use-cases #1 - #3

  Work Done : I finished all the three use-cases regarding movies/shows and integrated them in the GUI. Having completed, I believe, about 70% of the work on the core-algorithms in the pre-GSoC period, this was be relatively quick.

  Deliverable : Working software which will find interesting bits in movies/shows

  Potential challenges : Integrating the modules with PyQt presented some challenges as I had never worked with it before.

- June 3 - June 6

  Work Done : I documented the code and tested the software against different types of movies and shows. I also prepared a report for the mid-term evaluation.

- June 6 - June 20

  Task : Implementing use-case #4

  Work Done : I implemented the two research papers that segment the scoreboard along with their score extraction methods. I also dabbled with OCR and eventually choose the faster score extraction method to go forward with. After that, I compared the results of loudness +

scoreboard and excitement + scoreboard and integrated the one in the GUI that had a good combination of speed and accuracy.

Deliverable : Working, documented, and modular code that segments goals from a soccer video integrated in the GUI.

Potential challenges : Using algorithms for segmenting the scoreboard that were temporally and computationally expensive for big high-definition videos.

- June 21 - June 25/27

  Task : Use-case #5

  Work Done : Three pointers' extraction used the same core algorithms as goals' detection. However, they had a significant difference in algorithm parameters owing to the different structures and screens of the two games. So, to find the required parameters, I visualized the outputs of Sobel and Laplacian filters for different basketball games to find a pattern. After getting the parameters, and completing this use-case, I added it to the GUI.

  Deliverable : Working, documented, and modular code, that finds three-pointers in a basketball game, integrated in the GUI.

  Potential challenges : It was computationally expensive when the file was large.

- June 25/27 - July 1

  Task : Use-case #6

  Work Done : This use-case essentially used the same modules used in use-case #2 and required similar parameter tuning like in use-case #5. Algorithmically, it was very easy to implement.

  Deliverable : Working, documented, and modular code, that finds slo-mos in a soccer game, integrated in the GUI.

Potential challenges : None.

- July 2 - July 10

  Task : Use-case #7

  Work Done : I started with cleaning the acquired data such that only the face made up a photo and scaled them to a proper limit (~ 50 x 50). After proper labelling, I fed this data into a LBPH model.
  For a start, I implemented a simple ConvNet and checked results using different stride sizes, pooling layers et cetera. If gave alright accuracy, (> 78%), but then I decided to create the LBPH model instead which gave > 93% accuracy.

  Deliverable : Working, documented, and modular code, that finds occurences of a certain actor in a movie/show.

  Potential challenges : The acquired datasets did not contain relatively less popular actors turning the model obsolete when the user requested for them.

- July 11 - July 12

  Work Done : I wrote the midterm report and tested the s/w for different movies.

- July 13 - July 16

  Task: Continued work on use-case #7

  Work Done : Used the model, along with HOG results, to find people in the movie/show frame and added it to the GUI.

  Deliverable : Use-case #7 integrated in the GUI.

- July 17 - July 25

  Task : Use-case #8

Work Done : After parameter tuning for goal misses and using it with the pre-built score extraction module, thus completing the proposed detector, I researched more about other features, like the crowd's groaning, that were helpful in increasing the confidence score of the event being a goal miss.

Deliverable : Working, documented, and modular code, that finds goal misses in a soccer match, integrated in the GUI.

Potential challenges : Intense dribbling and hyped matchups have similar cues to that of a goal miss; i.e excitement among the crowd and no score change. Even if the crowd's groans were taken into account, the detector was getting fooled.

- July 26 - July 31

  Task : Tinker with algorithms and GUI.

  Work Done : I used this period as a buffer of sorts and tried to improve the efficiency of algorithms and add usable features to the GUI like -

  1. Getting just the timestamps instead of a compilation video.
  2. A "progress screen" telling the user what stage the program is at.
  3. Opening the compilation video directly from the GUI et cetera

  I also completed anything that remained incomplete during this period.

  Deliverable : Improved GUI and, possibly, core algorithms.

- August 1 - August 6~14

  Work Done :

  - Took advices and feedback from the community over improvements.
  - Documented code, removed redundancies, and packaged the project into a cross-platform software.
  - Submitted!

2.4 Project Usage

1. Run the main GUI by -

```
python3 main.py
```

2. To find your FabBit of choice -
   - Click `MOVIES` or `SPORTS` button for their respective use-cases
   - Select the use-case from the sidebar
     - A pop-up dialog will ask for the actor if actor-specific scene was chosen
   - Click on `Choose File` to select the input video
   - Click on `Find FabBits`
   - Move the slider in the blue areas, which are the extracted FabBits, and play the video
   - Click on `Save FabBits` to save the extracted stuff into a video file

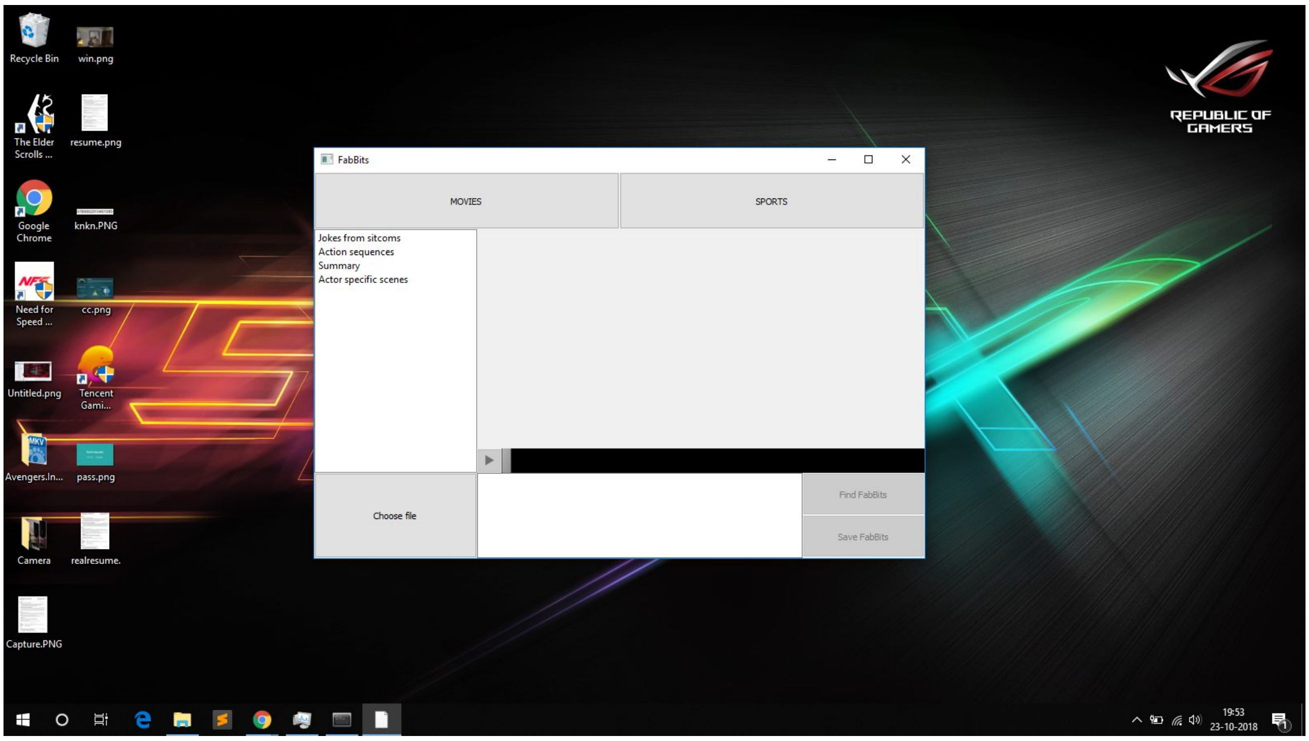
You can also run the respective files of use-cases to get their FabBit, like -

```
python3 goal_detector.py soccer_match.mp4
```

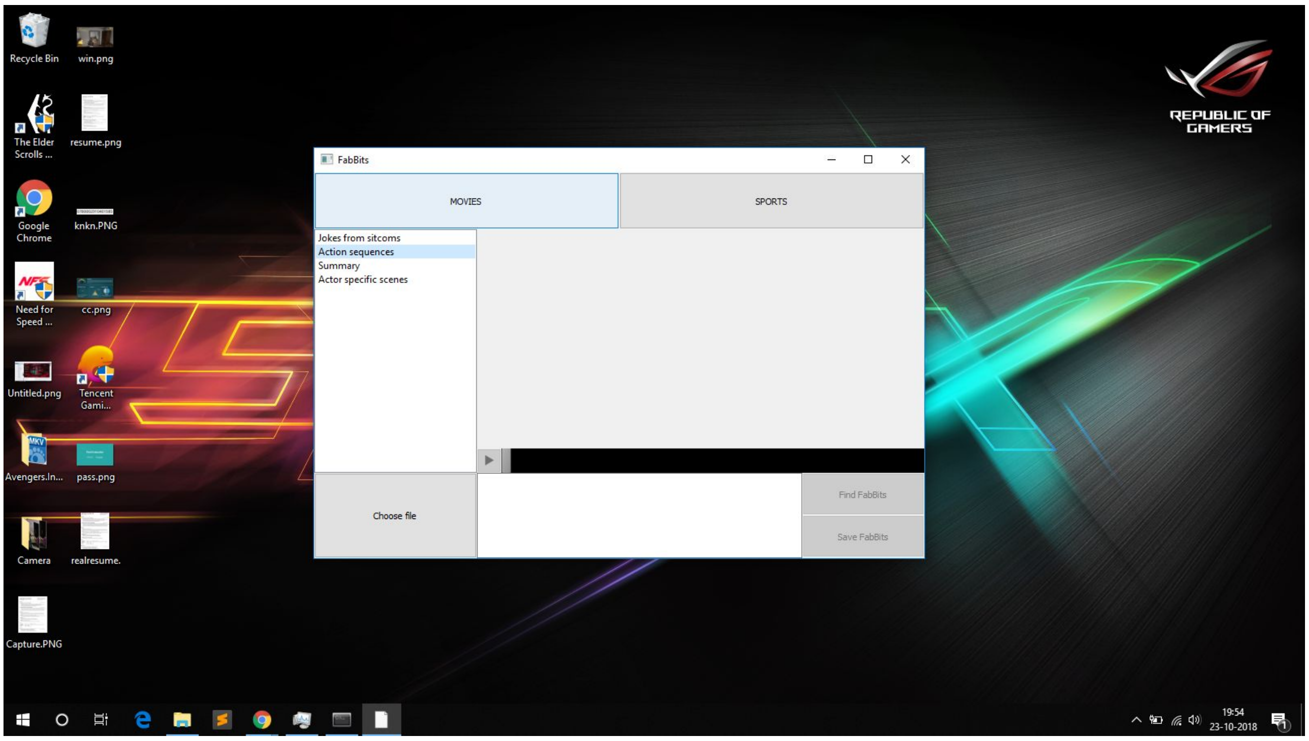
Chapter III

PROJECT SCREENSHOTS

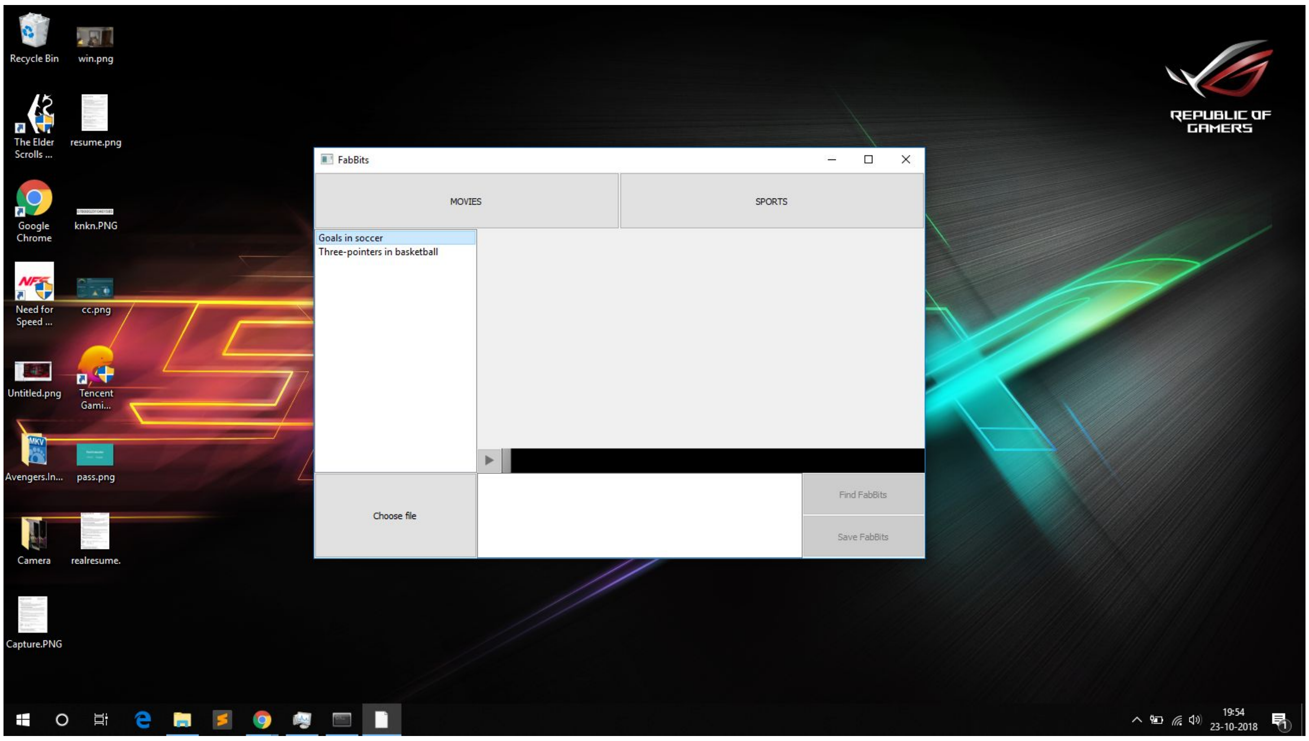
3.1 Starting FabBits



3.2 Choosing use case from Movies

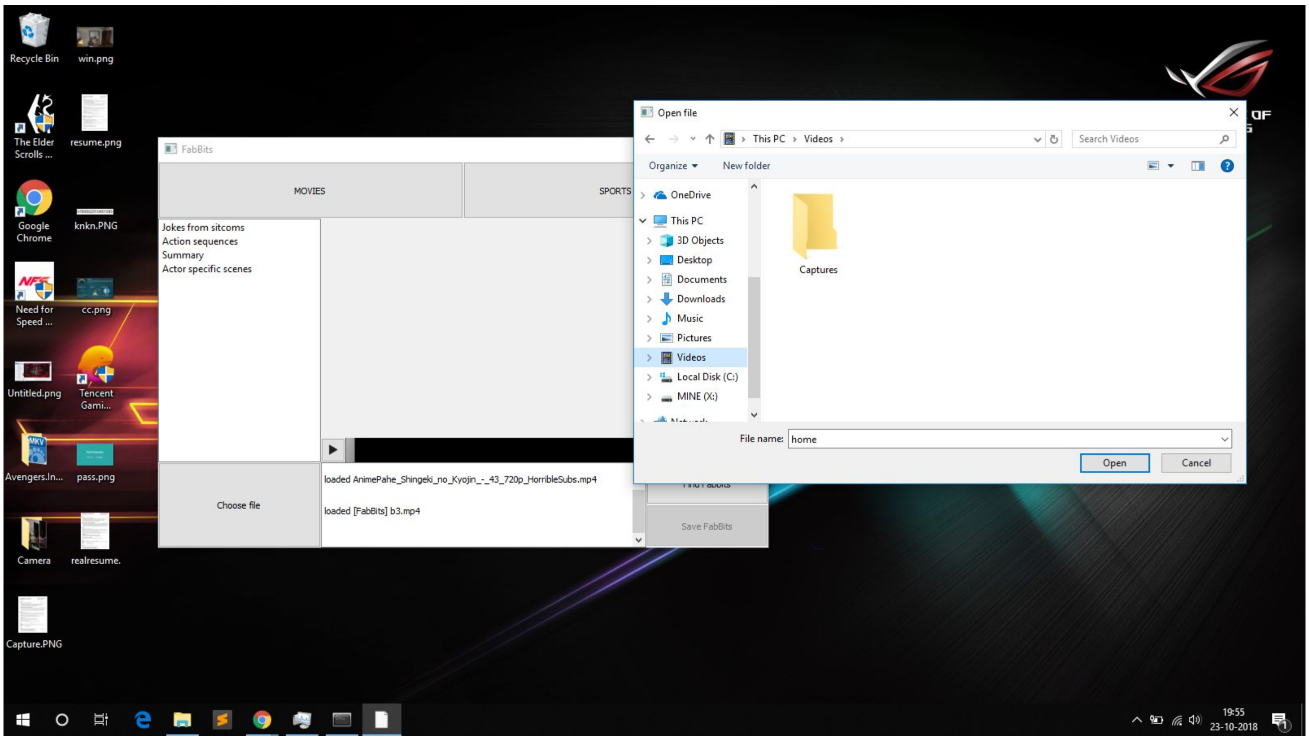


3.3 Choosing use-case from Sports



-

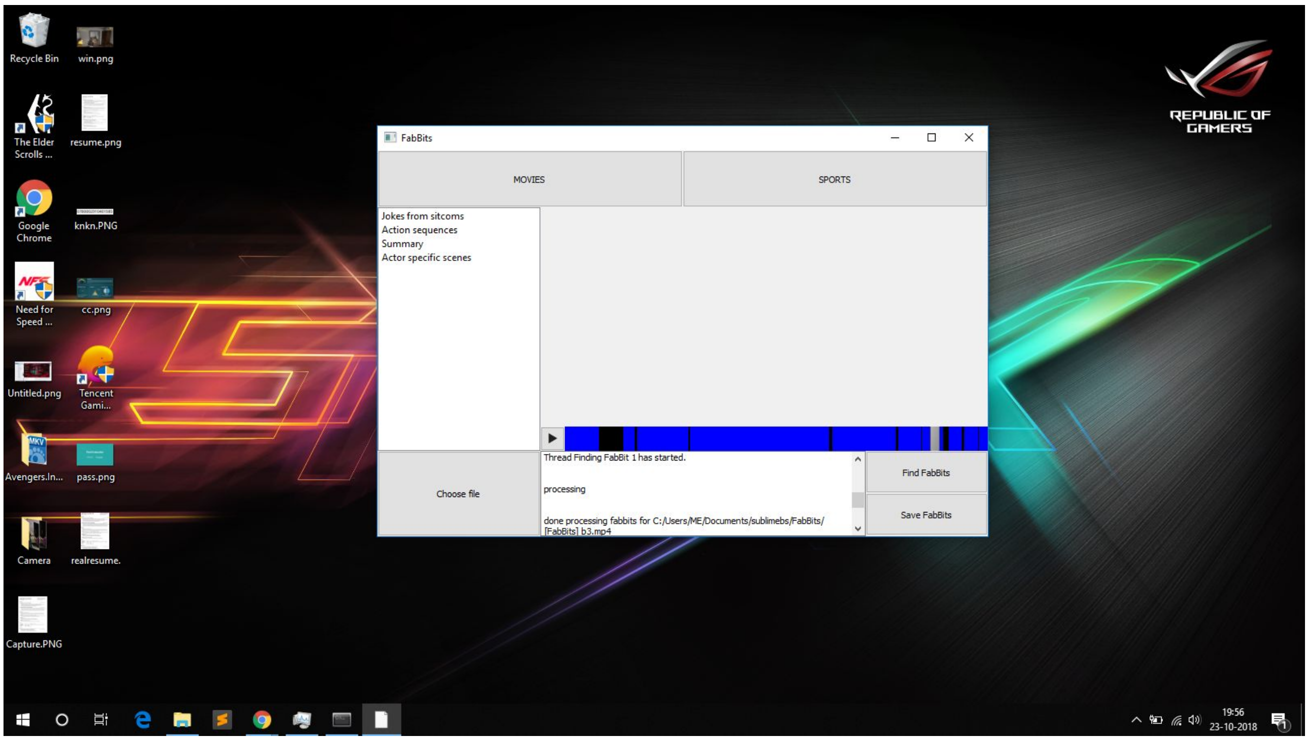
3.4 Choosing the file to extract FabBits from



```
loaded AnimePahe_Shingeki_no_Kyojin_-_43_720p_HorribleSubs.mp4

loaded [FabBits] b3.mp4
```

3.5 Extracting FabBits



```
Thread Finding FabBit 1 has started.

processing

done processing fabbits for C:/Users/ME/Documents/sublimebs/FabBits/
[FabBits] b3.mp4
```

CONCLUSION

This summer has been a great experience thanks to the CCExtractor community and Google. Building FabBits has made me learn a great deal about computer vision (more features for my DNNs!), data preprocessing, and GUI development! Let me tell you more about my summer.

The Project

FabBits is a video cataloguing software. I made it with PyQt so you can run it on every supported OS. It can catalogue stuff such as goals in soccer, scenes of a specific actor in a movie/show, jokes in sitcoms etc with high accuracy. I intended to do eight such use-cases but could only finish with six.

The Tech

- OpenCV ftw! Even though most of OpenCV's docs are all over the place and, like, every other function has a different naming convention, it is still a great library. And one I always look forward to work with.

- I used PyQt too and the development was very streamlined. There are many usable widgets and it has good customisation support too. Although if you don't build it from source, it may pose problems.

- Moviepy is very easy to use for video and audio editing and has certainly become my go-to library for the same.

- I never required multithreading or multiprocessing before, so never really used them but I have become accustomed to their API now.

- I also used little bit of stuff from SciPy, tesserocr, and matplotlib.

The Status

As I mentioned before, I wanted to release eight use-cases but was unable to do so. The two that I could not integrate into FabBits were — Slow Motion in Sports and Goal Misses in Soccer.

Slo-mos, well, just failed. I read a lot of literature but was not able to get results. As for Goal Misses, that's on me. I did not model my project timeline well enough and overestimated my work ethic. That, and, my laptop finally gave in (screen failure) so had to buy a new one which took a while.

Even though GSoC has ended, I would probably continue working to add new use-cases and make the existing stuff better.

The Growth

Working this summer has really broadened my horizons and has even given me great insights about things I knew before. What excites me the most now is the great many features I can extract for my ML models thanks to this 4-month grill in computer vision (I legit have so many ideas about stuff I want to build). I also completely read, understood, and implemented 11 research papers and referred to more than 25 for the seven use-cases I released. Great experience!

I also learned a great deal about myself too. About how I really, and urgently, need to work on my time management skills and how great at web surfing I have become! I hope these learnings continue forever!

REFERENCES

- **Audio-Based Action Scene Classification Using HMM-SVM Algorithm by Khin Myo Chit et al**
  (http://ijarcet.org/wp-content/uploads/IJARCET-VOL-2-ISSUE-4-1347-1351.pdf)

- **Action Scene Detection with Support Vector Machines; Liang-Hua Chen, Chih-Wen Su et al**
  (https://pdfs.semanticscholar.org/2a20/0432f71bf19c8efe19b686799e94226e2d32.pdf)

- **A Scoreboard Based Method for Goal events Detecting in Football Videos; Song Yang et al**
  (https://www.researchgate.net/publication/252020501_A_Scoreboard_Based_Method_for_Go al_Events_Detecting_in_Football_Videos)

- **Detecting Soccer Goal Scenes from Broadcast Video using Telop Region; Naoki Ueda et al**
  (http://www.iaiai.org/journals/index.php/IEE/article/view/187)

- **Anatomy of a Laugh Track**
  (https://archive.cnx.org/contents/a7ec0ec7-9093-4f55-93ef-3ac2ce71b1c8@3/anatomy-of-a-la ugh-track)

- **Real-Time Event Detection in Field Sport Videos**
  (https://pdfs.semanticscholar.org/b759/21d79a144e419f6fe71a06f65f26a51f6b44.pdf)

- **www.google.com**