Title: Action Classification

Project Team Members:

1. Sri Rupin Potula
2. Vibhu Dixit
3. Abhishek Mulasi

Abstract:

The objectives for Phase 3 are designed to perform thorough clustering video data analysis. This step is essential and required in achieving the desired enhanced similarity-based retrieval capabilities. To clearly achieve this objective, an initial computation of inherent dimensionality for every distinct label in the target videos will be required. Following such a computation, spectral and k-means clustering processes must necessarily be conducted without the use of libraries that facilitate such operations with the sole aim of putting the videos in their corresponding labels. The tasks to be performed are visualizing the visualizations in two different formats: one is the 2D MDS space representation, and the other is a mosaic composed of groups of video thumbnails. Further, some classifiers like k-Nearest Neighbors (k-NN) and Support Vector Machines (SVM) will also be implemented without using any kind of external libraries for their implementation. These classifiers will be applied to the video data to predict the most probable labels that are assigned to each video. Performance measures are also taken care of with utmost care only for the odd-numbered target videos so that their accuracy can be measured. In the end, a Locality Sensitive Hashing (LSH) tool will be implemented for indexing high-dimensional vectors representing videos. This implementation will provide the means to search for similarities between those vectors in a much more efficient way. The selected visual model to be used for LSH should be able to process at least 256 dimensions.

Keywords: OpenCV, LSH, MDS, Similarity, K-Means, k-NN.

1. Assumptions:

In task 3, we assume that there are only 10 buckets. Since the hyperplane is chosen randomly using the GaussianRandomProjection, we get the hash key as the bucket number where the vector lies in the plane rather than any discrete quantity for understanding purposes. The width of the bucket is decided on the range of dot products. As part of Task-0a, the feature space Color Histogram wasn't included. As part of Task-3b, the latent model KMeans with HOG wasn't completely functional.

2. Introduction:

In the digital era, the number of videos that are being captured has increased rapidly because of the increase in the use of mobiles, laptops, and tablets. With that rapid increase, the volume of videos has been increasing exponentially. So, it is important to understand the content of the video structure, specifically the action that was being performed in the video. The project focuses on advanced techniques for video analysis. We analyze different approaches to vector modeling, similarity measurement, and feature extraction in video data. The primary goals consist of:

1. Using the R3D-18 architecture to implement feature descriptors based on neural networks.
2. Using HoG and HoF descriptors, develop a Spatio-Temporal Interest Point (STIP) analysis.
3. Converting video frames into color histograms.
4. Taking a set of target videos' feature descriptors and storing them.
5. Putting in place a system for comparing similar videos to recognize and display them.

We use libraries like PyTorch, torchvision, PyAV, and OpenCV for deep learning and visual information extraction. With potential applications in duplicate detection and video recommendation systems, the project aims to improve the field of content-based video access and evaluation.

3. Description of solutions:

The latent models selected as part of the tasks from 1 to 4 are “Layer3 + PCA”, “AvgPool + SVD” and “HOG + KMeans”. To apply the dimensionality reduction technique, the value ‘s’ which is the latent semantics is determined using the Task-0a of this project. For Task 1a, each label has its inherent dimensionality used, for all other Tasks, the inherent dimensionality for all the labels was identified and the maximum values were stored and used as a value for ‘s’. So, the latent semantics are 50, 76, and 87 for Layer3, Avgpool, and KMeans.

3.1 Task 0a:

The task-0a expects the user to provide a feature space, namely, Layer 3, Layer 4, Avgpool, HOG, and HOF. The main objective of this task is to find the inherent dimensionality of a target video label for the given feature space. The implementation is that for a given feature space, all the features for each unique label are gathered. For the features, a feature matrix is constructed; on this matrix, the eigenvalues are gathered by forming a covariance matrix. The eigenvalues are sorted, and their mean is taken. The eigenvalues whose values are greater than or equal to the mean are retained, and others are discarded. The number of eigenvalues that remain is the inherent dimensionality for that specific label.

3.2 Task 1a:

Task 1a finds the most significant clusters for each target label using spectral clustering. The program takes a latent space as mentioned above, and an integer c for the number of clusters to be found. It collects all the even target videos for a specific label and uses the dimensionality-reduced matrix for clustering. We use Connected Components (CCs) to represent the clusters instead of structures such as cliques. A CC in the feature space is computed by connecting all points within a certain threshold distance of each other. The initial default threshold varies for each latent space (1 for "Layer3 + PCA", 10 for "AvgPool + SVD" and 50 for "HOG + KMeans"). The graph formed after connecting the points, is stored as an adjacency matrix for further analysis.

We get the Laplacian matrix of the graph using the degree matrix and adjacency matrix. The Laplacian matrix represents the graph as an nxn matrix for n points where the diagonal of the matrix denotes the degree of the point, each connected edge is denoted by a -1, and every non-connected pair of vertices is denoted by a 0. We count the number of CCs in the graph using the number of zero eigenvalues from the Laplacian matrix, we can also count the number of CCs produced by a simple Depth First Graph Traversal. Based on the number of CCs and the number of clusters needed, we recompute the adjacency matrix by changing the distance threshold. If the number of CCs are more than the required clusters, we increase the threshold by a factor of 1.5x, effectively reducing the number of edges. If the number of CCs is half of what is required, we reduce the threshold by a factor of 0.8x. This change in threshold is repeated until the number of CCs is more than half of what is required by less than what is required.

Once we are in a desirable range, we begin spectral graph decomposition by breaking the worst cluster by Cluster Distance metric. We find the longest cluster, take its subgraph, and calculate its Laplacian matrix. We find its Fiedler vector, the eigenvector corresponding to its second-smallest eigenvalue (the smallest eigenvalue will be 0). All the nodes with a positive value are split into a new cluster and all those with a zero or negative value are placed in another cluster. We repeat this process until we get the desired number of clusters.

3.3 Task 1b:

The task performs a K-Means clustering algorithm on latent model spaces emanating from three different latent models and showcases the results of such clustering either by MDS or by showing video labels for each cluster. This program asks a user to select one out of three latent models, which are: PCA with Layer 3 features, SVD with AvgPool features, and K-Means with HOG features. Depending on the model a user chooses, the correspondingly extracted feature vectors become inputs for K-Means. The K-Means algorithm starts by randomly initializing the centroid and iteratively assigns every data point to the nearest centroid, after which the centroid is updated until the algorithm converges; that is, the position of centroids changes very little. A tolerance value of 0.1 was used when the distance between the new and old

centroids distance is less than 0.1 the algorithm terminates. The distance between the new centroids and the old centroids was calculated using L2 norm. Also, the tolerance value was varied and checked for different values for 0.1 the better visual representations were observed. Using L2 norm, the distance from each point to the cluster centroids is calculated to assign points to the centroid. L2 norm was used as the results are more appropriate. Finally, after clustering, one can see cluster results.

The first option, MDS, reduces the feature space to 2D for a clearer, visual representation of the clusters. The second option visualizes the grouping of labels within the clusters; interactive annotations are enabled by the `mplcursors` library, showing additional information such as the cluster's label assignments. The code then proceeds to organize the video labels into their respective clusters, utilizing the `Counter` class to enumerate the most frequent labels in each group. For each cluster, the top 8 most frequent labels are assigned. Since there would be a redundant number of video labels. This provides further insight into the structure and distribution of the data, allowing for a deeper understanding and analysis of the latent model outputs.

3.4 Task 2:

The user selects the latent model and enters the query videoID. For the given videoID, the features were gathered. For all the labels, the features were extracted and stored as a feature matrix. Depending upon the latent space the given feature matrix was converted to the specific latent dimension. From this latent space, the query feature was removed with the help of the index. The remaining data acts as an input. For the class of the latent model, a list was prepared that contains the class label corresponding to that specific index.

To implement KNN, the function operates on the user-specified value of ‘K’. Depending on the choice, the latent model was formed. From the formed latent model, the query feature was removed depending on the index. Using the L2 norm, the distance between all entries in the latent model space to that of the query feature were formed. Then the topmost ‘k’ feature indices are selected that are closest to the query feature. The index of the top ‘k’ entries from the feature from the latent space is gathered. The classes of this identified. This was identified as a list made where action labels to a range of indices of the action in the latent model.

The following approach was used to employ the SVM classifier. Firstly, the latent model (feature set) and the class label list were prepared. For each unique class, the model treated the current unique class as positive (+ 1) and all other classes as negative (− 1). A binary SVM was then trained for each class. Initially, the weight vector and the bias term were initialized to zero. For each entry in the latent model, the feature vector and its corresponding class label were used to calculate the margin. If the margin was greater than or equal to 1, the classification was correct, and only the regularization term was updated. Otherwise, a misclassification occurred,

and both the regularization term and the hinge loss gradient were updated. The weight and bias were then adjusted using these gradients scaled by the learning rate. This process was repeated over multiple epochs to minimize the SVM loss function and achieve an optimal decision boundary for each class. The learning rate, epochs, and lambda parameters are predefined as 0.001, 1000, and 0.01 which are all random. The loss function that was used is hinge loss with L2 norm.

The main choice of this loss function is this includes both regularisation term and hinge loss. Regularisation prevents overfitting whereas the hinge loss penalises the model when the prediction margin is less than 1. Then, for the given query feature, the scores are calculated with weight and bias that are stored for each class. The scores that are computed are sorted and top m classes are taken and printed. The scored list is sorted from high to low. The highest scores are picked as they showcase the confidence of the sample to be in a specific class. If the true label of the class is a target label then the per-classifier score was calculated by gathering the rank of that label from the predicted label.

3.5 Task 3a:

The Python code implements a Locality Sensitive Hashing technique by generating random hyperplanes through Gaussian Random Projection. It projects the high-dimensional vectors onto the random hyperplanes and computes the dot product between the vector and the hyperplanes to hash these high-dimensional vectors into buckets. To get a hash key for each vector, a dynamic threshold `w` is applied on top; the threshold is based on the range of dot products across each layer of hyperplanes. This threshold controls the granularity of the hashing and therefore aids in clustering similar vectors together while reducing the chance of collisions for dissimilar vectors.

A unique aspect of the code is the multi-layer hashing approach it uses, where each layer takes a different set of random hyperplanes. This increases the chances that similar vectors will be correctly grouped. Also, the dynamic choice of threshold ‘w’ concerning the range of dot products and random biases make the hash function adaptive and robust. This way, the algorithm efficiently enables the organization of high-dimensional data for applications such as approximate nearest neighbor search, clustering, and similarity retrieval without expensive distance calculations.

3.6 Task 3b:

The code provides a realization of the Video Search Tool that uses LSH in conjunction with different methods of dimensionality reduction in order to efficiently retrieve videos similar to a query video. It retrieves video feature data from an SQLite database and, depending on user input, applies one of three possible dimensionality reduction techniques: PCA, SVD, or KMeans clustering. Each method reduces the high-dimensional feature space of the videos to a

low-dimensional representation; it enables the tool to handle and compare video features effectively in that reduced space with their critical characteristics preserved.

The central role of the tool lies in the efficiency of LSH in search. To start with, for every video, its reduced feature vectors are hashed into buckets according to their projections to randomly generated hyperplanes. The hash processes allow similar videos to likely appear in the same bucket for a given number of possible comparisons. For a given query video, it would compute its hash keys under each LSH layer, and then fetch candidate videos located in the same buckets as the query hash key. Results in such candidates, which then rank the query video in this reduced feature space by its Euclidean distance.

It then highlights the top ‘t’ most similar videos, including their similarity scores. Also, it visually represents those in a grid showing their thumbnails and their respective similarity scores. This allows a more user-friendly approach for assessing search results, using a combination of efficient back-end computation with intuitive front-end visualization. The tool is designed for scalability, efficiently allowing searches in big video databases.

4. Interface Specifications:

To keep the interface simple, a terminal approach is chosen where the code is executed in the terminal and the user provides input like a video name. Depending on the task the similar video scores are printed in the terminal.

5. System Requirements:

5.1 System Requirements:

1. CPU: 8 Core CPU
2. RAM: 8GB
3. GPU: Not Essentially used.

5.2 SOL Requirements:

1. Nodes: 2 (Jupyter Notebook)

5.3 Programming Language:

1. Python

5.4 Modules Required:

1. OS
2. JSON

5.5 Libraries:

1. Pytorch
2. OpenCV
3. Sci-learn
4. Numpy
5. Matplotlib

6. Av

6. Execution Instructions:

As a prerequisite, the dataset being a large file size, as part of the submission process, the directory structure along with the code would be supplemented. The users are requested to download the dataset from [1] and place it in a directory named 'hmdb51_org'. During the execution of the tasks, the database and output directories would be populated.

The 'Code' directory consists of all the code for this respective phase which are all Python files. To execute them, the command used is 'python <filename>'. Execution description for each of the tasks is described below.

Task 0a:

Ensure to install all dependencies that might be used in the code in PCA Inherent Dimensionality Calculation: database manipulation -sqlite3, JSON read json, numpy - numerical calculations. Put files Phase_3.db, total_target_features.json, videoID.json and category_map.json in./database/ folder. This script will ask you to choose a feature space upon running. You can choose among the following: 1 for `Layer_3`, 2 for `Layer_4`, 3 for `AvgPool`, 4 for `BOF_HOG`, or 5 for `BOF_HOF`. Whichever you choose, the script will calculate and print out the inherent dimensionality for each action in the `target_videos` list, which includes actions such as 'golf', 'shoot_ball', 'brush_hair', etc. Each of these actions will print its result by indicating the computed intrinsic dimensionality. Make sure the database and JSON files are well formatted to avoid issues during execution.

Task 1a:

As part of this task, the users are prompted to select the latent models: 1 for PCA with Layer_3, 2 for SVD with Avgpool, or 3 for K-Means with HOG, and they have to select the number of clusters they want to group the data into. Then the users are prompted for visualization technique: 1 for MDS or 2 for clusters represented as thumbnails of videos. If MDS is selected, it visualizes the clusters in 2D. If the visualizations are as a group of video thumbnails, all the thumbnails are displayed with the respective video names.

Task 1b:

As part of this task, the users are prompted to select the latent models: 1 for PCA with Layer_3, 2 for SVD with Avgpool, or 3 for K-Means with HOG, and they have to select the number of clusters they want to group the data into. Then the users are prompted for visualization technique: 1 for MDS or 2 for cluster grouping. If MDS is selected, it visualizes the clusters in 2D. If the visualizations are as a group of clusters, the users can hover over the points to see the labels specific to that cluster.

Task 2:

As part of this task, the users are prompted to select the latent models: 1 for PCA with Layer_3, 2 for SVD with Avgpool, or 3 for K-Means with HOG. The users need to provide the videoID. Also, the value of ‘m’ and ‘k’ are gathered. The value of ‘k’ is gathered if the classification technique is ‘k-NN’. Based on the classification technique, the top ‘m’ labels are predicted and printed. If the given videoID is odd and the label of it is in target videos then the per-classifier score would be displayed.

Task 3a:

The program asks for several inputs: Number of Layers, Hash number per Layer, Vectors Number, and Vectors' dimensionality. Then call create_hyperplanes_from_data() which, given vectors generates random hyperplanes by utilizing class GaussianRandomProjection of each layer by computing their dot product in relation with vectors; afterwards uses output from a dot-product to calculate the hash key. These hash keys are stored in an LSH index such that vectors under similar projections fall into one group. It prints out each layer's final hash index after hashing is done such that every hash key of the index would map back to the original vector's index.

Task 3b:

To first get the thumbnails for corresponding video IDs, run ‘Task-3b_preprocess.py’ to get the folder containing the jpg thumbnails saved as “{videoID}.jpg” format. First, it asks for three inputs: the path for the database, feature column-for example, AvgPool or Layer_3, and query video ID. It instantiates a VideoSearchTool class with those three inputs and parameters including several layers: L; the number of hashes: k; and the value of ‘w’ in the hashing function. It is going to load video features from a database and hash them into an LSH index. Given a query video ID, the program will compute the hash of the query video and pull similar videos from the LSH index. It will then compute the Euclidean distance for candidates. Then, it will select the top ‘t’ most similar videos. It displays thumbnails of most similar videos in grid format.

7. Related Work:

In (Yoon & Han, 2022), the authors of this paper worked on ‘Content Based Video Retrieval’, to extract the features models the authors utilized 3D-CNN as their pre-trained neural network, the main choice of this approach as this would benefit the developers when working on limited resources. But even, through this approach when dimensionality reduction happens there would be a feature loss. So, the authors used network representation (Snell, Swersky, & Zemel, 2017) to eliminate feature loss.

8. Results/Observations:

8.1 Task-0a:

```
srirupin@MacBookPro Code % python3 Task_0a.py
Select a Feature Space from the following: 1 - Layer3, 2 - Layer4, 3 - AvgPool, 4- HOG, 5 - HOF : 1
The Inherent Dimensionality for golf element is:  50
The Inherent Dimensionality for shoot_ball element is:  30
The Inherent Dimensionality for brush_hair element is:  30
The Inherent Dimensionality for handstand element is:  45
The Inherent Dimensionality for shoot_bow element is:  25
The Inherent Dimensionality for cartwheel element is:  37
The Inherent Dimensionality for hit element is:  46
The Inherent Dimensionality for shoot_gun element is:  32
The Inherent Dimensionality for hug element is:  40
The Inherent Dimensionality for sit element is:  42
The Inherent Dimensionality for catch element is:  43
The Inherent Dimensionality for jump element is:  45
The Inherent Dimensionality for situp element is:  36
The Inherent Dimensionality for chew element is:  37
The Inherent Dimensionality for kick element is:  45
The Inherent Dimensionality for smile element is:  47
The Inherent Dimensionality for clap element is:  33
The Inherent Dimensionality for kick_ball element is:  44
The Inherent Dimensionality for smoke element is:  34
The Inherent Dimensionality for climb element is:  33
The Inherent Dimensionality for somersault element is:  40
The Inherent Dimensionality for climb_stairs element is:  38
The Inherent Dimensionality for laugh element is:  30
The Inherent Dimensionality for stand element is:  39
srirupin@MacBookPro Code %
```

Fig 1: Displays the inherent dimensionality for feature space Layer3.

The users are prompted to enter the feature space of their choice. The user provided Layer3 as feature space. Depending upon the feature space, the inherent dimensionality for all the target video labels is calculated. Among all of the target video labels, ‘golf” represented the highest value which is 50 which showcases that to capture the action at least 50 dimensions are required.

```
srirupin@MacBookPro Code % python3 Task_0a.py
Select a Feature Space from the following: 1 - Layer3, 2 - Layer4, 3 - AvgPool, 4- HOG, 5 - HOF : 3
The Inherent Dimensionality for golf element is:  70
The Inherent Dimensionality for shoot_ball element is:  56
The Inherent Dimensionality for brush_hair element is:  63
The Inherent Dimensionality for handstand element is:  69
The Inherent Dimensionality for shoot_bow element is:  53
The Inherent Dimensionality for cartwheel element is:  65
The Inherent Dimensionality for hit element is:  66
The Inherent Dimensionality for shoot_gun element is:  61
The Inherent Dimensionality for hug element is:  73
The Inherent Dimensionality for sit element is:  76
The Inherent Dimensionality for catch element is:  55
The Inherent Dimensionality for jump element is:  71
The Inherent Dimensionality for situp element is:  43
The Inherent Dimensionality for chew element is:  58
The Inherent Dimensionality for kick element is:  70
The Inherent Dimensionality for smile element is:  69
The Inherent Dimensionality for clap element is:  61
The Inherent Dimensionality for kick_ball element is:  75
The Inherent Dimensionality for smoke element is:  61
The Inherent Dimensionality for climb element is:  66
The Inherent Dimensionality for somersault element is:  75
The Inherent Dimensionality for climb_stairs element is:  60
The Inherent Dimensionality for laugh element is:  63
The Inherent Dimensionality for stand element is:  75
srirupin@MacBookPro Code %
```

Fig 2: Displays the inherent dimensionality for feature space Avgpool.

The users are prompted to enter the feature space of their choice. The user provided Avgpool as a feature space. Depending upon the feature space, the inherent dimensionality for all the target video labels is calculated. Among all of the target video labels, ‘sit’ represented the highest value which is 76 which showcases that inorder to capture the action atleast 76 dimensions are required. The minimum was using 43 dimensions for action ‘situp’.

```
srirupin@MacBookPro Code % python3 Task_0a.py
Select a Feature Space from the following: 1 - Layer3, 2 - Layer4, 3 - AvgPool, 4- HOG, 5 - HOF : 5
The Inherent Dimensionality for golf element is:  50
The Inherent Dimensionality for shoot_ball element is:  49
The Inherent Dimensionality for brush_hair element is:  72
The Inherent Dimensionality for handstand element is:  56
The Inherent Dimensionality for shoot_bow element is:  44
The Inherent Dimensionality for cartwheel element is:  50
The Inherent Dimensionality for hit element is:  59
The Inherent Dimensionality for shoot_gun element is:  54
The Inherent Dimensionality for hug element is:  58
The Inherent Dimensionality for sit element is:  52
The Inherent Dimensionality for catch element is:  39
The Inherent Dimensionality for jump element is:  54
The Inherent Dimensionality for situp element is:  53
The Inherent Dimensionality for chew element is:  52
The Inherent Dimensionality for kick element is:  50
The Inherent Dimensionality for smile element is:  42
The Inherent Dimensionality for clap element is:  62
The Inherent Dimensionality for kick_ball element is:  53
The Inherent Dimensionality for smoke element is:  54
The Inherent Dimensionality for climb element is:  54
The Inherent Dimensionality for somersault element is:  49
The Inherent Dimensionality for climb_stairs element is:  51
The Inherent Dimensionality for laugh element is:  43
The Inherent Dimensionality for stand element is:  52
srirupin@MacBookPro Code %
```

Fig 3: Displays the inherent dimensionality for feature space Avgpool.

The users are prompted to enter the feature space of their choice. The user provided HOF as a feature space. Depending upon the feature space, the inherent dimensionality for all the target video labels is calculated. Among all of the target video labels, ‘brush_hair’ represented the highest value which is 72 which showcases that inorder to capture the action at least 72 dimensions are required. The minimum was using 39 dimensions for action ‘catch’.

8.2 Task-1a:

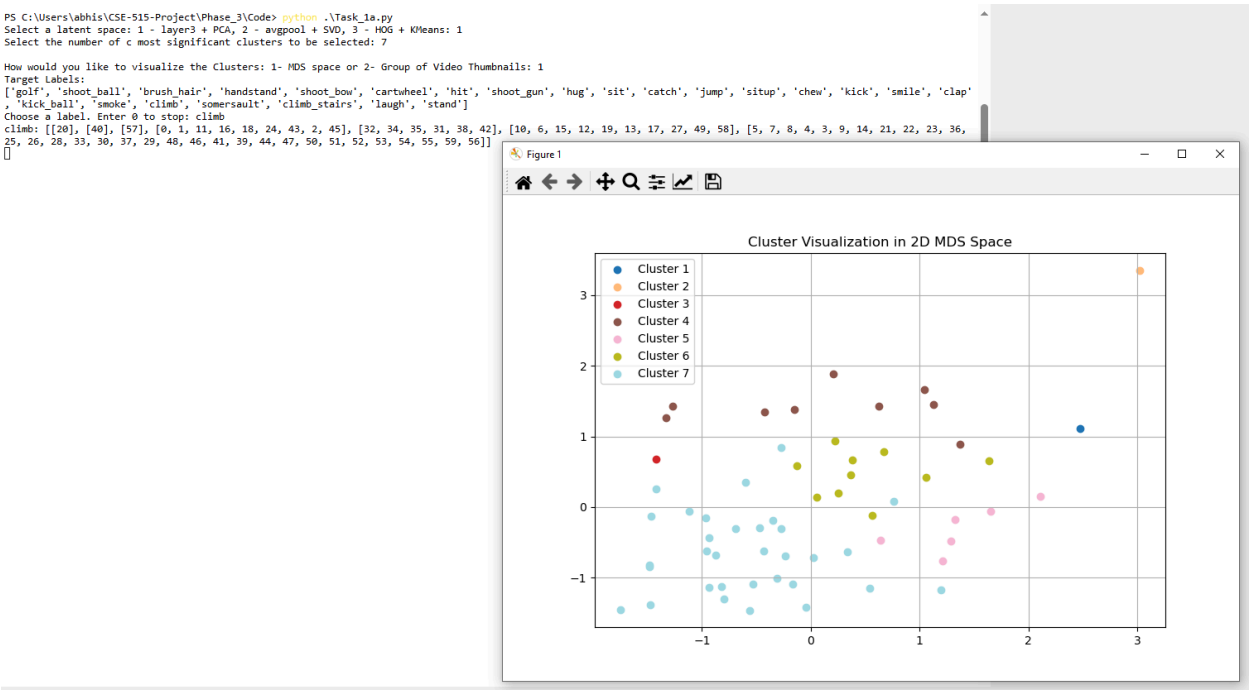

Fig 4: For the latent model as PCA+Layer3, Spectral clustering is visualized using MDS.

The user chose the latent model as PCA+Layer3, the number of clusters is 7 and the visualization method used is MDS. Each cluster is represented with a different color for differentiation. There is a main cluster that is much bigger than all the others. However, there are a few clusters that are spread out and have less similarity and the elements of those clusters vary.

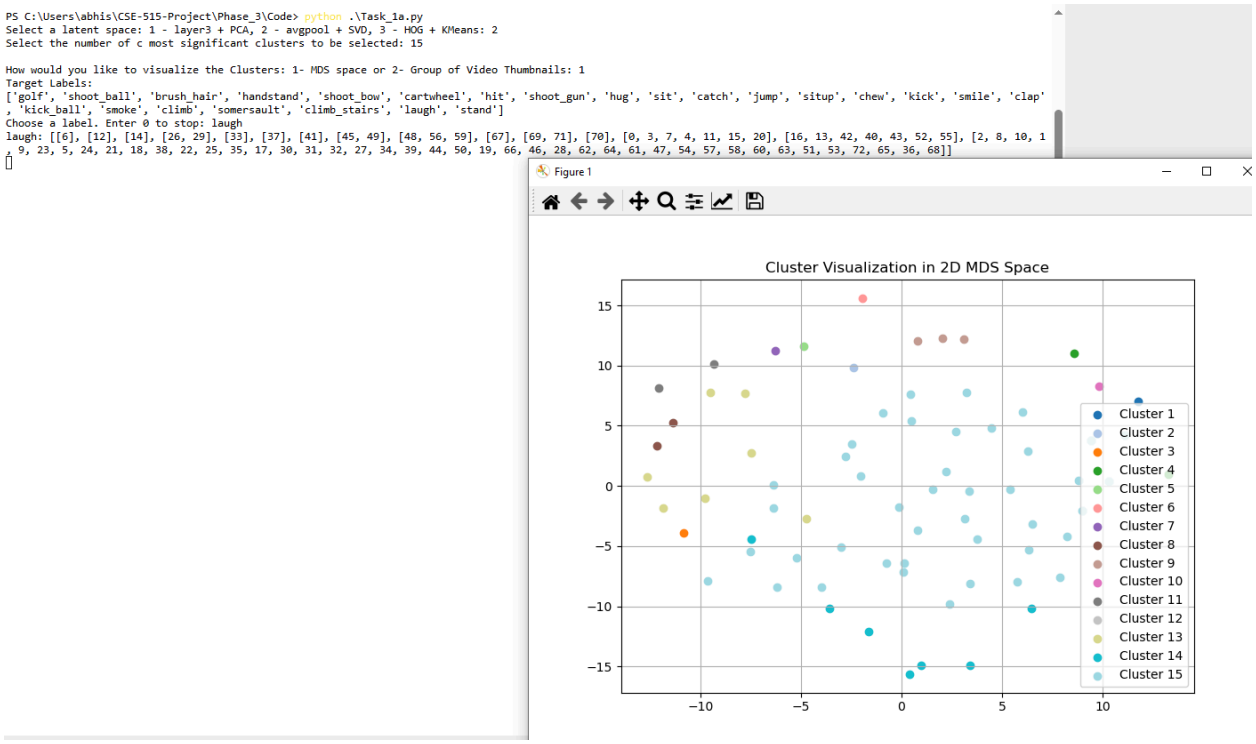

Fig 5: For the latent model as SVD+AvgPool, Spectral clustering is visualized using MDS.

The user chose the latent model as SVD+Avgpool, the number of clusters is 15 and the visualization method used is MDS. Each cluster is represented with a different cluster color for differentiation. As the number of clusters increases, the objects that represent these clusters are loosely packed which says that the groupings of clusters are similar.

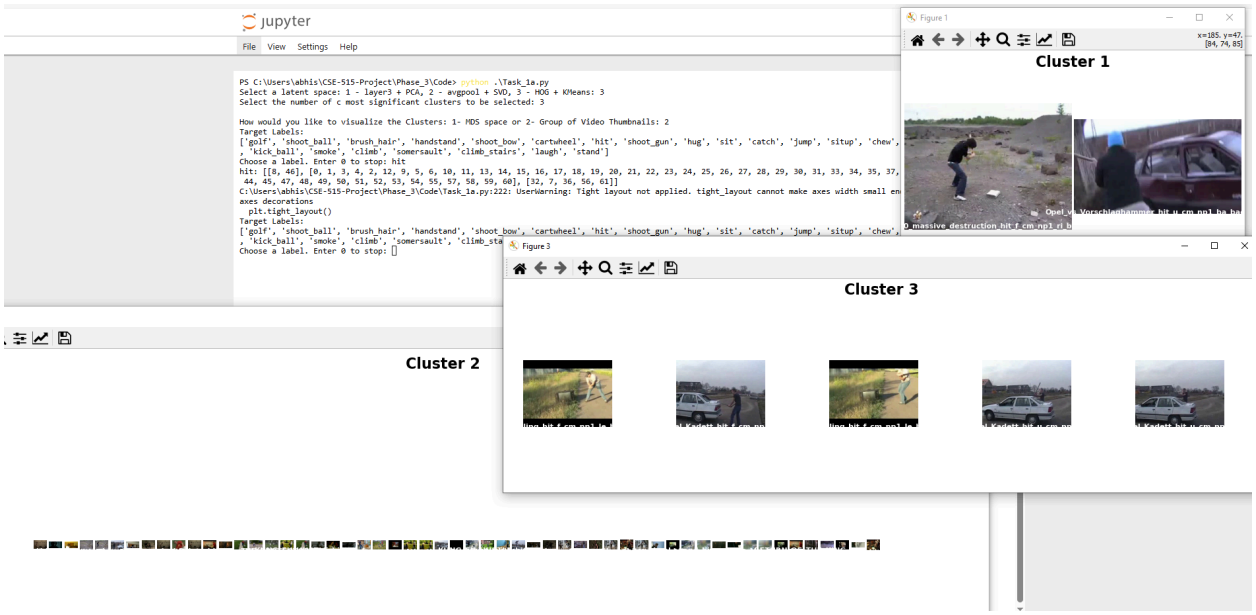

Fig 6: For the latent model as KMeans+HOG, Spectral clustering is visualized as video Thumbnails.

The user chose the latent model as KMeans+HOG, the number of clusters is 3 and the visualization method used is Video Thumbnails. Each cluster is represented by the thumbnails of the videos it consists of. As the number of clusters decreases, the disparity between the cluster distances increases, as outliers get assigned their own clusters.

8.3 Task-1b:

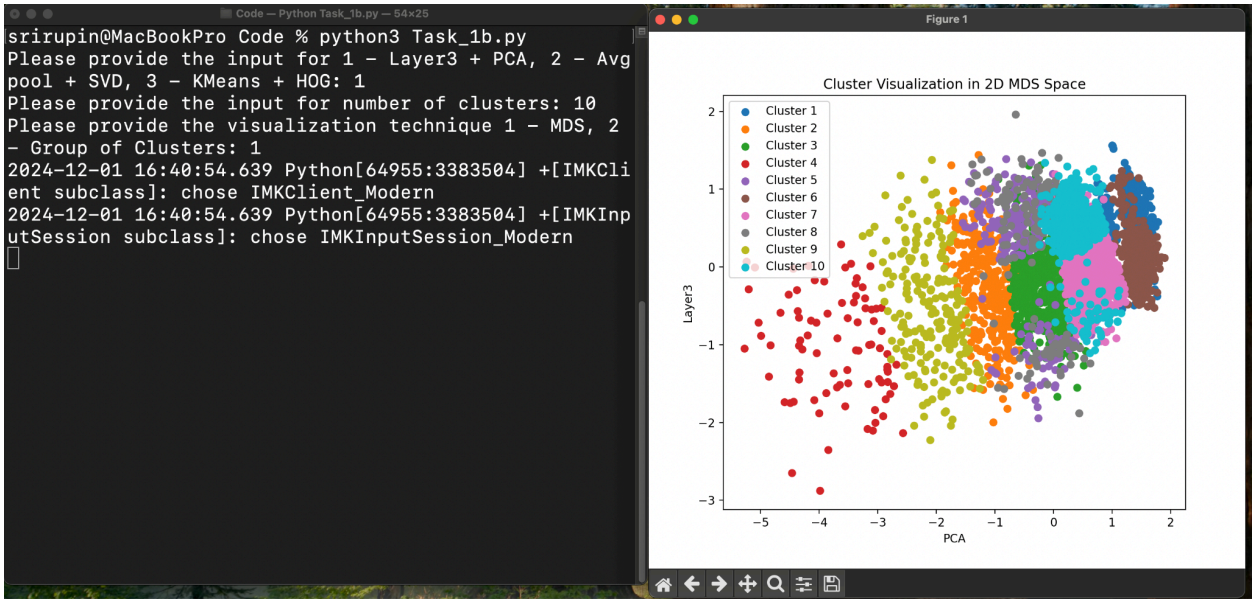

Fig 7: For latent model as PCA+Layer3, KMeans clustering is visualized using MDS.

The user chose the latent model as PCA+Layer3, the number of clusters is 10 and the visualization method used is MDS. Each cluster is represented with a different cluster color for differentiation. Some of the clusters are tightly packed which indicates the similarity among the elements of a cluster. However, there are a few clusters that are spread out and has less similarity and the elements of those clusters vary.

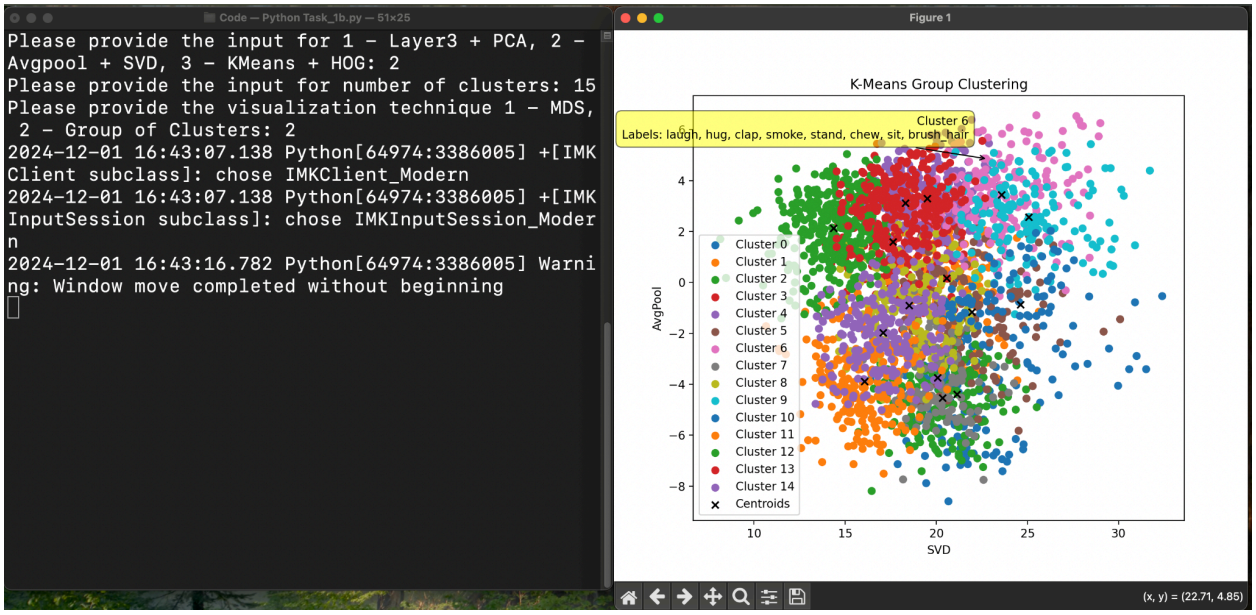

Fig 8: For latent model as SVD+AvgPool, KMeans clustering is visualized as group of labels.

The user chose the latent model as SVD+Avgpool, the number of clusters is 20 and the visualization method used is MDS. Each cluster is represented with a different cluster color for differentiation. When the user hovers over each data point, the labels corresponding to a specific cluster are represented. As the number of clusters increases, the objects that represent these clusters are loosely packed which says that the groupings of clusters are similar.

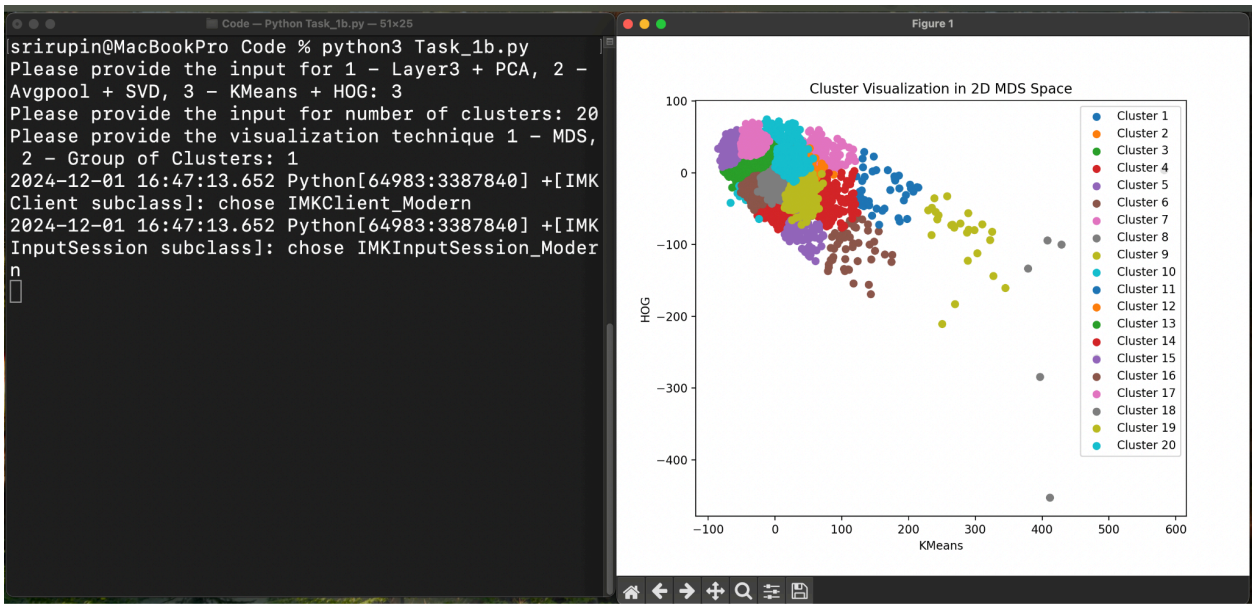

Fig 9: For the latent model as KMeans+HOG, KMeans clustering is visualized using MDS.

The user chose the latent model as SVD+Avgpool, the number of clusters is 20 and the visualization method used is MDS. Each cluster is represented with a different cluster color for

differentiation. When the user hovers over each data point, the labels corresponding to a specific cluster are represented. Most of the objects that correspond to a similar cluster are tightly packed but towards the right, the objects are loosely packed. This means that the objects of those clusters are dissimilar.

8.4 Task-2:

```
srirupin@MacBookPro Code % python3 Task_2.py
Provide the  VideoID: 1001
Provide the  classifier 1 - KNN, 2 - SVM: 1
Please provide the input for 1 - Layer3 + PCA, 2 - Avgpool + SVD, 3 - KMeans + HOG: 1
Provide the value for m : 10
Provide the value for k : 10
The predicted m labels are:  run
The predicted m labels are:  shoot_gun
The predicted m labels are:  run
The predicted m labels are:  throw
The predicted m labels are:  punch
The predicted m labels are:  sword_exercise
The predicted m labels are:  fall_floor
The predicted m labels are:  sword
The predicted m labels are:  run
The predicted m labels are:  turn
Per-classifier accuracy value is:  1.1
srirupin@MacBookPro Code %
```

Fig 10: For the latent model as Layer3+PCA, using KNN as a classifier 10 most significant labels are printed.

The user provided the input videoID as 1001, chose the classifier as KNN, and the latent model as Layer3+PCA. The value of m is 10 and k is 10. The KNN classifier found the top 10 similar objects that are related to the query and their respective features are printed. It could be observed that the 'run' is the most significant and repeated item. The per-classifier score is 1.1.

```
Provide the  VideoID: 1015
Provide the  classifier 1 - KNN, 2 - SVM: 2
Please provide the input for 1 - Layer3 + PCA, 2 - Avgpool + SVD, 3 - KMeans + HOG: 2
Provide the value for m : 10
The predicted m labels are:  wave
The predicted m labels are:  throw
The predicted m labels are:  dive
The predicted m labels are:  dribble
The predicted m labels are:  draw_sword
The predicted m labels are:  sword
The predicted m labels are:  run
The predicted m labels are:  swing_baseball
The predicted m labels are:  ride_horse
The predicted m labels are:  hit
The per-classifier accuracy value is zero
srirupin@MacBookPro Code %
```

Fig 11: For the latent model as SVD+AvgPool, using SVM as a classifier 10 most significant labels are printed.

The user provided the input videoID as 1015, chose the classifier as SVM, and the latent model as AvgPool+SVD. The value of m is 10. The SVM classifier found the top 10 similar

objects that are related to the query and their respective features are printed. The per-classifier score is 0.

```
srirupin@MacBookPro Code % python3 Task_2.py
Provide the  VideoID: 577
Provide the  classifier 1 - KNN, 2 - SVM: 1
Please provide the input for 1 - Layer3 + PCA, 2 - Avgpool + SVD, 3 - KMeans + HOG: 3
Provide the value for m : 20
Provide the value for k : 30
The predicted m labels are:  clap
The predicted m labels are:  pullup
The predicted m labels are:  pullup
The predicted m labels are:  pushup
The predicted m labels are:  situp
The predicted m labels are:  shoot_bow
The predicted m labels are:  punch
The predicted m labels are:  pullup
The predicted m labels are:  smoke
The predicted m labels are:  pour
The predicted m labels are:  situp
The predicted m labels are:  pullup
The predicted m labels are:  swing_baseball
The predicted m labels are:  smoke
The predicted m labels are:  swing_baseball
The predicted m labels are:  swing_baseball
The predicted m labels are:  swing_baseball
The predicted m labels are:  sword
The predicted m labels are:  pick
The predicted m labels are:  ride_horse
Per-classifier accuracy value is:  0.9
srirupin@MacBookPro Code %
```

Fig 12: For the latent model as KMeans+HOG, using KNN as a classifier 20 most significant labels are printed.

The user provided the input videoID as 577, chose the classifier as SVM, and the latent model as KMeans+HOG. The value of m is 20 and k is 30. The KNN classifier found the top 20 similar objects that are related to the query and their respective features are printed. It could be observed that the ‘pullup’ is the most significant and repeated item. The per-classifier score is 0.9.

8.5 Task 3a:

```
PS C:\Users\vibhu\OneDrive\Desktop\CSE-515-Project> & C:/Users/vibhu/AppData/Local/Programs/Python/Python312/python.exe c:/Users/vibhu/OneDrive/De
top/CSE-515-Project/Phase_3/Code/Task-3a.py
Enter the number of layers: 2
Enter the number of hashes per layer: 4
Enter the number of vectors: 4
Enter the dimensionality of each vector: 4
Enter each vector as space-separated values:
Vector 1: 1 5 2 3
Vector 2: 5 2 7 3
Vector 3: 1 7 2 9
Vector 4: 9 2 5 0
Layer 1 Hyperplanes (Random Projections):
Hyperplane 1: [ 0.02344246  0.034189    0.39550693 -0.22168445]
Hyperplane 2: [ 0.38408353  0.52527607 -0.10627555 -0.67317732]
Hyperplane 3: [-0.0302903  -0.02690155 -0.29564962 -0.17371228]
Hyperplane 4: [-0.19268309 -0.0545498   0.41269433 -0.37563098]
Layer 1 Dot Product Range: Min=-3.129821872526476, Max=3.9759261582118923
Layer 1 Selected w: 0.7105748030738368
Layer 1 Random Biases: [ 2.24800139  1.18117885  1.61629733 -0.60623194]
Layer 2 Hyperplanes (Random Projections):
Hyperplane 1: [-0.00545058 -0.08820834 -0.38277648 -0.39159458]
Hyperplane 2: [-0.7588564   0.41249331  0.58527722 -0.40192023]
Hyperplane 3: [-0.05769562 -0.13417695  0.40236155  0.55237907]
Hyperplane 4: [-0.19879699 -0.1928392   0.74389792 -0.48243382]
Layer 2 Dot Product Range: Min=-4.912813153361991, Max=4.779200444740449
Layer 2 Selected w: 0.9692013598102441
Layer 2 Random Biases: [ 3.34045294  1.77828285 -0.70410983  3.61840256]

Layer 1 hash index:
Hash Key: (3, 2, 0, 0) -> Vectors: [0]
Hash Key: (6, 1, 0, 0) -> Vectors: [1]
Hash Key: (1, 0, 0, 0) -> Vectors: [2]
Hash Key: (6, 7, 0, 0) -> Vectors: [3]

Layer 2 hash index:
Hash Key: (0, 3, 1, 2) -> Vectors: [0]
Hash Key: (0, 1, 3, 6) -> Vectors: [1]
Hash Key: (0, 1, 4, 0) -> Vectors: [2]
Hash Key: (1, 0, 0, 5) -> Vectors: [3]
PS C:\Users\vibhu\OneDrive\Desktop\CSE-515-Project>
```

Fig 13:For the given input of vectors, layers, as well as other inputs, we get the Hashing index and final result.

The user has provided the input of the number of layers as 2, and the hashes per layer is 4. The number of vectors has been put as 4 and each vector has 4 dimensions. Vector 1 is (1,5,2,3), vector 2 is (5,2,7,3), vector 3 is (1,7,2,9), and vector 4 is (9,2,5,0). We get the hyperplanes for each layer. For each layer, The Hash key maps to the vectors. For instance, Hash key:(3,2,0,0) contains vector 0, i.e., our first vector input. This means that the vector lies in the 6th bucket of the 1st hyperplane, the 2nd bucket of the 2nd hyperplane, the 0th bucket of the 3rd hyperplane, and the 0th bucket of the 4th hyperplane.

8.6 Task 3b:

```
PS C:\Users\vibhu\OneDrive\Desktop\New folder> & C:/Users/vibhu/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/vibhu/OneDrive/Desktop/
New folder/try.py"
Please provide the input for 1 - Layer3 + PCA, 2 - Avgpool + SVD, 3 - KMeans + HOG: 1
The query is:  SELECT videoID, Layer_3 FROM data
Enter the query videoID: 10
Enter the number of similar videos to retrieve: 10
Unique Candidates: 2992
Overall Candidates: 3826
Video 10: Distance 0.0000
Video 6342: Distance 0.6318
Video 4616: Distance 0.6382
Video 1226: Distance 0.6587
Video 6691: Distance 0.6780
Video 2182: Distance 0.6887
Video 5488: Distance 0.6945
Video 2669: Distance 0.6954
Video 3633: Distance 0.6978
Video 6745: Distance 0.7037
PS C:\Users\vibhu\OneDrive\Desktop\New folder>
```

Fig 14: Input from the user, and the most similar videos are found using the LSH mapping

In Fig 14, the user has chosen the input of the visual model as layer_3+PCA, query ID as 10, and the number of similar videos to be retrieved be 10. The first is the video being compared to itself, hence the distance shown is 0. Fig 15 shows the thumbnails of the videos.

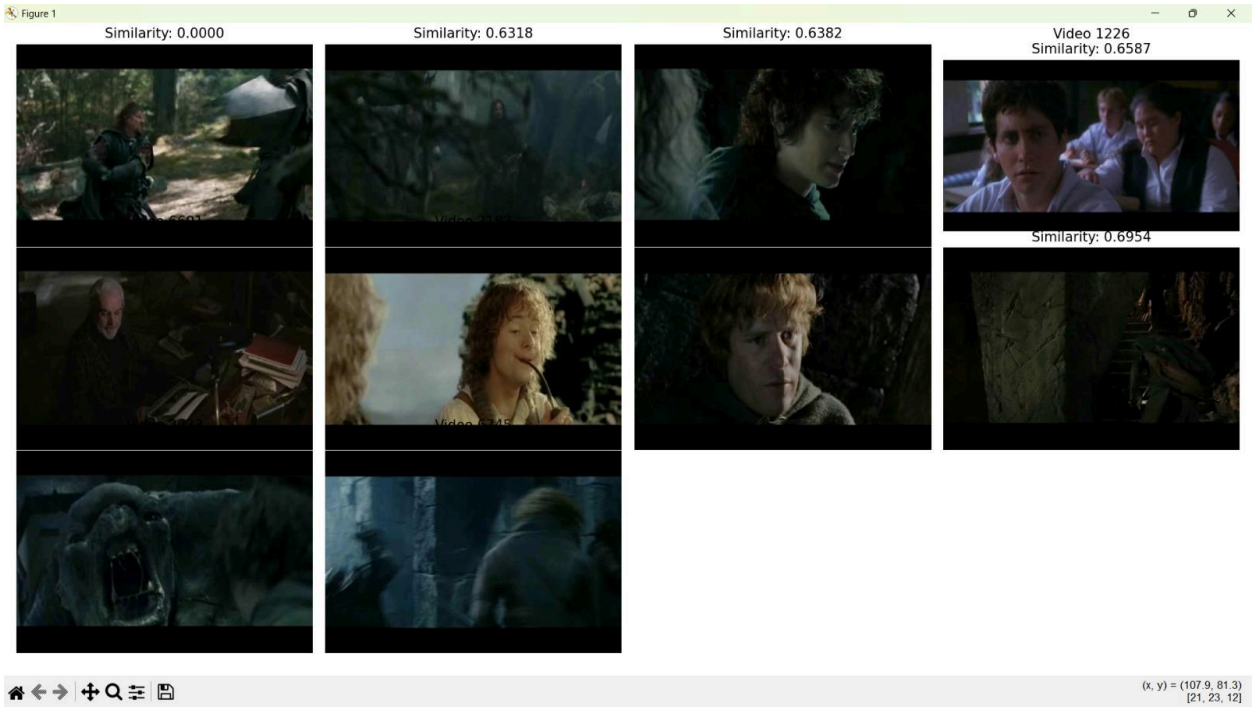

Fig 15: Thumbnails of the videos which are similar to the user queries.

9. Conclusion:

As part of this project, the inherent dimensionality for all the labels that are in the dataset is found. Spectral clustering was implemented to visualize using MDS and as video thumbnails. Using KMeans clustering the latent models are visualized using MDS and group of labels by varying number of clusters. For the given query video, selected user space, and similar labels are predicted using classification techniques like KNN and SVM. Also, Locality Sensitive Hashing which is an index structure was implemented using the latent models for all the videos in the database.

10. Appendix

1. Abhishek Mulasi: Task-1a, Task-3, Readme, Documentation.
2. Sri Rupin Potula: Task-0a, Task-1b, Task-2, Task-3b, Readme, Documentation.
3. Vibhu Dixit: Task-1b,Task-3, Documentation.

11. bibliography

1.https://serre-lab.clps.brown.edu/resource/hmdb-a-large-human-motion-database/#Downloads
2.Yoon, H., & Han, J. H. (2022). Content-based video retrieval with prototypes of deep features. IEEE Access, 10, 30730-30742.