CSE 515

Multimedia and Web Databases
Group 6

Team

1. Aryan Patel (apate294@asu.edu)

2. Tejas Ajay Parse (tparse@asu.edu)

3. Om Patel (opatel14@asu.edu)

4. Tanishque Zaware (tzaware@asu.edu)

Abstract

The main focus for this phase is classification tasks and relevance feedback. We work on finding the inherent dimensionality of each labels, finding significant clusters for both videos as well as clusters using techniques like KMeans and Spectral Clustering , classifying a video query into most similar labels using K – Nearest Neighbors classifier and Support Vector Machine, implementing Locality Sensitive Hashing to create index structures and using them for similar video retrieval, as well as taking user feedback into account in the form of "liked" and "disliked" videos to provide a new set of ranked videos.

Keywords

Multimedia Retrieval, KNN, SVM, Decision Tree, Relevance Feedback.

Introduction

- **Terminology**
    1. **Cluster**

       A group of data points that are similar to each other in a feature space. Clustering is used to identify inherent groupings in data.
    2. **KMeans**

       A clustering algorithm that partitions data into a predefined number of clusters (kkk) by minimizing the variance within each cluster. It iteratively assigns data points to the nearest cluster centroid and updates the centroids until convergence.
    3. **Spectral Clustering**

       A clustering technique that uses the eigenvalues of a similarity matrix (derived from the data) to perform dimensionality reduction before applying a clustering algorithm. It is particularly effective for data with complex, non-convex structures.
    4. **K-Nearest Neighbors (KNN)**

       A classification algorithm that assigns a label to a query based on the majority label of its k-nearest neighbors in the feature space.
    5. **Support Vector Machine (SVM)**

       A supervised machine learning algorithm that finds a hyperplane in a high-dimensional space to classify data points.
    6. **Decision Tree**

       A classification method that splits data into subsets based on feature value thresholds, forming a tree structure to reach a decision.
    7. **Locality Sensitive Hashing (LSH)**

       A technique for approximate nearest neighbor search in high-dimensional

data, designed to hash similar items into the same bucket with high probability.

8. **Relevance Feedback**

   A mechanism where users provide feedback (e.g., "liked" or "disliked" videos) to improve the quality of retrieved results in subsequent iterations.

- **Goal Description**

  The goal of this phase can be broadly described as clustering, ranked retrieval, and classification. Task0 focuses on finding the inherent dimensionality of each label in the latent space, task 1 focuses on finding c most significant clusters for videos as well as labels using Kmeans and Spectral Clustering, task 2 deals with the classification of a video query into m similar labels using KNN and SVM, task 3 works on implementing a Locality Sensitive Hashing (LSH) tool for Euclidean Distance and then using it to find similar videos to a query video. Lastly, Task4 uses the result of task 3 (similar videos) and asks the user for his/her feedback in the for of “like” and “dislike” based on which we give a new set of similar videos using KNN based and decision tree based algorithm.

- **Assumptions**

  We are assuming that while executing the project, all the paths will be modified as required and the provided latent models as well as the pickle files for the trained SVM Model will be downloaded and put in the working directory.

Proposed Solution

The three latent models we have selected are :

- Avgpool reduced with Kmeans to 300 dimensions
- Layer4 reduced with PCA to 300 dimensions
- Col Hist reduced with SVD to 300 dimensions

Please note that unless otherwise stated, the distant metric used in all tasks wherever applicable is cosine distance because it works well in higher dimensional space

Task 0

In task 0, Our aim is to find the inherent dimensionality associated with each unique label under target videos. First, we find the feature set of the data using one of the latent models selected by our group, in this case feature set with 300 dimensions obtained by applying PCA on layer4 is used and the reason for that is because PCA helps conserve variance and we will obtain inherent dimensionality using variance further in the process. Now feature matrix for a label under target videos is loaded using dataset, layer4_pca(300) csv file and VideoID_Mapping csv file. Here PCA is used since it preserves the variance in a dataset and to find inherent dimensions of data we need features which shows maximum variance and hence prevent redundancy by avoiding similar features. Covariance matrix is calculated using numpy and feature matrix obtained in the first step. Next we calculate eigen values and sort them in descending order using numpy library and sorting. Finally we calculate cumulative eigen values and filter them with threshold of 95%, This method is called Variance Explained and the reason for its use is so we can keep features which show 95% of the variance in the feature set and remove the redundant features from the set. Setting the threshold to 95% helps preserving import information in the data, meanwhile the remaining 5% may include minor details or noise that does not have significant impact on the overall patterns. We repeat this steps for each label to get the inherent dimensionality for all target video labels.

Task 1

The goal of Task 1a is to implement a program that identifies the most significant clusters of target videos based on their labels using spectral clustering. To begin with, the videos are grouped by their labels, ensuring that all videos belonging to the same label are processed together. The first step in this approach involves preprocessing the VideoID_Mapping file, which serves as a mapping between video IDs, video names, and

their corresponding file paths. This mapping is crucial as it allows for the retrieval of video thumbnails later in the process when visualizations are generated.

For each label, the next step involves working with the feature sets associated with the videos. The videos for a given label are analyzed by constructing an adjacency matrix based on pairwise distances between the videos. To compute these distances, we use the Euclidean distance metric, which is effective for measuring distances in vector space. Specifically, for a given set of videos (let's assume 40 videos for illustration), we calculate the distance between each video and every other video, resulting in a total of 1560 distance values (since each video is compared with 39 others). The distances are then used to determine a threshold value, which is set as the k-th percentile (1 in our case) among all the pairwise distances. This threshold serves to define which videos are considered "close enough" to be connected by an edge in the adjacency matrix.

Once the adjacency matrix is constructed, spectral clustering is performed to identify the video clusters. In this implementation of spectral clustering with KMeans, we first compute the Laplacian matrix of the graph from its adjacency matrix. The Laplacian captures the structural properties of the graph, and its eigenvectors provide a lower-dimensional representation of the graph's nodes. We select the smallest num_clusters eigenvectors (excluding the trivial first eigenvector) to capture the most relevant features for clustering. These eigenvectors are then used as the input to the KMeans algorithm, which partitions the nodes into clusters based on their positions in this new feature space. To implement k-means clustering from scratch, we initialize k random centroids from the feature vectors and assign each vector to the nearest centroid using Euclidean distance. This distance metric is effective as it measures the straight-line proximity between points, making it well-suited for clustering when the magnitude and spatial relationships of feature vectors are important. After assigning labels to the centroids, we recompute the centroids for each cluster by taking the mean of all the feature vectors that belong to that cluster. The process of reassigning clusters and recomputing centroids is repeated iteratively until the centroids

no longer change, indicating that the algorithm has converged. This iterative process allows the algorithm to find stable clusters of labels based on their feature vectors.

After the clustering process is complete, the resulting clusters are visualized in two ways. First, the clusters are visualized as point clouds in a 2-dimensional space using Multidimensional Scaling (MDS). We have used a library function to utilize MDS. The MDS algorithm is used to reduce the dimensionality of the feature space to two dimensions, allowing for a clear visual representation of the clusters. The videos in each cluster are then represented by points in the 2D MDS space, with each cluster assigned a unique color for distinction.

The second visualization involves creating a group of video thumbnails for each cluster. Each cluster's thumbnails are gathered and displayed together, providing a visual representation of the content within each cluster. Thumbnail of any given video is the middle frame of that video.

In Task 1b, the objective is to implement a program that computes the most significant label clusters for target videos using k-means clustering. Unlike Task 1a, where clustering was done on individual videos, this task focuses on clustering the labels themselves, which represent groups of videos. To achieve this, we first need to create a representative feature vector for each label. Since each label corresponds to multiple videos, the feature set for each label is derived by taking the mean of the feature vectors for all the videos under that label. This ensures that every feature has equal weight in defining the label's representative cluster vector.

Once the representative feature vectors for each label are computed, we proceed to perform k-means clustering on these label vectors. To implement k-means clustering from scratch, we begin by initializing k random centroids. These centroids are selected randomly from the feature vectors of the labels. The next step is to assign each label’s feature vector to the closest centroid. The closeness is determined using the cosine distance, which is well-suited for clustering in high-dimensional spaces where the magnitude of the vectors is

less important than their orientation. The cosine distance between two vectors is computed as the dot product of the vectors divided by the product of their magnitudes. This step is crucial as cosine distance allows us to capture the angular similarity between feature vectors, which is especially useful when working with high-dimensional data, where distances can often be very large and nearly uniform.

After assigning labels to the centroids, we recompute the centroids for each cluster by taking the mean of all the feature vectors that belong to that cluster. The process of reassigning clusters and recomputing centroids is repeated iteratively until the centroids no longer change, indicating that the algorithm has converged. This iterative process allows the algorithm to find stable clusters of labels based on their feature vectors.

Once the k-means algorithm has converged and the label clusters have been determined, we proceed to visualize the results. The first visualization involves displaying the clusters in textual format, where each cluster is represented as a group of labels. This provides a straightforward view of which labels are grouped together, making it easier to interpret the clustering results. The second visualization involves plotting the clusters in a 2-dimensional space using Multidimensional Scaling (MDS). Since the feature vectors are high-dimensional, we use MDS to reduce the dimensions to two, allowing us to visualize the clusters in a 2D space.

Task 2 involves classifying a query video id (even or odd) into m most similar target video labels using either KNN or SVM.

KNN Implementation: When a user enters a query video ID (can be odd numbered target video or non target video), we find the cosine distance between the query features and all the rest of the features of even target videos in the feature space. We use cosine distance as it works well in higher dimensional feature spaces. Only even target videos are considered since we have stored labels only for the even target videos. The distances are

stored in a list of tuples where each tuple contains (distance, label). This list is then sorted based on the distance value in ascending order. Now, using the top k elements in the list, we count the occurrence of the labels and store them as a list of similar format (label, vote). This list is sorted based on the value of votes and we extract and print the top 'm' labels. Also, if the query video id is an odd target video id, we also print the accuracy of the classifier. In case of odd non target video id, the above process remains same but it should be noted that the result will be similar *target video labels* for the corresponding non target video.

SVM Implementation: To implement a multi class SVM, I have trained a binary svm for each of the class. For each class label, I consider that class as +1 while the rest of the class as -1 and find a separating hyperplane for the classes. I have implemented a soft margin svm to handle cases where the data is not linearly separable. and hence an error or loss function is also introduced. I have used the hinge loss function for this task which is defined as follows:

$$L = \frac{1}{n}\sum_{i=1}^{n} \max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + b)) + \frac{\lambda}{2}|\mathbf{w}|^2$$

Where, yi = true label (+1+1+1 or −1-1−1) of sample i.
xi = the feature vector of sample i.
w and b are the weights and bias of the model.
λ = regularization parameter that controls overfitting by penalizing large weights.

The hinge loss ensures that points correctly classified with a margin greater than 1 contribute no loss, while misclassified points or points within the margin contribute a positive loss. This works well with our SVM implementation. Each binary SVM is trained using gradient descent. The weights w and bias b are updated iteratively to minimize the hinge loss. A regularization term $\lambda\|w\|^2$ is added to prevent overfitting and ensure generalization. Once trained, the decision function f(x) = w*x + b is used to compute the

distance of a sample from the hyperplane. This score indicates the confidence of classification for a given class. We have also used early stopping technique using patience and threshold to stop the training process early if no significant improvement is found. If the improvement in loss is below a threshold for a specified number of epochs (patience), training halts early.

Now, for multiclass prediction, the decision functions of all binary SVMs are evaluated for a given sample to get decision score for each of the target labels. So, we will get a list of 24 decision scores with their labels. We sort these scores in descending order and use the top m scores to get the most similar m labels. We take the highest scores since the decision score signify the distance of the point from the hyperplane and the farther away it is, the more confidence we have in the classification. We store this trained model for all 3 latent spaces as pickle files for future uses.

Hence, when a user inputs a query video id (odd target or non-target), we load the trained model and calculate the decision scores of all the 24 target video labels for this query video and select m labels with highest decision scores to output to user. If the query was an odd target video id, we also show the classifier accuracy which is defined below.

Please Note: The hyper parameters (learning rate, no. of iterations, lambda and so on) used to train the SVM (both binary and multi class) were selected through a trial-and-error process. We performed a grid search technique with multiple values for each hyper parameter and trained the SVM for all possible combinations of the hyperparameters one by one. The hyperparameter set that gave us the highest accuracy was then selected.

They were :

(for binary svm)

learning_rate=0.001, lambda_param=0.01, n_iters=2000, early_stopping_patience=10, early_stopping_threshold=0.001

(for multiclass)

learning_rate=0.00001, lambda_param=0.001, n_iters=4000, early_stopping_patience=10, early_stopping_threshold=0.001

In both above implementations (KNN and SVM), the accuracy of the classifier is computed based on the following formula :

$$accuracy\_classifier(v_i, m) = \frac{m - rank_i + 1}{m}$$

Where m – (user specified) no. of similar labels to be extracted

$Rank_i$ = rank of the true label of query video

Task 3 involves implementing an efficient video similarity search system using Locality Sensitive Hashing (LSH). It allows for querying a specific video and finding the most similar videos based on feature vectors extracted using various models.

First of all the extracted features stored in csv files are processed during the initialization phase. Next, for Hashing and Planes the LSH structure was initialized with L layers and h random planes per layer which were taken from user input. Each layer contains hash tables to group videos with similar hash signatures. The hash vector for a video is calculated by projecting its feature vector onto the random planes. For each video, hash signatures are calculated across all L layers and stored in the corresponding hash tables.

The videos in the same hash buckets are retrieved as candidates and the final list of candidates is sorted based on cosine similarity to the query video, and the top N results are returned. Here the overall candidates are the total number of vectors considered before deduplication and the unique candidates are the distinct video vectors retrieved across all the layers.

Finally, we have visualized these query video along with its similar videos through thumbnail and taken the frames from middle of the video to avoid the blur or irrelevant portion of the video.

Task 4

Task 4 will take the result of similar videos of task3b and ask user for feedback on those videos. The user can either “like” a video or “dislike” it. Liked videos are considered relevant to the user while disliked videos are considered irrelevant. We incorporate this user feedback and produce a new set of similar videos.

Decision Tree : Once the results are displayed based on a query video, the user provides a feedback (1 for like and 0 for dislike) by marking videos relevant or irrelevant and we store them in lists. Further we extract the features of both relevant and irrelevant and store them in list along with assigning the labels (1 for relevant and 0 for irrelevant). A decision tree is built recursively based on this labeled data and the tree splits the data based on feature threshold (in this case considered as mean) to maximize the separation between the relevant and irrelevant videos. For making it more effective we have used Gini Impurity to determine the best split that divides the dataset into subsets where each subset is as pure (homogenous) as possible.

$$Gini(D) = 1 - \sum_{i=1}^{k} p_i^2$$

Here p_i represents proportion of samples belonging to class i

G=0 represents pure nodes

Weighted Gini=(Left Size×Left Gini+Right Size×Right Gini)/ Total Size

Sizes here are the number of left or right labels.

We keep a track of feature and threshold with the lowest Gini Impurity. The splitting stops when the maximum depth is reached or all the videos have same labels. Using the decision tree to score all the feature vectors and again we rank them by relevance scores predicted by the tree in reverse. Further using these we visualize the top required number of videos to be retrieved.

KNN : To incorporate user feedback here, we have used the method of query rewriting. The liked and disliked videos are stored in lists named relevant and irrelevant. We define variable alpha as 1 / m, where m is the no. of videos to retrieve. We then take the sum of features of all videos in relevant and irrelevant set and store them in 2 separate variables. The query is rewritten so that the new query is closer to the relevant video objects and away from the irrelevant video objects. This is done by adding the original query to the product of alpha and sum of relevant features to bring the query close to relevant videos, and subtracting the product of irrelevant sum and alpha to bring the query away from irrelevant videos. This new query is passed to either the knn function of decision tree function based on the choice of the user and a new set of results are produced.
The implementation of KNN here is similar to the one in task 2 . The main change between the two implementations is that in task 2 we were predicting labels while here we are predicting the videos. Also, instead of looking at only the even target videos, we now look at the whole feature space to find the similar videos. The distance metric remains the same – cosine, because it works well with high dimensional spaces. To do this, based on the value of k decided by the user, we take the k nearest neighbors to our query (by computing the cosine distance and sorting) and compute the distance between these neighbors and the query. The result is stored and sorted based on the distance and the top m specified videos are extracted.

Interface Specifications

Please refer to the provided readme file for the interface specifications

System requirements and Execution Instructions

The code was run with the following version of python and its libraries :

Python : 3.10

Numpy: 1.23.2

Torch: 2.3.1

Torchvision: 0.20.0

Opencv-python: 4.10.0

Scipy: 1.13.1

Scikit-learn: 1.5.0

Pandas: 2.2.2

(versions of libraries used)

Detailed execution instructions for each task are provided in the Readme file present in Code Zip File.

Outputs and Discussions

Task 0

Feature set:

300 dimensions obtained by applying PCA on layer 4.

```
Inherent dimensionality of each label:

brush_hair: 53
cartwheel: 56
catch: 51
chew: 50
clap: 55
climb: 55
climb_stairs: 53
golf: 57
handstand: 59
hit: 63
hug: 60
jump: 72
kick: 68
kick_ball: 69
laugh: 55
shoot_ball: 50
shoot_bow: 44
shoot_gun: 52
sit: 73
situp: 35
smile: 59
smoke: 52
somersault: 71
stand: 77
```

We used csv file containing 300 dimensions for all labels and the output indicates inherent dimensionality for each label computed after applying PCA and 95% variance explained. For most labels the inherent dimensionality is between 55 to 60 and the reason for that is because videos under these labels do not need a lot of details to differentiate like in climb

or shoot ball the important information is within the body movements and hand actions/ ball throws and are enough to distinguish the action, Meanwhile for some labels like stand the inherent dimensionality is 77 and the reason is because the body movement may not be enough to identify the action but some environmental details are also required such as shifting of chair and table.

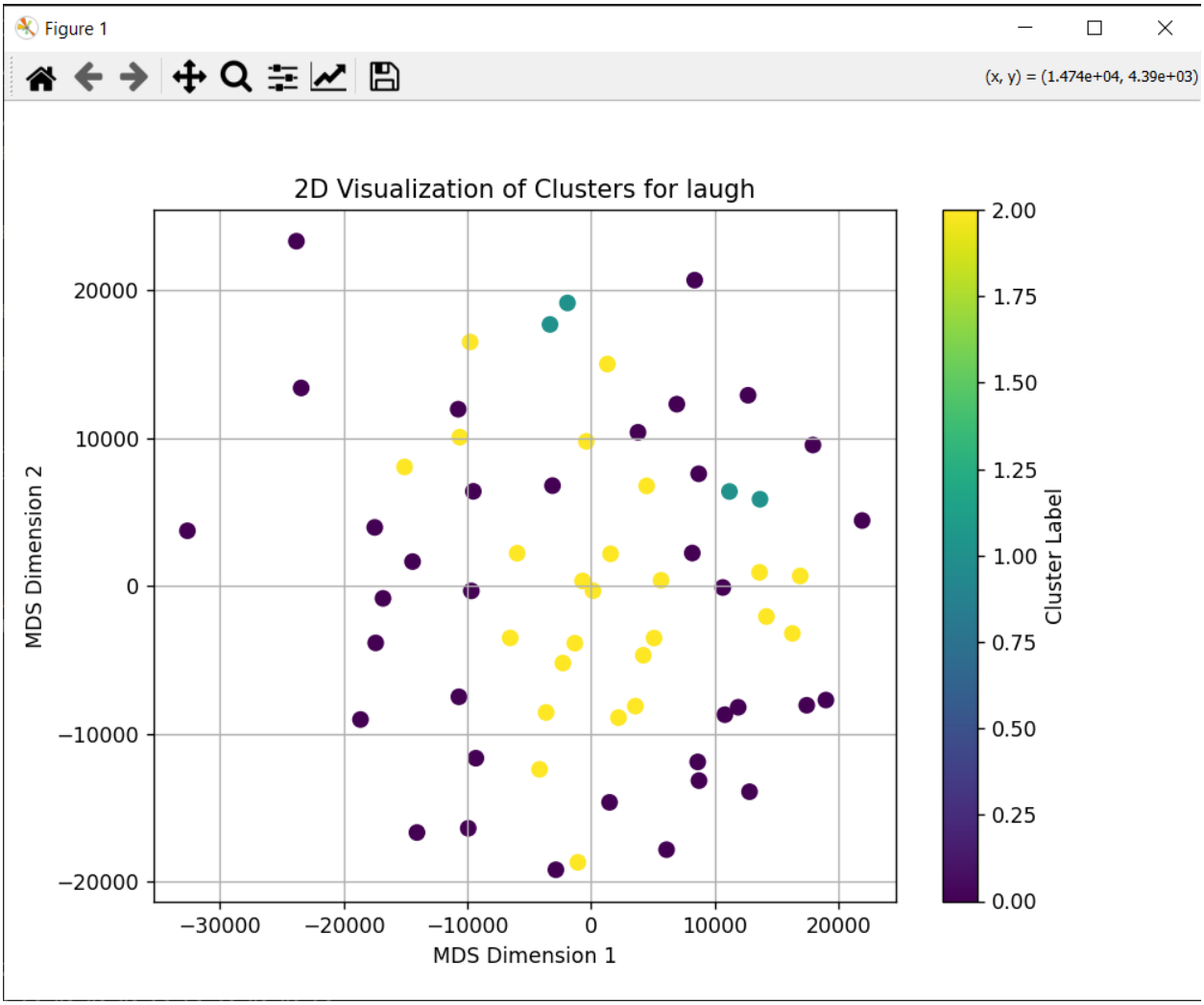
Task1a

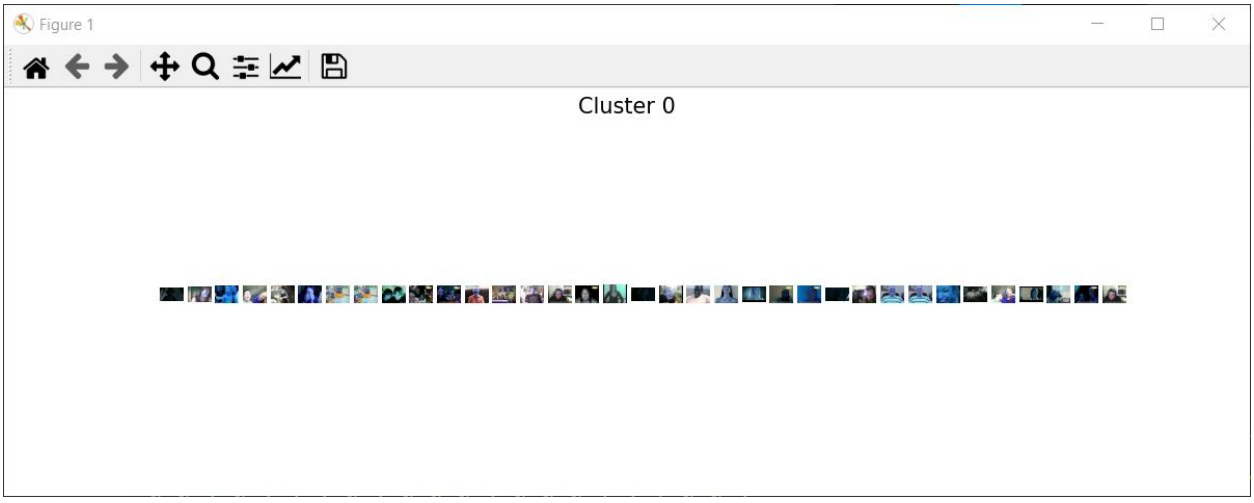
Input1:

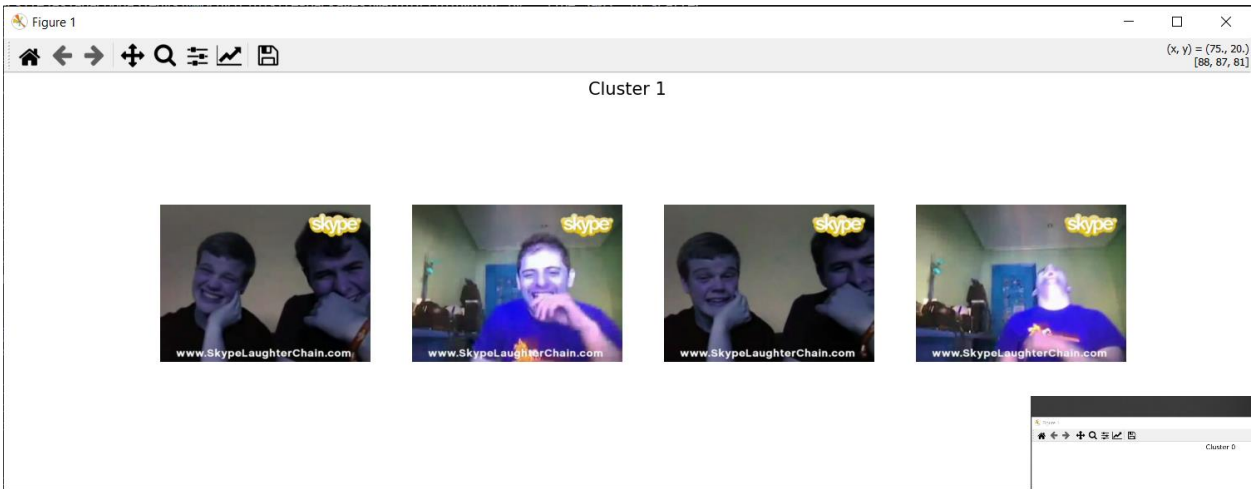
Colhist + SVD + 300

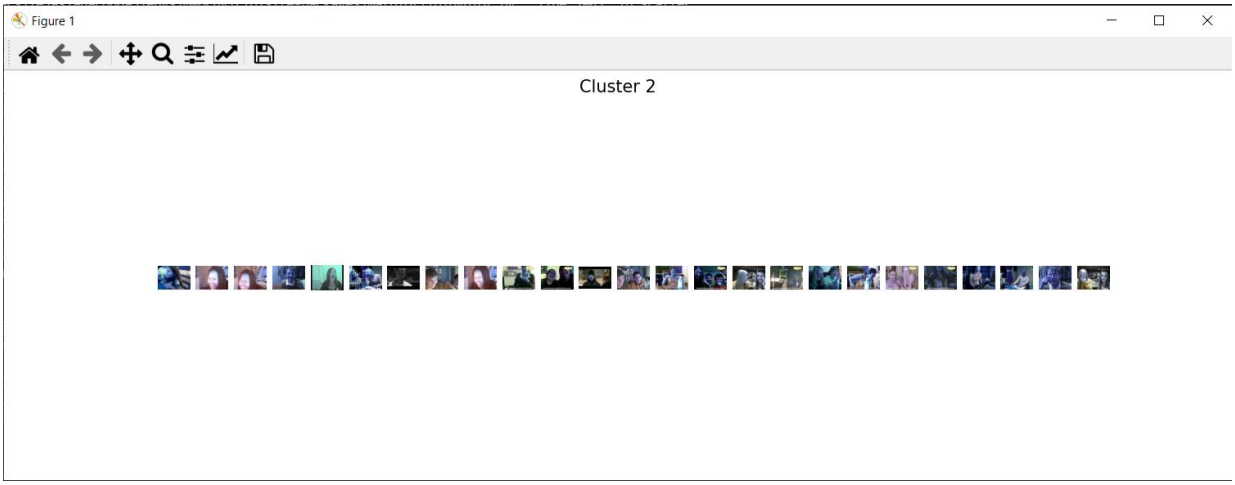
3 clusters

Below is Cluster of laugh videos









Input2:

layer4 + PCA + 300

5 clusters for label cartwheel

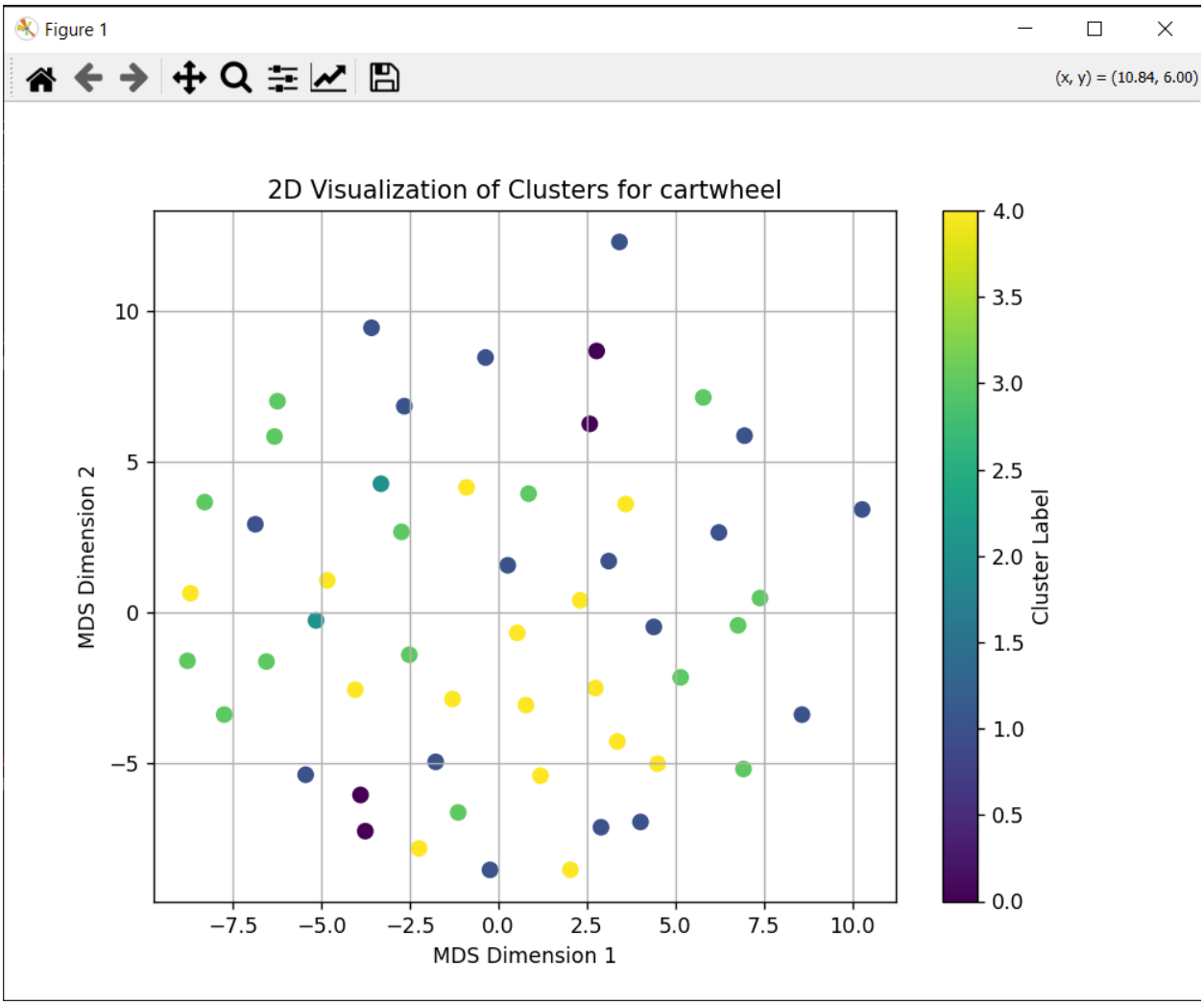


Figure 1
(x, y) = (10.84, 6.00)
2D Visualization of Clusters for cartwheel
MDS Dimension 2
MDS Dimension 1
Cluster Label
10
5
0
−5
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0

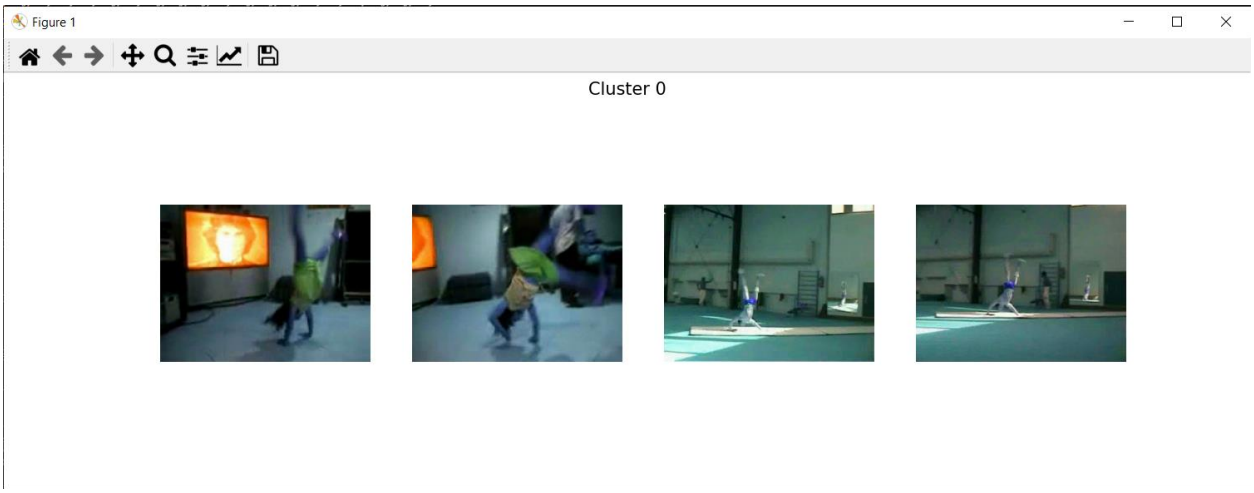


Figure 1
Cluster 0

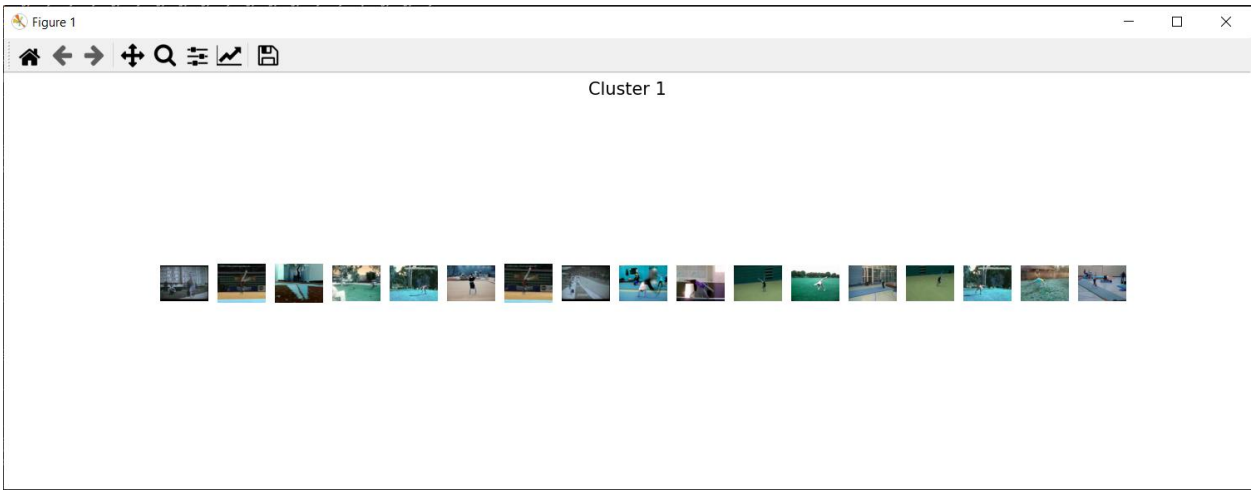
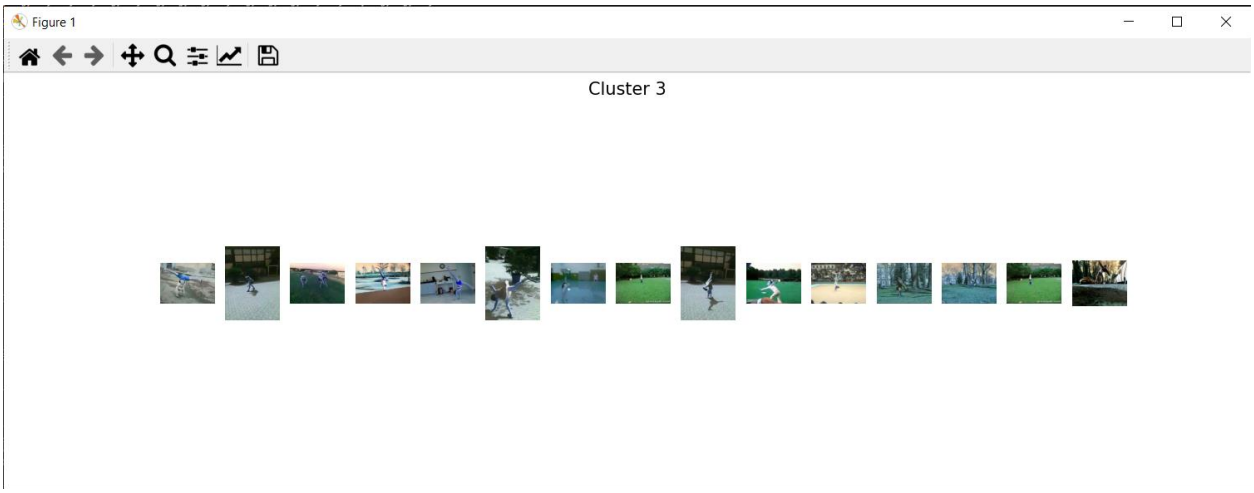
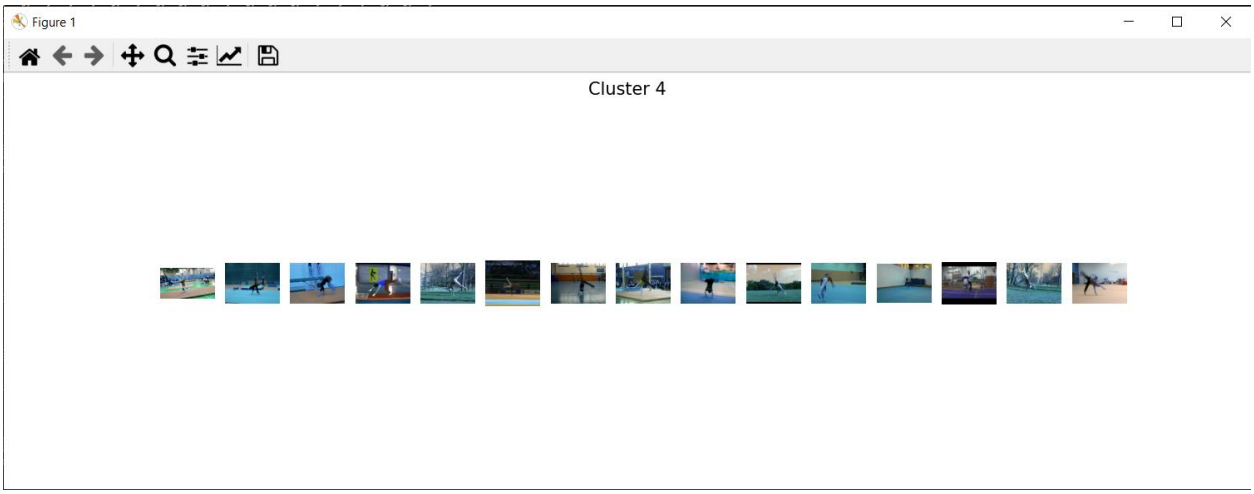


Figure 1
Cluster 1


Figure 1
Cluster 2





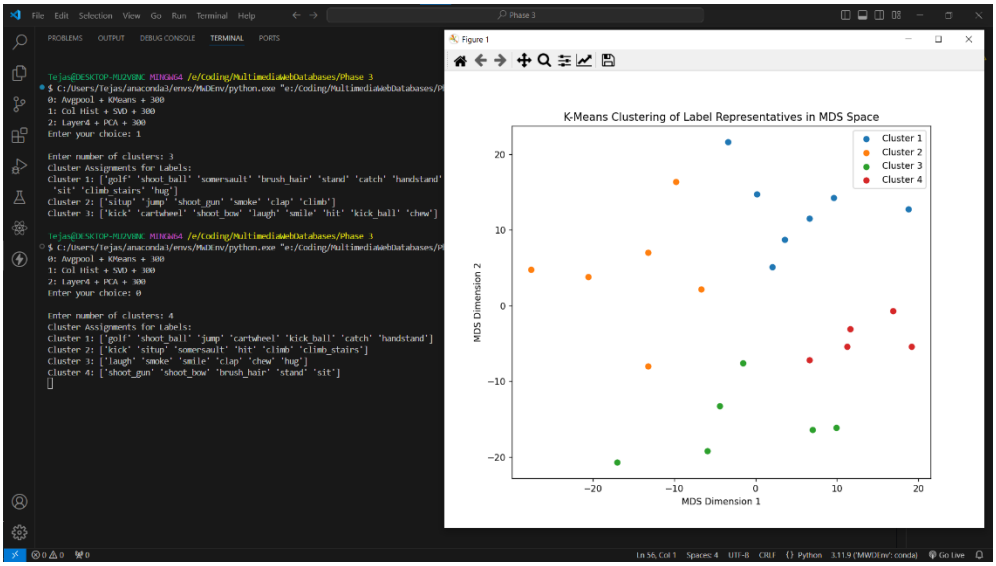
In Task 1a, we clustered videos based on their feature labels and visualized the results using MDS in 2D. While the 2D plot suggests poor clustering, examining the clusters through their thumbnails shows good results, with videos in each cluster being visually similar. This demonstrates that the clustering effectively grouped similar content, despite the limitations of the 2D visualization. The results heavily depend upon the threshold distance taken while creating adjacency matrix. We have taken it as 1% of total distances among all nodes to create a balanced graph.

Task1b

Input1:

Avgpool + KMeans + 300

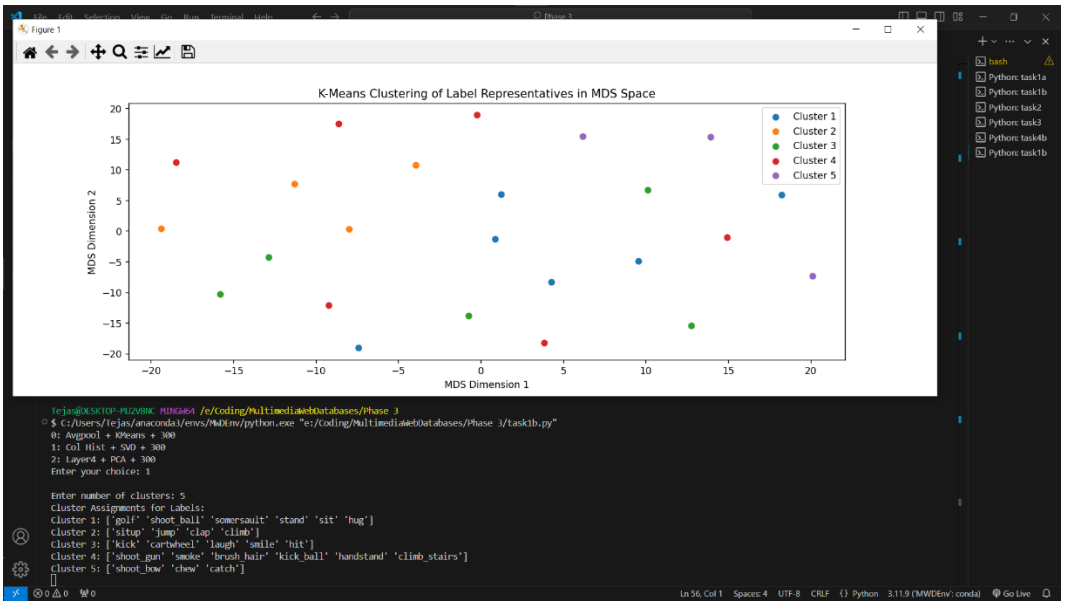
4 clusters



Input2:

Colhist + SVD+ 300

5 clusters

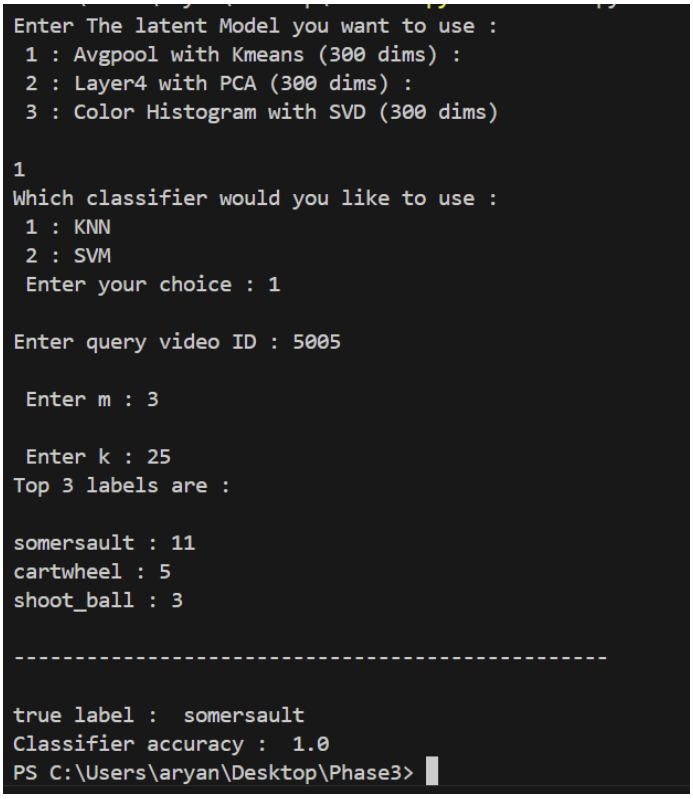


In Task 1b, we clustered videos based on their feature labels and visualized the results using MDS in 2D. The 2D plot shows decent clustering, with each cluster representing videos with similar actions. This indicates that the clustering effectively grouped videos

based on their content, making the results promising and aligned with the underlying patterns.

Task2

Input : Model – avgpool , classifier – KNN , video ID = 5005, m = 3, k = 25



Our query was of label somersault and as it can be seen, the top label retrieved is somersault with 11 votes. Cartwheel was ranked second which is also obvious since the action of cartwheel is very similar to somersault. Since query was of an odd target video, the classifier accuracy was also shown.

Input : Model – Layer4 with PCA , classifier – SVM , Video ID – 4879 , m – 3

```
PS C:\Users\aryan\Desktop\Phase3> python final2.py
Enter The latent Model you want to use :
 1 : Avgpool with Kmeans (300 dims) :
 2 : Layer4 with PCA (300 dims) :
 3 : Color Histogram with SVD (300 dims)

2
Which classifier would you like to use :
 1 : KNN
 2 : SVM
 Enter your choice : 2

Enter query video ID : 4879

 Enter m : 3
Model loaded from svm_layer4_pca(300).pkl
Top 3 labels :
laugh : 77.74283551666764
smile : 63.542175941421775
chew : 61.340487644868766

------------------------------------------------

true label :  laugh
Classifier accuracy :  1.0
```

This query (odd target video) was of label laugh and we obtained laugh as the first result. Hence our classifier accuracy became 1. The 3 returned labels all share a characteristic which is mouth movement. This might have been captured and caused the return of these labels.

Input : Model – Color Histogram with SVD, classifier – SVM , Video ID : 2000, m – 5
The query here is for non-target videos, but since we only have the labels of target videos, we got target labels as output. The action of the query video was of a boxer punching and hitting someone, which might have caused the return of label kick (kicking someone).

```
Enter The latent Model you want to use :
 1 : Avgpool with Kmeans (300 dims) :
 2 : Layer4 with PCA (300 dims) :
 3 : Color Histogram with SVD (300 dims)

3
Which classifier would you like to use :
 1 : KNN
 2 : SVM
 Enter your choice : 2

Enter query video ID : 2000

 Enter m : 5
Model loaded from svm_colhist_svd(300).pk
Top 5 labels :
kick : 66.10205846110458
sit : 62.332172948400526
clap : 60.77477087264495
smile : 60.442165580898966
hug : 60.28481682759176
```

Task3

Input 1 – Model= Layer4 + pca+300, L=5, h=10, VideoId =1313 and t= 5

```
tanishquezaware@Tanishques-MacBook-Air Sem 1 % /usr/bin/python3 "/Users/tanishquezaware/Documents/MS/Sem 1/CSE 515 Multimedia/Project/Phase 3/ta
sk3.py"
Select model number from the below options
 1. Average_pool + k_means + s=300
 2. Col_hist + svd + s=300
 3. Layer4 + pca + s=300
  3
Please enter # of layers :  5
Please enter # of hashes per layer :  10
Enter the VideoID :  1313
Enter the number of similar videos need to retrieve :  5
```

Results -

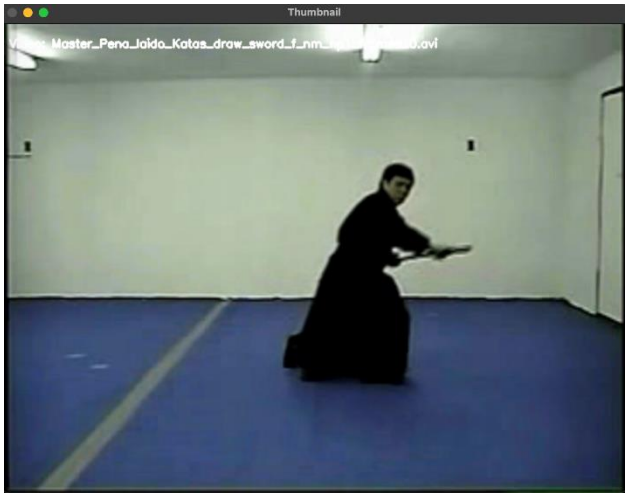
```
Number of unique candidates considered: 48
Number of overall candidates considered: 54
```

```
Top 5 similar videos for Video ID 1313:
Rank 1: Video ID 2936, Distance: 0.4973622371812447
Rank 2: Video ID 989, Distance: 0.6706715428608356
Rank 3: Video ID 868, Distance: 0.6971731758241209
Rank 4: Video ID 6733, Distance: 0.6977027264955992
Rank 5: Video ID 1594, Distance: 0.7041253947439577
```

Here the input given was for walk label along with the parameters for layers and number of hashes. Out of the 5 similar videos retrieved 2 are from walk and rest are having walking action or related action in them and the model would have found it similar to walk.

Input 2 - Model= Average pool +Kmeans + 300, L=5, h=10, VideoId =17and t= 6

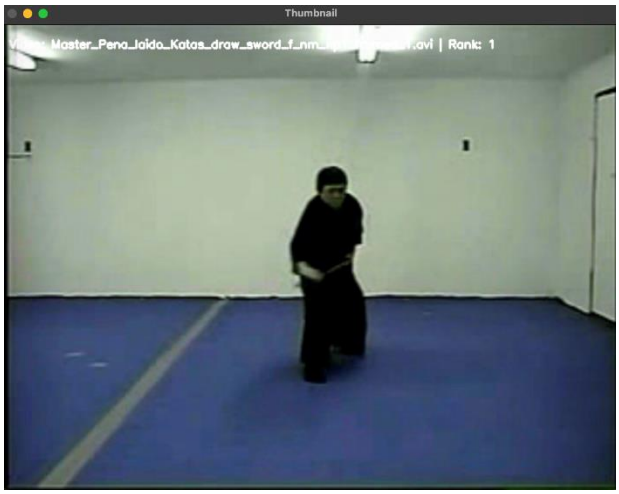
```
tanishquezaware@Tanishques-MacBook-Air Sem 1 % /usr/bin/python3 "/Users/tanishquezaware/Documents/MS/Sem 1/CSE 515 Multimedia/Project/Phase 3/task3.py"
Select model number from the below options
 1. Average_pool + k_means + s=300
 2. Col_hist + svd + s=300
 3. Layer4 + pca + s=300
  1
Please enter # of layers :  5
Please enter # of hashes per layer :  10
Enter the VideoID :  9
Enter the number of similar videos need to retrieve :  6
```

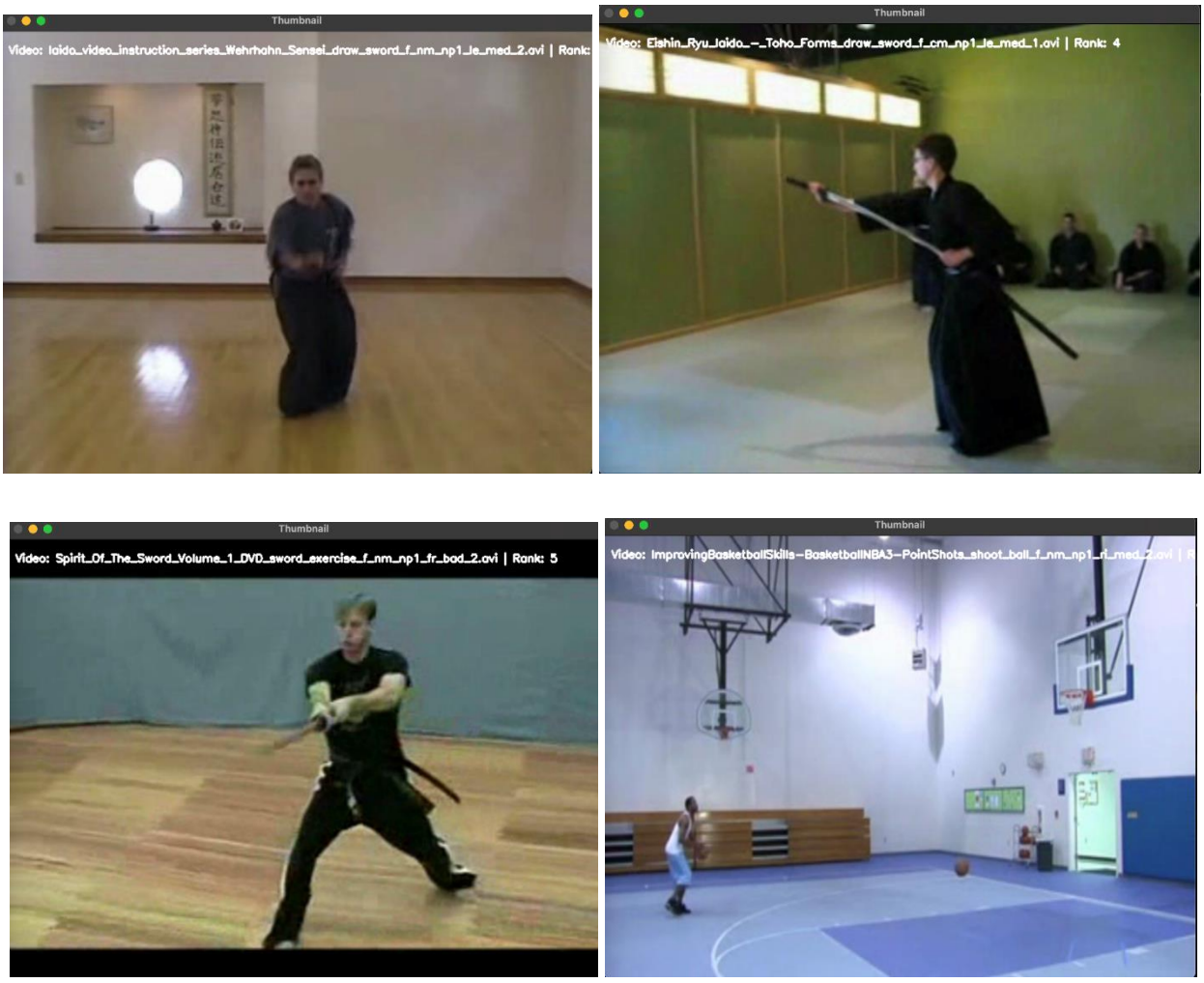


Results -

```
Number of unique candidates considered: 6764
Number of overall candidates considered: 22403
```

```
Top 6 similar videos for Video ID 9:
Rank 1: Video ID 78, Distance: 0.00044868238010942196
Rank 2: Video ID 69, Distance: 0.00048197076097744596
Rank 3: Video ID 80, Distance: 0.0008223532708208081
Rank 4: Video ID 25, Distance: 0.0010264624508452558
Rank 5: Video ID 2212, Distance: 0.0010780106170197312
Rank 6: Video ID 4157, Distance: 0.0011763742034365254
```





Here we have an input video from sword label and we have been given the required parameters. Out of the 6 similar videos we have 5 from sword label and one from other, the other would been due to model finding similar action of hands as similar to our query video.

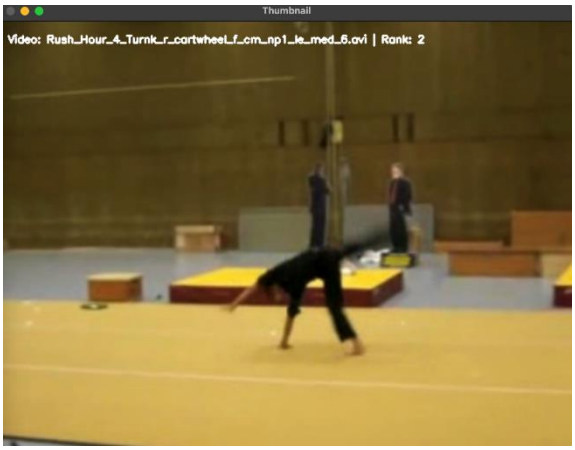
Input 3 – col_hist +svd + 300, L=6, h=13, VideoId =5000 and t=5

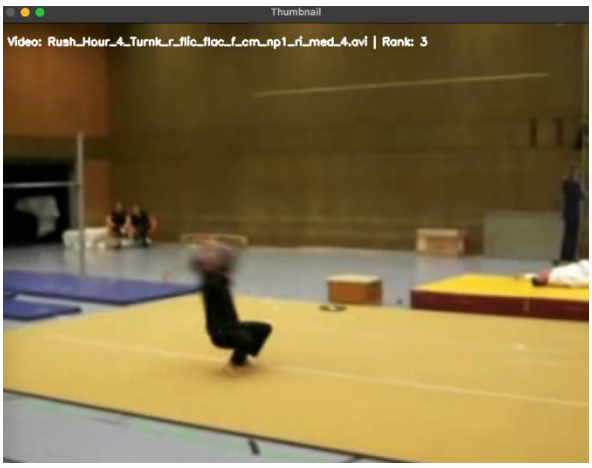
```
tanishquezaware@Tanishques-MacBook-Air Sem 1 % /usr/bin/python3 "/Users/tanishquezaware/Documents/MS/Sem 1/CSE 515 Multimedia/Project/Phase 3/ta
sk3.py"
Select model number from the below options
 1. Average_pool + k_means + s=300
 2. Col_hist + svd + s=300
 3. Layer4 + pca + s=300
  2
Please enter # of layers :  6
Please enter # of hashes per layer :  13
Enter the VideoID :  5000
Enter the number of similar videos need to retrieve :  5
```

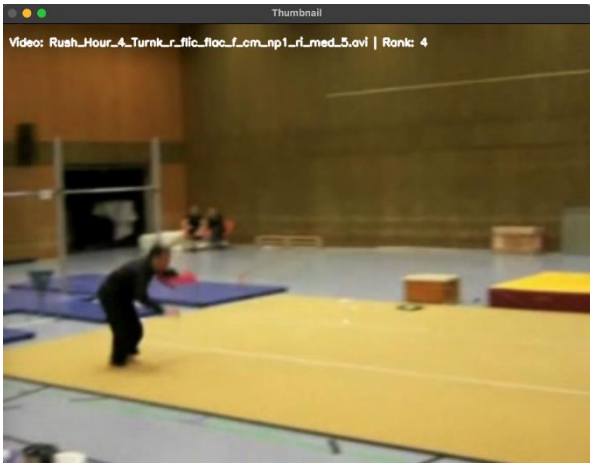
Result -

```
Number of unique candidates considered: 21
Number of overall candidates considered: 28
```

```
Top 5 similar videos for Video ID 5000:
Rank 1: Video ID 4689, Distance: 0.05921173911173949
Rank 2: Video ID 4624, Distance: 0.06845567586621693
Rank 3: Video ID 3243, Distance: 0.09762355706366743
Rank 4: Video ID 3297, Distance: 0.11201198215169417
Rank 5: Video ID 6353, Distance: 0.18998161954898207
```







In this example we have the input of somersault label and the with given parameters. From the results, the model has associated similar hand and leg actions of cartwheel, flip flop and handstand to somersault and thus found it similar.

Task4

(1)

Our query was of label somersault and using task3b we got 2 results of somersault while there was 1 irrelevant result, as it can be seen after the user feedback, all 3 videos came out to be somersault videos. We used knn here, so the query was moved towards the relevant (somersault) videos and the new result came out correct.

```
PS C:\Users\aryan\Desktop\Phase3> & C:/Users/aryan/anaconda3/pytho
Select model number from the below options
 1. Average_pool + k_means + s=300
 2. Col_hist + svd + s=300
 3. Layer4 + pca + s=300
3
Please enter # of layers :  5
Please enter # of hashes per layer :  5
Enter the VideoID :  5003
Enter the number of similar videos need to retrieve :  3
Number of unique candidates considered: 1159
Number of overall candidates considered: 1289
Top 3 similar videos for Video ID :
Rank 1: Video ID 5003, Distance: 0.0
Rank 2: Video ID 5096, Distance: 0.20881801985533432
Rank 3: Video ID 6303, Distance: 0.2454599543438558
Enter 1 for like and 0 for dislike against each video ID
rank : 1 , videoID : 5003 , feedback : 1
rank : 2 , videoID : 5096 , feedback : 1
rank : 3 , videoID : 6303 , feedback : 0
relevant :  [5003, 5096]
irrelevant :  [6303]

 1 : KNN Based
 2 : Decision Tree based1
Enter value of k for knn :
20
-------New Results---------
rank : 1 , videoID : 5096 , distance : 0.1447143489840358
rank : 2 , videoID : 5008 , distance : 0.32826476711535646
rank : 3 , videoID : 5028 , distance : 0.34763661239461163
PS C:\Users\aryan\Desktop\Phase3>
```

(2)

```
PS C:\Users\aryan\Desktop\Phase3> & C:/Users/aryan/anaconda3/python.exe c:/Users/aryan/Desktop/Phase3/final4.py
Select model number from the below options
 1. Average_pool + k_means + s=300
 2. Col_hist + svd + s=300
 3. Layer4 + pca + s=300
  3
Please enter # of layers :  5
Please enter # of hashes per layer :  5
Enter the VideoID :  4071
Enter the number of similar videos need to retrieve :  2
Number of unique candidates considered: 989
Number of overall candidates considered: 1082
Top 2 similar videos for Video ID :
Rank 1: Video ID 4071, Distance: 0.0
Rank 2: Video ID 2104, Distance: 0.40095707613646214
Enter 1 for like and 0 for dislike against each video ID
rank : 1 , videoID : 4071 , feedback : 1
rank : 2 , videoID : 2104 , feedback : 0
relevant :  [4071]
irrelevant :  [2104]

 1 : KNN Based
 2 : Decision Tree based1
Enter value of k for knn :
20
-------New Results---------
rank : 1 , videoID : 4070 , distance : 0.5265148553394012
rank : 2 , videoID : 4050 , distance : 0.5297143510668154
PS C:\Users\aryan\Desktop\Phase3>
```

This time our query was of type golf and we got 1 video of golf (which was the query itself) in our result. After the feedback, both of the video we got were of type golf.

(3)

```
PS C:\Users\aryan\Desktop\Phase3> & C:/Users/aryan/anacond
Select model number from the below options
 1. Average_pool + k_means + s=300
 2. Col_hist + svd + s=300
 3. Layer4 + pca + s=300
  2
Please enter # of layers :  4
Please enter # of hashes per layer :  5
Enter the VideoID :  10
Enter the number of similar videos need to retrieve :  2
Number of unique candidates considered: 2120
Number of overall candidates considered: 2285
Top 2 similar videos for Video ID :
Rank 1: Video ID 10, Distance: 0.0
Rank 2: Video ID 2219, Distance: 0.07794785978058638
Enter 1 for like and 0 for dislike against each video ID
rank : 1 , videoID : 10 , feedback : 1
rank : 2 , videoID : 2219 , feedback : 0
relevant :  [10]
irrelevant :  [2219]

 1 : KNN Based
 2 : Decision Tree based2
Top 2 videos after feedback:
Rank 1: Video ID 0, Relevance Score: 1
Rank 2: Video ID 1, Relevance Score: 1
PS C:\Users\aryan\Desktop\Phase3>
```

Our decision tree-based relevance feedback does not work as well, because of the way the tree is built. We chose feature mean as threshold and perhaps due to bias in the features the tree was more populated on one side than other which causes some random leafnodes to be retrieved.

Related Work

[1] This paper discusses on relevance feedback mechanisms as a way to refine retrieval results by incorporating user input iteratively. Also, it shows how user feedback can guide the system to focus on regions of interest within the feature space.

[2] It involves a method to learn compact binary codes for similarity search using spectral analysis. Also, it leverages the data distribution for improved accuracy compared to data-

independent methods like random projection-based LSH and thus is helpful in crucial compact representation of visual data.

Conclusion

This phase built upon the previous phase where we were predicting similar labels as well as videos and extended it to include various established methods like Spectral Clustering, Support Vector Machines, Locality Sensitive hashing and so on. We selected 3 unique latent models and found inherent dimensionality for each of the target video labels. We also took the feedback of the user in order to improve our retrieval results by identifying the videos that the user found as relevant and irrelevant.

Bibliography

1. Rui, Yong, et al. 'Image Retrieval: Current Techniques, Promising Directions, and Open Issues'. Journal of Visual Communication and Image Representation, vol. 10, no. 1, Mar. 1999, pp. 39–62. ScienceDirect, https://doi.org/10.1006/jvci.1999.0413.

2. Weiss, Yair, et al. 'Spectral Hashing'. Advances in Neural Information Processing Systems, vol. 21, Curran Associates, Inc., 2008. Neural Information Processing Systems, https://papers.nips.cc/paper_files/paper/2008/hash/d58072be2820e8682c0a27c0518e805e-Abstract.html

Appendix

Specific roles of the group members

Aryan Patel – Task2, Task4b

Tejas Parse – Task1, Readme

Om Patel – Task0

Tanishque Zaware – Task3, Task4a

List of Target Video Labels :

brush_hair, cartwheel, catch, chew, clap, climb, climb_stairs, golf, handstand, hit, hug, jump, kick, kick_ball, laugh, shoot_ball, shoot_bow, shoot_gun, sit, situp, smile, smoke, somersault, stand.

The rest are non-target labels