Video Similarity Capstone Project Report

Team Members: Hemant Jain, Nimisha Sharath, Ryan Bae

I. Project Motivation

Video content is becoming increasingly common and rising as the preferred media due to its ability to engage the user and communicate complex information in short periods of time. Social media is the leading driver of video distribution with YouTube, Instagram and Facebook giving proof that videos account for over 80 percent of their total internet traffic. While it is intuitive and easy for humans to understand videos, the same cannot be said for computers. While computers can now understand images to a great degree, they must be *taught* how to understand video content and this is often an order of magnitude harder.

While websites such as YouTube allow search through millions of videos via keywords / queries, no such product exists that can search a video database by means of an input video. This form of video to video search aims to use the content of the actual video content. For example, if a user wants to search for a video similar to a man walking in a forest, currently the user must type search queries such as '*man walking in forest*' to look for similar videos. This project aims to create a search engine of this form that can take a video as input, and return videos that contain similar *objects, actions, scenes and sound* via using actual features of the video.

The features we used from the video include frame-level CNN features, frame-level dominant colors, second-level audio features and action/motion features. Through this project we have gained experience in topics such as computer vision, deep learning, software development, and data engineering.

II. Proposed Work

Jan 9 - Revised proposal due[Experimental setup + Implementing first 2 approaches]
- Long-term Recurrent Convolutional Networks for Visual Recognition and Description
- Learning Spatiotemporal Features with 3D Convolutional Network

Feb 6 - Interim project delivery [Results of first two methods to be shown + Implementing next two methods]
- Temporal Segment Networks: Towards Good Practices for Deep Action
- Temporal 3D ConvNets: New Architecture and Transfer Learning for Video

Mar 6 - Poster review
- Results of all 4 methods to be compared
- Improvements over each method to be shown

Mar 13 - Final project fair

Mar 18 - Final project documentation and materials due

III. Datasets

Main dataset for video querying and accuracy evaluation:

UCF-101 - http://crcv.ucf.edu/data/UCF101.php
This dataset is a collection of 13,200 videos that contain various actions collected from YouTube by University of Central Florida. There are total of 101 labels for the actions represented in the dataset. The videos range from roughly 10 to 30 seconds in length, and represent a single type of action, such as applying eye make up, golf swing, skiing, etc.

For unsupervised method models training:

Kinetics-400 - https://deepmind.com/research/open-source/open-source-datasets/kinetics/
The Kinetics dataset comes from Google Deepmind, and is a large dataset of ~500,000 videos that contain action sequences from YouTube. There are 400 classes of actions in this dataset, and each clip lasts roughly 10 seconds. Similar to the UCF-101 dataset, the Kinetics-400 dataset also contains a single type of action.

Audioset - https://research.google.com/audioset/
The AudioSet dataset is a large-scale collection of human-labeled 10-second sound clips drawn from YouTube videos. To collect all our data we worked with human annotators who verified the presence of sounds they heard within YouTube segments. To nominate segments for annotation, we relied on YouTube metadata and content-based search.

There are 2,084,320 YouTube videos containing 527 labels. Due to class imbalance, we have chosen a balanced segment of the same, as provided by the Audioset dataset, It contains 22,176 segments from distinct videos chosen with the same criteria: providing at least 59 examples per class with the fewest number of total segments.

For supervised video caption generation model training:

MSVD -
https://www.microsoft.com/en-us/download/details.aspx?id=52422&from=https%3A%2F%2Fresearch.microsoft.com%2Fen-us%2Fdownloads%2F38cf15fd-b8df-477e-a4e4-a4680caa75af%2F
MSVD stands for Microsoft Video Description Corpus. A set of YouTube videos were collected by Mechanical Turks that depict a single activity. The video clips were then annotated using a single sentence and put through minimal processing such as tokenization and punctuation removal to get the labeled corpus. It consists of 120,000 sentences describing the videos.

Challenges

- Size of datasets : all the datasets are >30GB in size. This certainly poses memory constraints on the machines initially available to us.

- None of these datasets have features extracted from them. By features we mean common image features that are the penultimate layers of common model architectures (eg VGG16) and this means we have to spend compute power on this.
- Adobe videos are not annotated. They have no descriptions and hence cannot be used for any of our study. The MSVD dataset is diverse and has descriptions which come close to the diverse nature of the videos present on the Adobe website.
- Quality of annotations are good, but not for all the videos. Since there are hundreds of clips and we cannot visually inspect all we have to accommodate some noise into the model.

IV. Models and Approaches

Our video similarity project breaks into the following steps. Video classification followed by clustering. We have taken multiple approaches to do the same.

A. Unsupervised Approach

The second approach researched is an unsupervised approach, where different types of video features are extracted in a vector format and k-nearest neighbor (KNN) algorithm is used to for similarity search. This method is broken down into 4 verticals, based on the types of video features each vertical extracts:

1. ResNet Features (Object Level)
2. 3D ResNet Features (Action Level)
3. Dominant Color Features (Color Level)
4. Audio Embeddings/Features (Audio Level)

ResNet Features (Object Level): Using CNN features from frames followed by KNN search

Convolutional Neural Networks are exceptionally useful for image classification and it was seen that they outperformed several other methods with little manual changes to architecture and hyperparameters.

Transfer learning is a method that uses pre-trained models to achieve significant performance (sometimes even without further training on the new dataset). For this, we used a *ResNet50* model that was pre-trained on the ImageNet dataset that was modified for feature extraction in Pytorch. Without retraining on the frame level data for the UCF101 video dataset, I then stored these in an HDF5 file.

We used Resnet 50 instead of ResNet 18 because it provides better features. However, it takes **20ms** with a GPU while ResNet 18 takes **10ms**.

We sample 8 frames uniformly from the video and fun the feature extraction script on it. Each feature representation is a **2048** dimensional vector. After the extracting features using a CNN (by removing the last layer) and we run a KNN similarity search as shown in Fig. 5.

The frame CNN features are then run in the GPU KNN implementation like before and we get K similar frame CNN features. These K frame CNN features are used to get the videos they correspond to and the duplicates are removed. The resultant similar videos are the one we are interested.

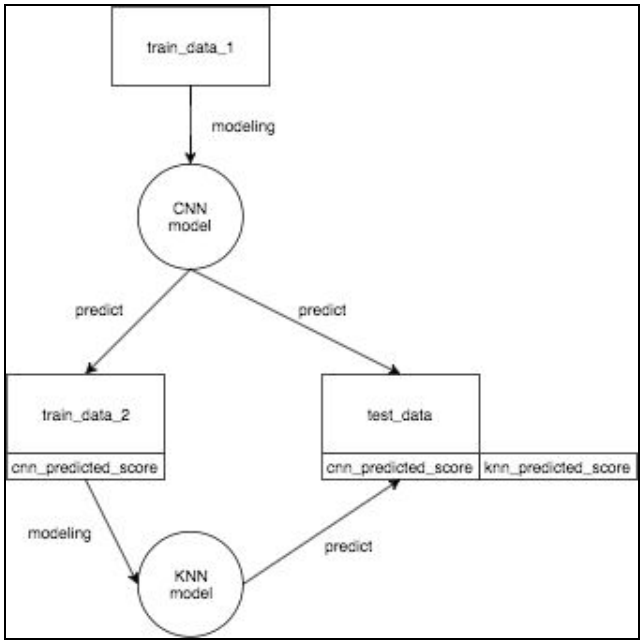


Fig. 5. A diagram showing the Hybrid CNN + KNN flow

Table 2 and 3 show some benchmarks based on number of queries and size of *k* for running KNN on the CNN features respectively.

No. of Queries	Runtime (s)
1	0.2
10	0.2
100	0.2

Table 2. Runtime per query for KNN on CNN features

Value of K	Runtime (s)
3	0.2
10	0.2
100	0.2

200	0.2

Table. 3. Runtime based on *k* for KNN on CNN features

When using 100 percent of the data, we get a class-wise accuracy of **96.4%** for the 101 classes. By taking a 70:30 split for train and test we still get a class-wise accuracy for test of **91.6%** for the 101 classes. This is excellent performance.

Only **3.2%** values on diagonal are less than **0.9** (Even though overall accuracy is 91.6%)

True Label	Predicted Label	% of error
Rock Climbing Indoor	Biking	5.2
Kayaking	Rafting	6.9
Rock Climbing Indoor	Skiing	5.4
Kayaking	Skijet	9.2

Table. 4. Misclassification error for KNN > 5 percent

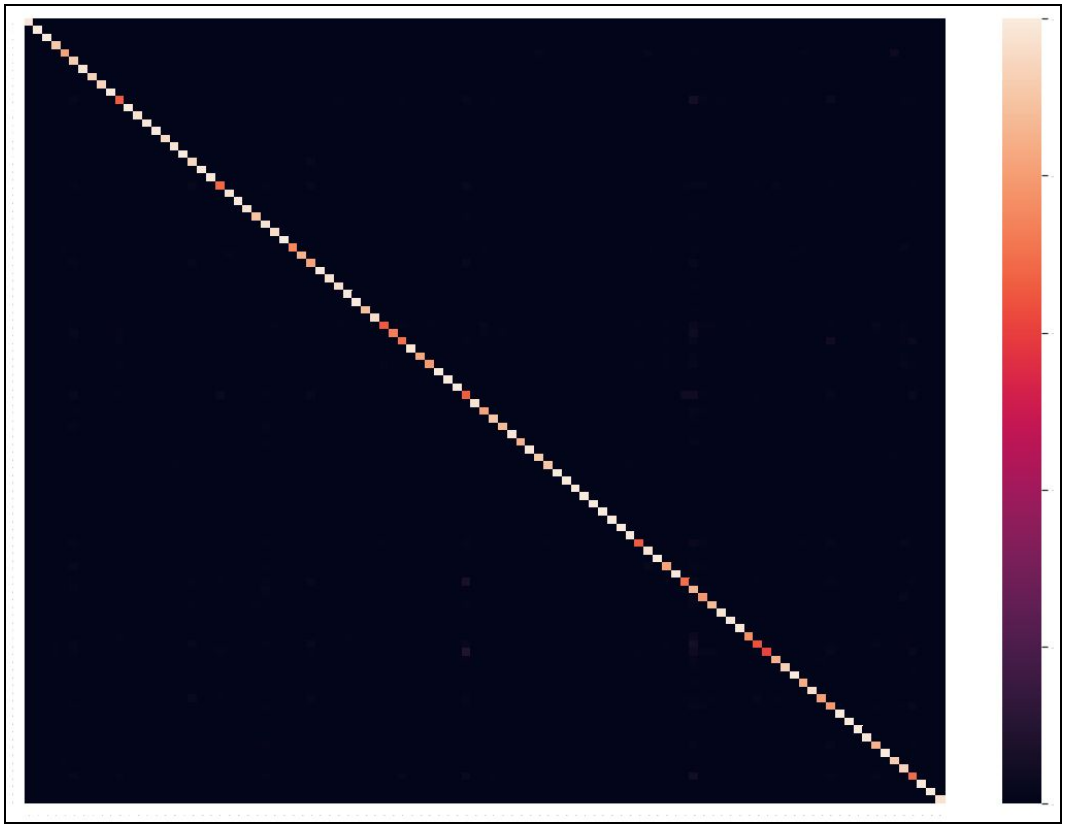


Fig. 6. Confusion matrix for KNN on CNN features

3D ResNet Features (Action level): Using features from 3D CNN ResNet model to capture action and motion of objects in videos

Feature similarity in 3D is similar (pun intended) to 2D feature similarity. However, the layers in CNN model are in 3D, which means the convolution, batching, and max pooling are all done in 3 dimensions. This means that a 3D CNN model can capture spatiotemporal aspects of videos in an end-to-end pipeline.

More information about 3D convolution neural nets is in this paper from Facebook Research and another paper with implementation from a Japanese research group.

An implementation of 3D CNN in PyTorch was forked into the project repo as submodule 3d-cnn with some modifications.

Feature extraction was done using a pre-trained model based on ResNet-34 from this Google Drive, which also contains other pre-trained models, from the authors of the PyTorch implementation. The pre-trained models are trained using the Kinetics-400 dataset, which has 400 classes instead of the 101 in the UCF-101 dataset.

The length of the 3D CNN feature vector is 512, same as that of the 2D CNN feature vector for a single frame. The sampling of the videos is done with 8 frames per video.

The total feature extraction process for 13,200 videos in the UCF-101 dataset was done in an AWS GPU instance with at least 32GB of memory, and took approximately 8 hours.

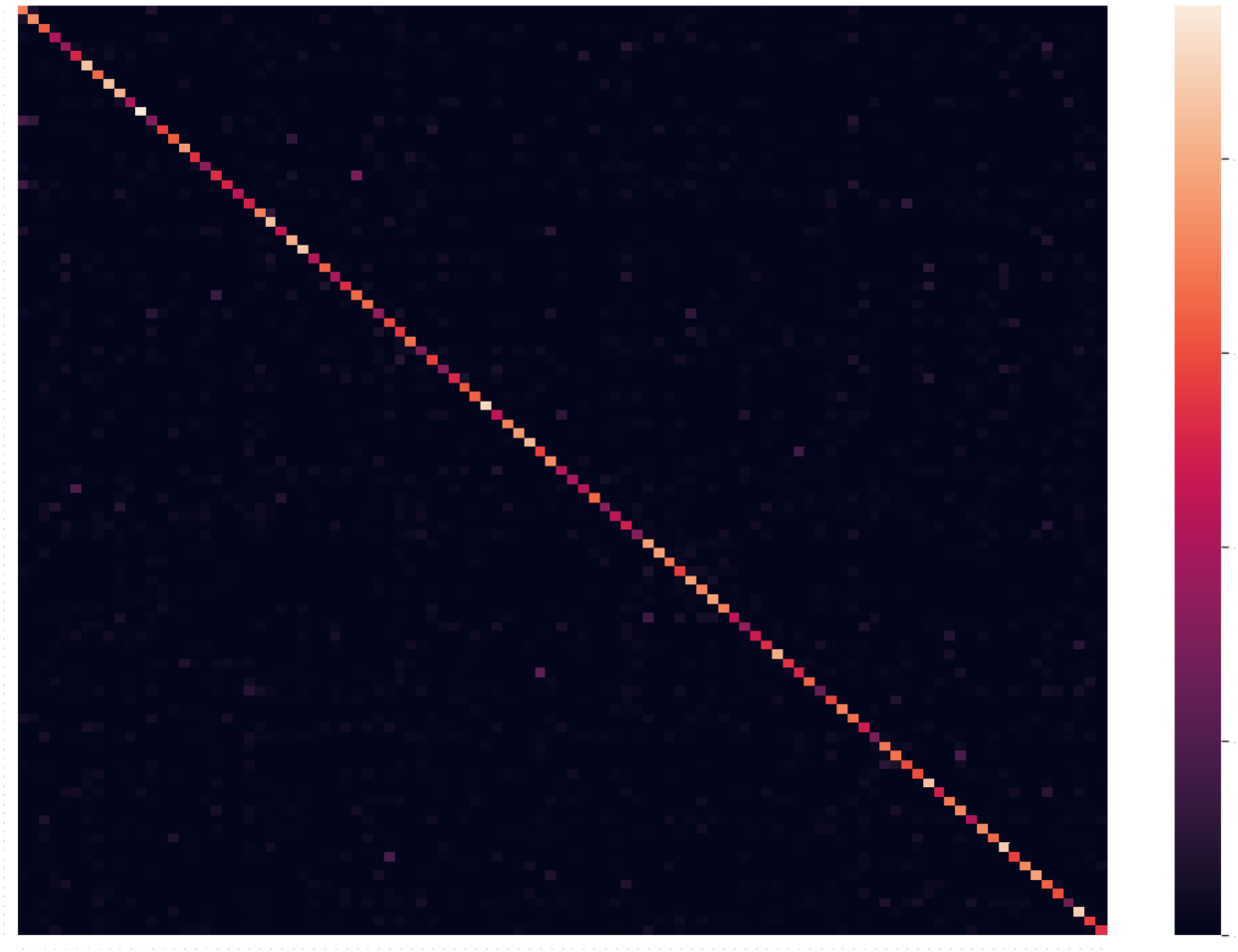
Please refer to the cnn3d_similarity.ipynb notebook for running the pipeline with extracted features.

kNN Accuracy

The class-wise accuracy using kNN with k=3 of 3D CNN features using the UCF-101 dataset is 89.7%. When the UCF-101 dataset is split into a test/train set using a 70/30 split similar to the 2D CNN feature based similarity, the accuracy drops to 74.7%.

While the k=3 accuracy for 3D CNN based features are lower than 2D based similarities, it is important to note that the 3D CNN features are capturing spatiotemporal information of the entire video instead of a single frame, and doing so in a vector of same size. A more complex 3D CNN model (such as DenseNet) will likely give better accuracy results.

Confusion Matrix



The confusion matrix shows that across 101 classes in the UCF-101 dataset, most classes are predicted with good accuracy, as shown by the diagonal bright line of squares in the matrix.

Looking at some of the most mischaracterized labels:

True Label	Predicted Label	% of error
Front Crawl	Breast Stroke	
Kayaking		
Band Marching		
Surfing		
Hammer Throw		

Table. 1. Misclassification error for KNN > 15 percent

True Label	Predicted Label	% error
FrontCrawl	BreastStroke	29.3
Kayaking	Rafting	22.9
BandMarching	MilitaryParade	19.2
Surfing	Skijet	18.6
HammerThrow	ThrowDiscus	17.6

It looks like some of the mislabels are with similar activities, such as BandMarching and MilitaryParade.

Dominant Colors (Color Level): An unsupervised color based approach using k-Means followed by KNN search

Algorithm

We perform feature reduction by creating histograms of the *k* dominant colors for each selected frame of the video. To do this, we perform k-means clustering on each image/frame using each RGB pixel as a datapoint. This creates *k* clusters (which we have set to 10) that each have their own cluster centroids. These centroids refer to the dominant color in that cluster and therefore give us the 10 dominant colors in each image. We 'weight' each dominant color by the size of the corresponding cluster and sort them according to cluster size.

Thus we find the *k* dominant colors by using k-means clustering.

Implementation

For our project, we have decided to leverage the GPU implementation provided by the **faiss** library that was created by Facebook Research. The chart Fig. 1. below compares the runtime with that of a standard CPU implementation provided by scikit-learn.

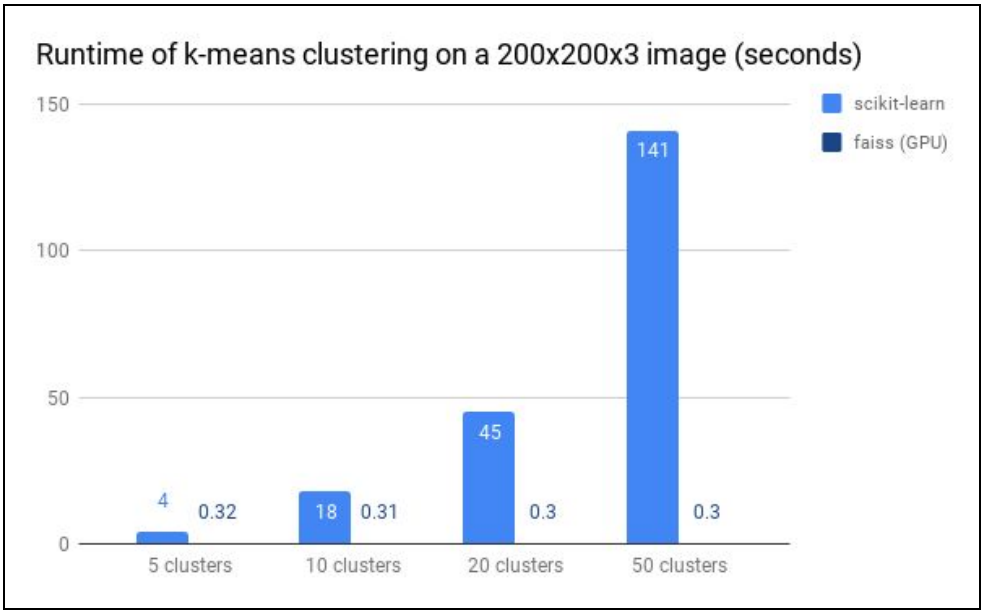


Fig. 1. Runtimes for the CPU and GPU k-means implementation.

As we can see, using a GPU gives a massive performance boost in this case from 13x to nearly 500x as the cluster size (k) increases from 5 to 50. We see sklearn takes exponentially more time and a GPU can definitely speed up this process by capitalizing on parallelism. Although the algorithms are not exactly the same, the performance and quality of results are comparable and the speedup is worth the very minor loss in accuracy.

PS: In all experiments, the runtime includes the time to load the image and cluster it.

Then we can perform a similarity search using these dominant colors to find similar images and by concatenating dominant colors we can find similar videos.

The dominant colors are then given weightage based on the size of the clusters and we get a 1x100 image as shown in Fig. 2. for each of the frames we cluster i.e. we cluster the resized image of size 200x200 and create a smaller image of size 100 (compressed to 0.25% of original) that contains its dominant colors in descending order of dominance (size of the cluster):

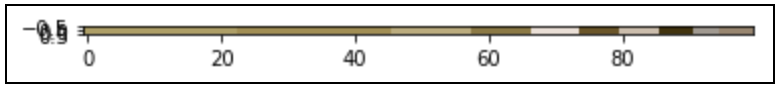


Fig. 2. K-dominant colors as extracted by using k-means

This process is repeated for *N* (20 for example) frames that are uniformly sampled from the entire video and a single image representation (called the **vid2img** representation hereafter) is created as shown in Fig. 3. (below).

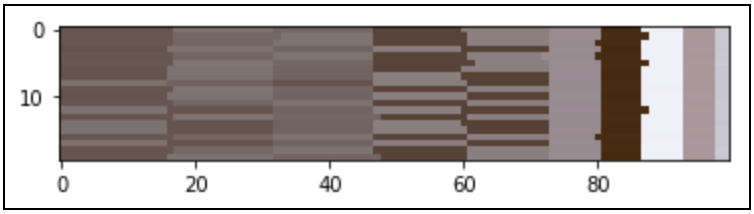


Fig. 3. The vid2img representation for a sample video

This compressed **vid2img** image representation is thereafter used to find similar images using using the KNN algorithm to search for the *k* similar videos for an input video. We will use the GPU implementation for this provided by facebook research's **faiss** library since we need to do this in near real-time.

We run the k-Means for 8 frames (as the median length of the videos in the UCF101 dataset is 8 seconds) that are uniformly distributed across each video in the UCF101 dataset. These are used to build a 8x100x3 (=2400 dimensional) vector representation (**vid2img)** for each video (for RGB channels) and store these in an HDF5 file.

The runtime of k-means as shown before was **0.2-0.3 seconds per frame**. This means each vid2img representation takes around **2-2.3 seconds per video** (including reading the video and splitting into frames).

The final KNN search for *k* similar videos takes **100 ms** for a single video query. The performance is measure by using a class-wise accuracy metric and we get a that **20%**.

As we can see, class-wise accuracy for this method was not that good. With a test accuracy of only 20 percent. However while looking at similar videos based on color we see that there are a lot of classes that have similar color backgrounds.

True Label	Predicted Label	% of error
Skiing	Horse Riding	32.7
Skiing	Skijet	44.1
Skiing	Swing	29.4

Table. 1. Misclassification error for KNN > 25 percent

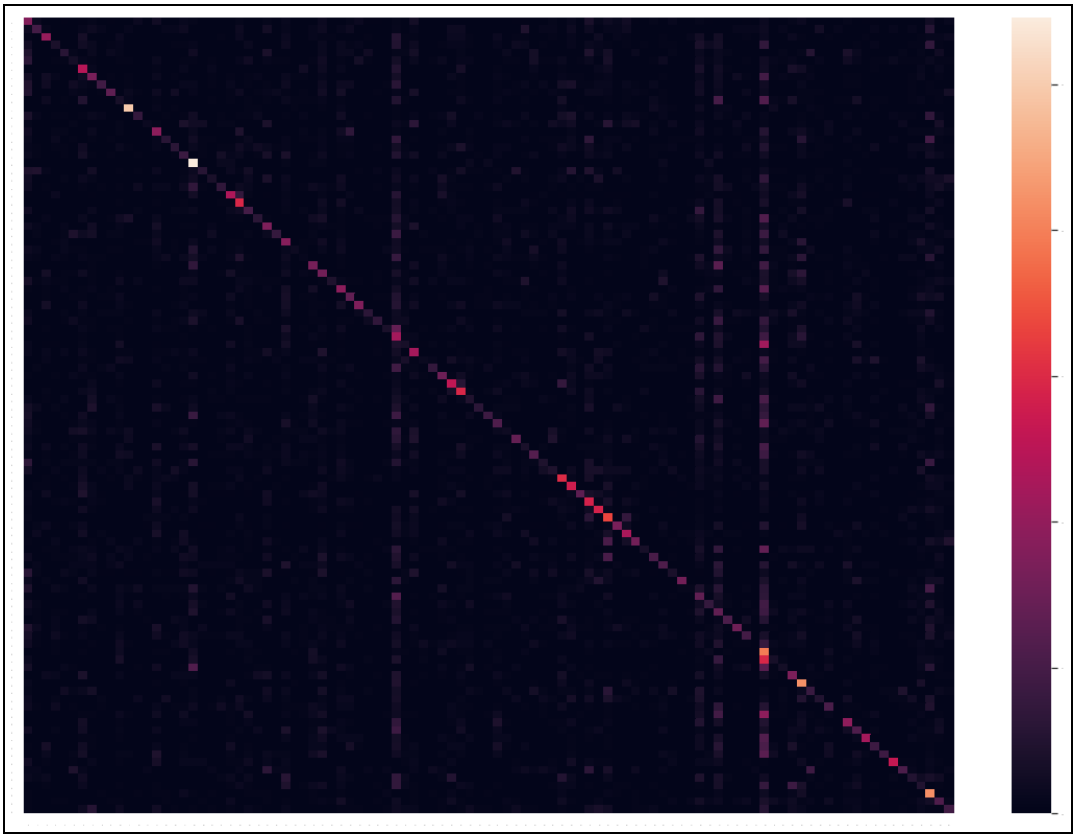


Fig. 4. Confusion matrix for KNN on Dominant Color Features

As we can see there may be some issues with this approach but it is able be capture some color features. For example: One of the swimming class had a test accuracy of 82%.

Audio Embeddings/Features (Audio Level): Using Audio features from video followed by KNN search

Algorithm

Audio data is tricky to work with. Google's Audioset project used a common method that is currently used to convert audio signals to images called log mel spectrograms. Then they run a special model called VGGish. This model is inspired by the VGGNet is used for this process.

The pre-trained model and pre-processing script called VGGish is provided by Google.

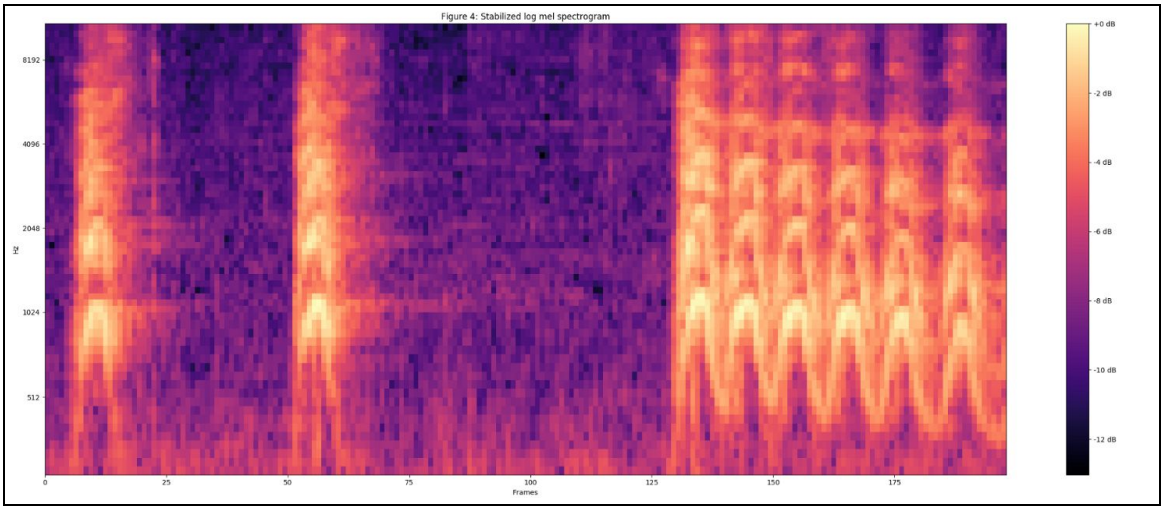


Fig. 7. A sample mel spectrogram

Implementation:

Steps for extracting audio features:

1. Extract audio signal and save in a wav (raw audio) file from the video (if it has an audio signal).
2. Subsample the audio signal at a fixed standard rate.
3. Build a log mel spectrogram after applying correction on it. (1 per second)
4. Pass the log mel spectrograms to the VGGish model and extract audio a **128** dimensional audio embedding.
5. Store these in an HDF5 file.

Only **51.3% (51/100 classes)** of the videos in the UCF dataset have an audio channel and we use only these for the KNN similarity search.

We repeat the process of transfer learning like before to extract audio features. Each set of audio embeddings (per second) that is extracted from the video takes around 1 second. This includes time to extract audio from video and then to extract audio embedding from the video. We store these audio embeddings in an HDF5 file.

After the extracting features using a VGGish (by removing the last layer of the neural network) and we run a KNN similarity search as shown in Fig. 4.

The VGGish audio features are then run in the GPU KNN implementation like before and we get K similar VGGish audio features. These K VGGish audio features are used to get the videos they correspond to and the duplicates are removed. The resultant similar videos are the one we are interested.

No. of Queries	Runtime (s)
1	0.16

10	0.16
100	0.16

Table 5. Runtime per query for KNN on VGGish audio features

Value of K	Runtime (s)
3	0.16
10	0.16
100	0.16
200	0.16

Table. 6. Runtime based on size of *k* for KNN on VGGish audio features

When using 100 percent of the data, we get a class-wise accuracy of **78%** on the UCF101 data. By taking a 70:30 split for train and test we still get a class-wise accuracy for test of **63.5%** on the UCF101 data.

The class-wise accuracy for UCF101 (51 out of 101 have audio channel) is then calculated and it comes to 63.5%.

PS: The class-wise accuracy for AudioSet is 47.3% (for 448 classes when assuming only first label).

True Label	Predicted Label	% of error
Kayaking	Balance Beam	11.1
Baby Crawling	Basketball Dunk	11.3
Front Crawl	Basketball Dunk	10.2
Boxing Punching Bag	Bowling	10

Table. 7. Misclassification error for KNN > 10 percent

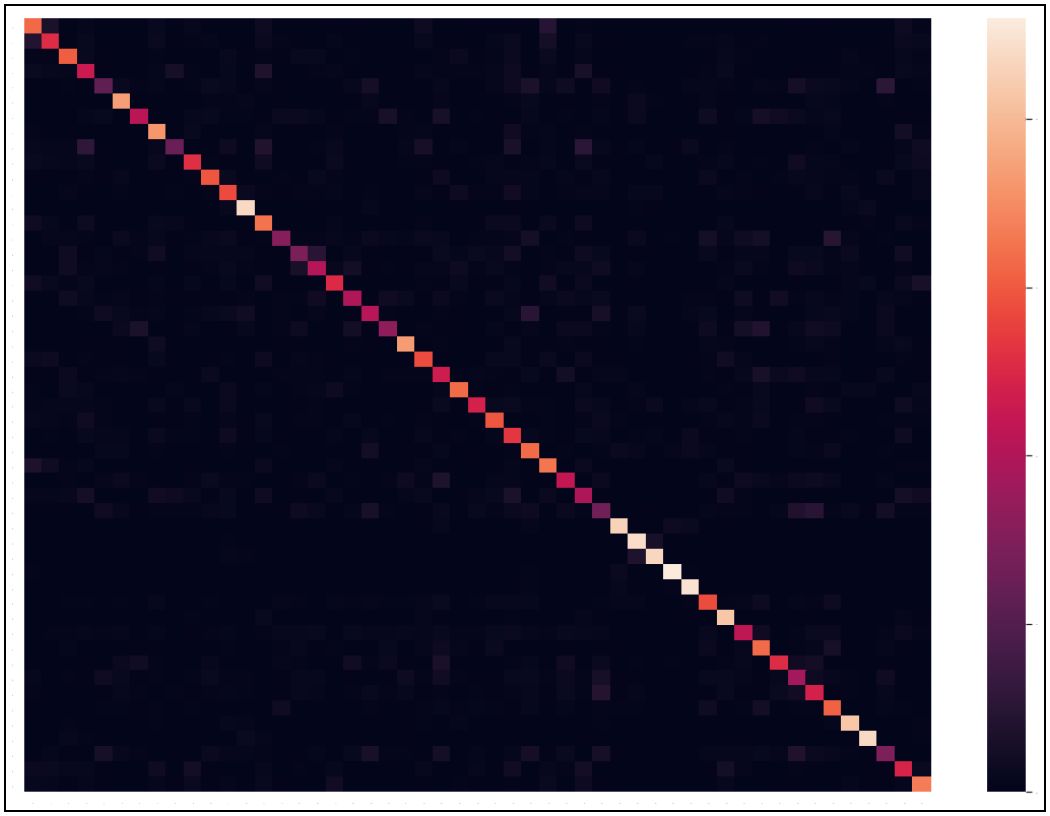
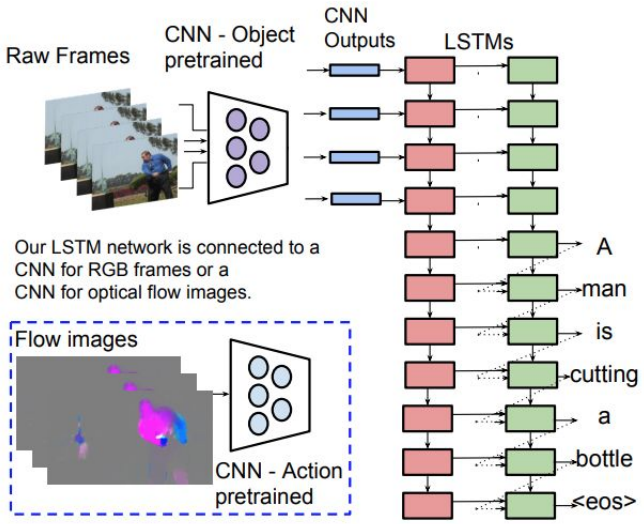


Fig. 8. Confusion matrix for KNN on Audio VGGish features

B. Supervised Approach (Nimisha)

Generate a caption/sentence describing every video followed by clustering these sentences to find the most similar videos. TLDR; Converting an Image domain to text domain and clustering that.

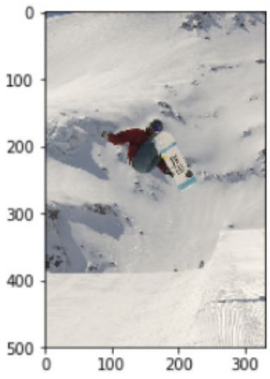
The network architectures that we implemented for the above approach follows the s2vt paper:



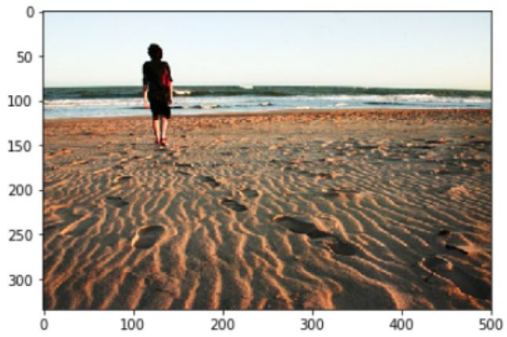
This sequence to sequence model works with the MSVD dataset described earlier. It basically uses the features from an image frame to generate text. The sequence of these image features are the input to an LSTM which will then produce a word at every time period. The training dataset for this approach has videos from sources such as youtube that already have descriptions. Therefore the dataset has the raw structure - <video clip, sentence> which is then translated into <[image features], [words]>

Here are some of the results of some of the videos in the dataset:

`Greedy: young boy wearing red shirt is standing in front of red plastic slide`



Greedy: man in red jacket snowboarding



Greedy: a boy is walking on the beach with the ocean .

After generating a sentence describing the test video, this generated sentence is then clustered with other sentences.

Quantitative evaluation of the models are performed using the METEOR metric which was originally proposed to evaluate machine translation results. The METEOR score is computed based on the alignment between a given hypothesis sentence and a set of candidate reference sentences. METEOR compares exact token matches, stemmed tokens, paraphrase matches, as well as semantically similar matches using WordNet synonyms.

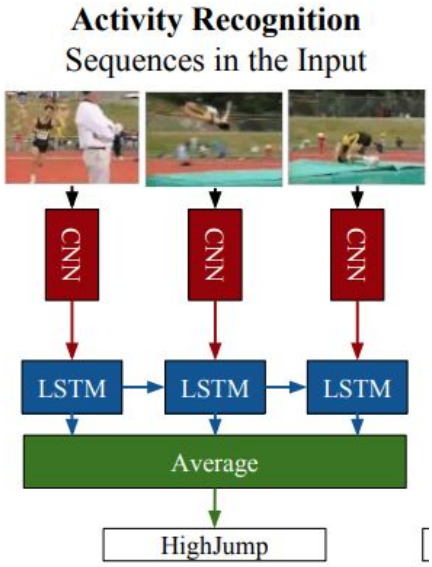
Hits and Misses

- What we observe is that this method captures various activities much better than just image feature clustering
- reduces memory footprint
- Does not retain context a whole lot because the clustering method being used for sentences is now just a word count preceded by synonym generator from nltk

To get some context, let's assume our repository is filled with clips of various games in a university. They could be a mix of basketball dunking shots, cheerleaders making various formations, the swim team winning a relay, etc. Our aim would be - given a clip of a gymnast

performing a split, find other videos of gymnasts performing in the university, among the same pool of videos. To do the above, we first manually classify the given pool of videos into some labels that make sense to us. In this case the labels could be the name of the sports themselves.
We used several methods to create classifiers to learn the labels of similar such datasets. Therefore, once we are giving a video of a gymnast and the classifier labels it correctly, we then pull up only gymnastics videos and use colour or frame clustering to find videos that are most similar to the one being considered.

The network architectures that we implemented for the above approach are inspired by the papers that follow architectures similar to the one shown below.



We explored many more papers than those that were proposed in the project proposal as we found that efficiencies improved with these newer models:

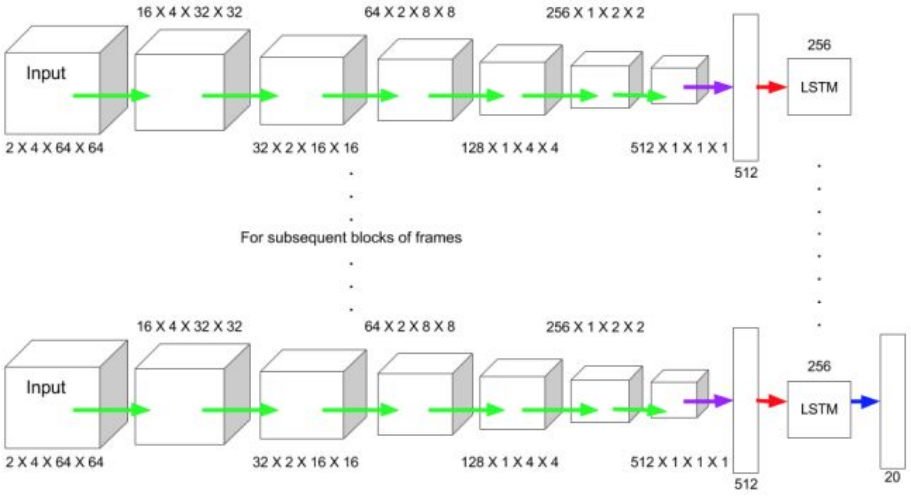
Method - 1: Frame wise classification with CNN

We first 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. Model performance was around 65% in video classification on the UCF101 dataset. This could be better, according to literature surveys of recent papers.

Method - 2 : Video feature extraction using pre-trained 3D CNN
This method uses modified PyTorch implementation of a traditional 2-dimensional convolutional neural network into a 3-dimensional neural network. The convolution and pooling layers are done using 3 dimensional blocks, and this model takes in as an input the entire video divided into 8 frames. The output is a feature vector that corresponds to a video with temporal aspects of the video features built in. The feature extraction on a 4 GPU virtual machine using ResNet-34 based

3D CNN takes about 12 hours. The accuracy and the performance of this method is still being evaluated on the UCF101 dataset.

Method - 3: Use a time-distributed CNN, passing the features to an RNN, in one network
We now take into consideration the temporal aspects of the video. We now have a CNN layer for every time step that we feed into the RNN as shown below.



This model cannot use pre-computed image features and hence training time is enormously large! Takes a few days and the classification accuracy was very poor, given that hyper parameter tuning could be done on this. It was almost 20% which is worse than a random classifier! Onto the next method.

Method-4 : Pre-computed CNN features passed into an RNN

This method uses VGG16/InceptionV3 to extract features from every frame of the video (we have sampled only 40 to accommodate for memory constraints) and then concatenate this into a 40 item vector which is then passed into an RNN. For the RNN part, we have Gated Recurrent Units which are shallow in nature. We switched to using GRUs instead of RNNs or LSTMs because the network was experiencing a stagnancy in loss which is a common issue while using RNNs. This network gave about 83% accuracy on the UCF101 dataset in correctly recognising an activity. However, the misclassifications were on videos of vastly different classes so we wonder what noise the model is picking up that it's misclassifying into vastly varied classes.

Hits and Misses :

- The above models work well for videos that have a single human activity in them

- They would not work well for a video that has multiple actions as in that case labelling a video with one activity makes no sense.
- We ought to move to sentences or labels to identify multiple continued activities.
- This is why we took the next approach.

Summary of Results

Feature Extraction + Unsupervised Models

Model/Approach	Top-3 Class-wise Accuracy on UCF-101 Dataset	UCF-101 Dataset Feature Extraction time	Per Video Feature Extraction Time (s)	KNN Query Time (s)
ResNet Features + KNN	91.6%	3 days	~1	0.2
3D ResNet Features + KNN	74.7%	8 hours	~5	0.2
Dominant Colors + KNN	20.0%	11 hours	~7	0.3
VGGish Audio + KNN	63.5%	8 hours	~2.5	0.2

Caption generation model

Model/Approach	METEOR score	Training time	Prediction Time	Dataset
s2vt	~24 %	~ 14 hours	~ 1 hour	MSVD, ~ 40GB

Schedule Review

Completed as part of schedule	Additional methods explored
Papers implemented as part of exploratory analysis.	- Video Captioning and clustering - ResNet (CNN) feature based KNN similarity - Audio feature based KNN similarity - Merged (all 4) feature based KNN similarity

-Long-term Recurrent Convolutional Networks for Visual Recognition and Description , Arxiv Link -Learning Spatiotemporal Features with 3D Convolutional Network, Arxiv Link -Temporal 3D ConvNets: New Architecture and Transfer Learning for Video Classification, Arxiv Link - KNN with Dominant Color features	+ built demo - Clustering sentences generated with retaining context

Challenges faced

The major time consuming parts of this project have been:

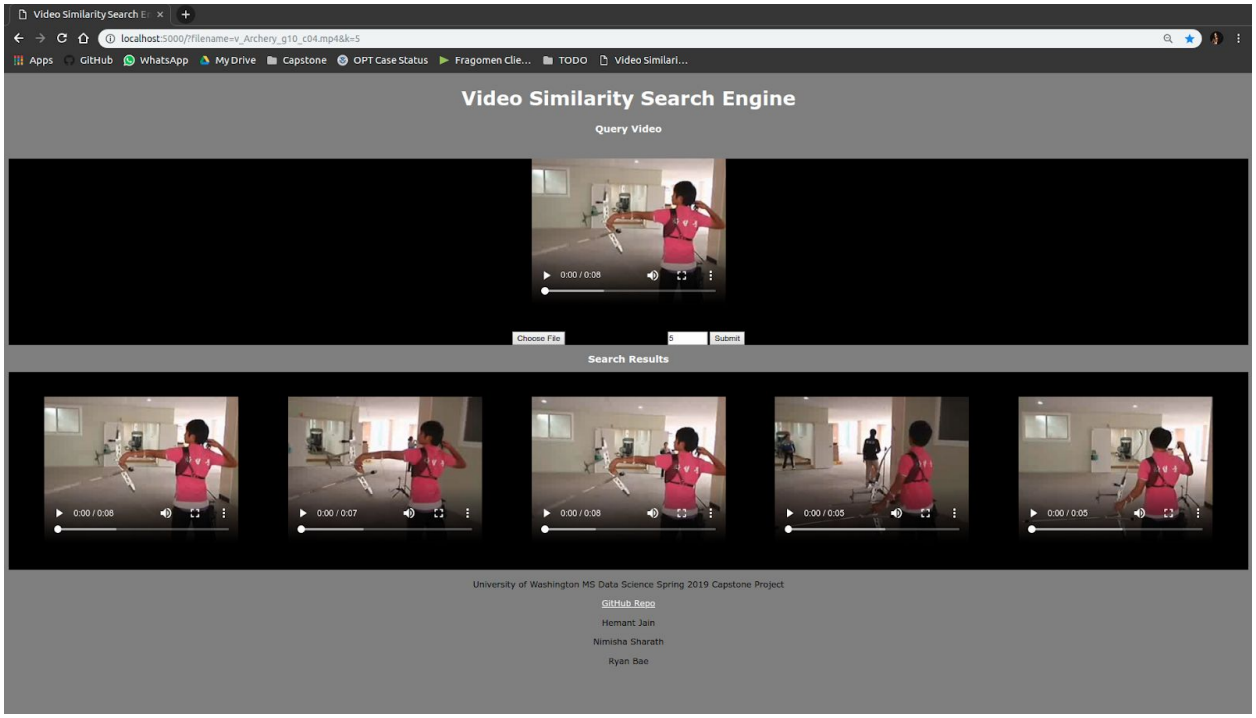
- Feature extraction from the videos : solved with Cloud credits
- Memory blow up during training : solved with access to larger machines on AWS
- Lack of labelled datasets from Adobe : solved by using publicly available datasets such as UCF101, MSVD etc.

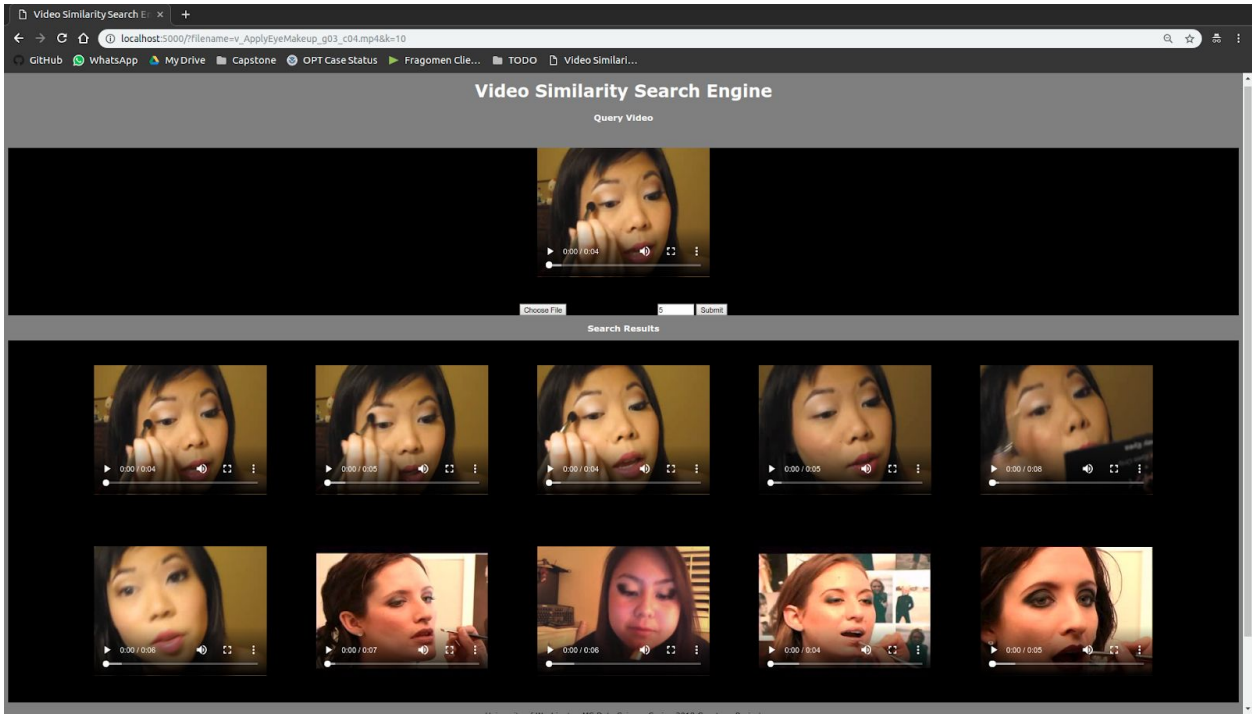
Conclusions:

The approaches we explored allowed us to have a multi-modal approach to solving the problem of searching for Similar Videos. With the different signals being Objects, Actions, Audio and Color levels. Apart from the fact that this approach can extend for any number of classes and does not require manual annotation, the method allows us to perform the search in real-time.

The supervised approaches allowed a method to build a model that is not just easily interpretable and also work for videos that are longer and contact multiple scenes/ complex activities. Both approaches are great for different reasons and solve two different issues. The project was a success.

Live Demo





Hardware used for Demo:

- **Processor:** 7th Gen Intel i7
- **GPU:** GTX 1060 Ti - 6GB
- **RAM:** 16 GB

Process requirements:

- **CPU Consumption:** 15% Percent
- **RAM:** 3.6 GB
- **GPU Memory Consumption -**
  2700 MB of GPU Memory (For new videos)
  900 MB of GB Memory (For old videos)

References:

1.