

University of Alabama in Huntsville

LOUIS

Theses

UAH Electronic Theses and Dissertations

2019

Temporal querying of faces in videos using bitmap index

Buddha Raj Shrestha

Follow this and additional works at: <https://louis.uah.edu/uah-theses>

Recommended Citation

Shrestha, Buddha Raj, "Temporal querying of faces in videos using bitmap index" (2019). *Theses*. 641.
<https://louis.uah.edu/uah-theses/641>

This Thesis is brought to you for free and open access by the UAH Electronic Theses and Dissertations at LOUIS. It has been accepted for inclusion in Theses by an authorized administrator of LOUIS.

**TEMPORAL QUERYING OF FACES IN VIDEOS USING
BITMAP INDEX**

by

BUDDHA RAJ SHRESTHA

A THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in
The Department of Computer Science
to
The School of Graduate Studies
of
The University of Alabama in Huntsville

HUNTSVILLE, ALABAMA

2019

In presenting this thesis in partial fulfillment of the requirements for a master's degree from The University of Alabama in Huntsville, I agree that the Library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by my advisor or, in his/her absence, by the Chair of the Department or the Dean of the School of Graduate Studies. It is also understood that due recognition shall be given to me and to The University of Alabama in Huntsville in any scholarly use which may be made of any material in this thesis.

Buddha R Shrestha

Buddha Raj Shrestha


04/05/2019

(date)


THESIS APPROVAL FORM

Submitted by Buddha Raj Shrestha in partial fulfillment of the requirements for the degree of Master of Science in Computer Science in Computer Science and accepted on behalf of the Faculty of the School of Graduate Studies by the thesis committee.


We, the undersigned members of the Graduate Faculty of The University of Alabama in Huntsville, certify that we have advised and/or supervised the candidate of the work described in this thesis. We further certify that we have reviewed the thesis manuscript and approve it in partial fulfillment of the requirements for the degree of Master of Science in Computer Science in Computer Science.



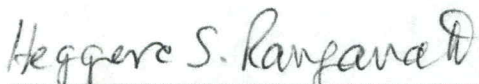
Dr. Ramazan Aygun 4/5/2019 Committee Chair
(Date)



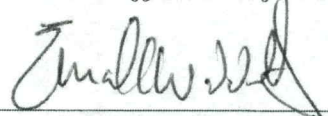
Dr. Haeyong Chung 4/5/2019 (Date)




Dr. Tathagata Mukherjee 4/5/2019 (Date)



Dr. Heggere S. Rangamath 4-5-2019 Department Chair
(Date)



Dr. Sundar Christopher 4-5-19 College Dean
(Date)



Dr. David Berkowitz 4/23/19 Graduate Dean
(Date)


ABSTRACT

School of Graduate Studies
The University of Alabama in Huntsville

Degree Master of Science College/Dept. Science/Computer Science
in Computer Science
Name of Candidate Buddha Raj Shrestha
Title Temporal Querying of Faces in Videos Using Bitmap Index

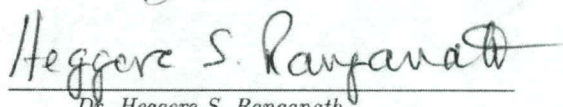
Bitmap indexing has recently been used for big data systems especially for column-based retrieval. In this paper, we study bitmap indexing for temporal querying of faces that appear in videos. Although bitmap index is suitable for finding records that satisfy a condition, its application to databases having temporal context is not straightforward. We consider temporal operators such as *next* and *eventually*, as well as queries for co-appearance. After recognizing faces and indexing them, our method can be used to filter and extract intervals of appearance and can also be used to compute additional conditions using bitwise operations. We have applied our method on a sample video database to show how bitmap index can be used for temporal querying of faces for retrieving relevant intervals and videos.

Abstract Approval: Committee Chair




Dr. Ramazan Aygun

Department Chair



Dr. Heggere S. Ranganath

Graduate Dean



Dr. David Berkowitz

ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest gratitude to Dr. Ramazan Aygun, who took the time to provide constant suggestions and helping to refine my work, without which this thesis would not have been possible. I would also like to extend my sincere gratitude towards Dr. Haeyong Chung and Dr. Tathagata Mukherjee for providing valuable suggestions to improve my thesis.

Finally, I would also like thank my family and friends who supported me and helped me through the thesis.

TABLE OF CONTENTS

Chapter	
1	Introduction 1
1.1	Motivation 1
1.2	Research Problem 2
1.3	Our Approach 3
1.4	Thesis Organization 4
2	Background 5
2.1	BitMap Index 6
2.2	Face Video Retrieval 8
2.2.1	Pre-processing the input 12
2.2.2	Detecting Faces 12
2.2.3	Detecting Facial Landmarks and Aligning Faces 13
2.2.4	Extracting Embeddings 14
2.2.5	Similarity Search 15
2.3	Indexing for Temporal data 17
2.4	Summary 18
3	Method 19
3.1	Dataset 20

3.2	Video parsing	20
3.3	Creation of Bitmap index	23
3.4	Querying in Videos	26
3.4.1	Co-appearance Query	27
3.4.2	Next-appearance Query	28
3.4.3	Prior-appearance Query	30
3.4.4	Eventual-appearance Query	30
3.5	Summary	30
4	Experiments and Results	32
4.1	User Interface for Temporal Querying of Faces in Video	32
4.2	Accuracy	35
4.3	Comparison of Temporal Querying	37
4.3.1	Finding videos	38
4.3.2	Checking and computing intervals	39
4.3.3	Pre-indexing	42
4.4	Summary	44
5	Conclusion	45
5.1	Future Works	46
6	REFERENCES	48

CHAPTER 1

INTRODUCTION

1.1 Motivation

In just the last ten years, there has been a tremendous increase in the number of videos published and viewed each and every day. With more than 3 billion videos being uploaded per day [1] on YouTube alone, the interest in querying information from those videos in order to capture the desired content is increasing day by day. Generally, videos contain a great deal of content in the form of raw data and features that make the task of arranging the data in proper structure more difficult. Hence, implementing the ability to classify video-related data represents a burgeoning and exciting research area [2]. In order to manage this task on a large scale, we need to develop a proper and efficient indexing structure, along with simplistic bridging from high-level retrieval query to actual computation.

Among the categorizing options for video content, face searching in videos is gaining popularity—particularly due to the rise of social media. To be able to identify the same face with a high degree of accuracy is useful for a wide range of applications, such as analyzing a large number of videos where a particular person appears, fast

forwarding to jump to a section where a specific person appears, or even locating and tracking suspects in crime scenes from surveillance videos.

Even though modern face-recognition algorithms [3] [4] [5] have proven to achieve near-human accuracy in various image datasets, querying and retrieval of peoples faces from video content remains quite challenging, due to the large volume of content present in the videos and the time required to process and categorize all that information.

1.2 Research Problem

The increased use of video based multimedia applications in the past few years has created a demand for a strong video database support which includes efficient query processing capabilities. Since video datasets could have unstructured nature with a heavy variation in size and complexity, it generates a key problem in the query processing stage that presents a lot of research challenges [6].

In traditional database systems, the nature and structure of data are pre-defined, which utilizes relational algebra algorithms to select the ‘optimal’ execution plan. This is not feasible in the case of video data, so we require to find a structure that could properly reflect the data with an efficient query processing step. In our research, we hypothesize that transforming the video data in the form of bitmaps can be used for temporal querying.

One of the challenges of multimedia applications is user satisfaction [7]. Despite the potential power of an algorithm, if users are not satisfied with its functionality for any reason, it will receive little attention. Herein, we focus on retrieving faces

from a video database based on its temporal context. We apply operators such as co-appearance, next, eventually, etc. for videos. Our goal is to answer questions such as "Does Person A and Person B appear together?", "Does Person A appear right after Person B?", or "Does Person A appear some time after person B appears?". The main challenge for implementing these queries is the use of a bitmap index which has a bit assigned as 1 for a specific record for a specific condition.

1.3 Our Approach

In this thesis, we present a novel video-retrieval technique for video databases, which utilizes bitmap indexing for mapping faces present in video clips or intervals in order to provide simple and efficient temporal querying. This bitmap indexing approach is particularly suitable for column-based retrieval [8]. Specifically, the primary contribution of this work is a new retrieval approach to incorporate multiple conditions to video queries focusing mainly on human faces, such as (a) locating videos and intervals where multiple faces appear together, (b) identifying instances where a person appears temporally directly after another person, and (c) determining if an individual appears sometime later after the appearance of another person.

In a variety of video-searching applications, the process of temporal querying becomes more complex if it requires the inclusion of more query elements (i.e., faces). Since our technique employs images as the required addition query element, and the database contains the mappings of faces in videos with their corresponding timeline in the form of bitmaps, the videos in which the faces of interest appear can be filtered out, after which the temporal regions can be identified with respect to given

conditions using bitwise operations. Importantly, this technique is generalizable to various video-retrieval applications, since it can handle multiple query elements without increasing computational complexity.

We propose an indexing scheme targeting only on faces, where we create a bitmap for each occurrence of a face in a video. Using these bitmaps we intend on using bit manipulation strategies for computing the query components and eventually return the query results. Once the framework is implemented, we will perform experiments on real datasets and compare the efficiency with other strategies to validate that the proposed approach results in a generalized and efficient solution.

1.4 Thesis Organization

The remainder of the thesis is organized as follows. Chapter 2 covers brief background on bitmaps and similarity search and provides research studies that are related to this thesis. Chapter 3 presents how the dataset is built, the methodology that was used to create the indexing structure and detailed steps on computing the query elements. Chapter 4 provides the results and analysis of the experiments based on accuracy and efficiency of the system. Finally in Chapter 5 we conclude the thesis with possible future directions that could be explored.

CHAPTER 2

BACKGROUND

Since videos generally have lots of content, it is particularly necessary to parse the contents and select only the relevant information, followed by storing it in a proper format so that it could be retrieved easily later. In this work case, we focus on tracking faces along with their time of occurrence. There are a number of state of the art face detection algorithms [3,4] that are very effective for such purposes.

Once the parsing is complete, the information can then be used to create suitable indexing structure. This structure is very important in making a very efficient query processing flow. There are different indexing schemes [9,10] that focus on spatio-temporal context. In this thesis, we study whether bitmap index can be used for temporal querying.

In this section, we firstly provide background on bitmap index. We then describe briefly some of the previous works done in face video retrieval and temporal querying.

2.1 BitMap Index

A bitmap index is a special type of database index which uses bitmaps to optimize search for low-variability data (which have a modest number of distinct values). Basic bitmap indexes use one bitmap for each distinct value but it is also possible to reduce the number of bitmaps using different encoding methods [11].

Student ID	Gender
1001	F
1002	M
1003	F
1004	F

↓

Female	Male
1	0
0	1
1	0
1	0

Table 2.1: Sample database for bitmap indexing.

A simple example is shown in Table 2.1. In this case, a two dimensional array is constructed for 'Gender' column for bitmap indexing. There will be a row in the bitmap index table corresponding to each row in the data table and each value in the row will be marked as '1' or '0' based on the condition, whether the student is 'Male' or 'Female'.

A bitmap index can also be created for columns having many distinct values. This is done by including more columns in the bitmap array. In Table 2.2, there are

Student ID	Age Range	
1001	18-22	
1002	18-22	
1003	23-27	
1004	27-30	

↓

	18-22	23-27	27-30
1	0	0	
1	0	0	
0	1	0	
0	0	0	1

Table 2.2: Bitmap index for attribute with high variability.

four different age ranges, which translate into three distinct columns in the bitmap index.

We can take real advantage of bitmap indexes when we can join several bitmap indexes together by applying bitwise logical operations between different bit arrays.

Student ID	Gender	Married	Age Range
1001	F	N	18-22
1002	M	Y	18-22
1003	F	Y	23-27
1004	F	Y	27-30

↓

	Male	Married	27-30
0	0	0	0
1	1	0	0
0	1	0	0
0	1	1	1

Table 2.3: Bitmap index for multiple attributes.

In Table 2.3, while none of the columns has a lot of uniqueness, by having a bitmap index on each column and then conducting a logical ‘AND’ operation, a demographic query such as ‘finding all the males who are married and in the age range of 27 to 30’ can easily be answered. With bitmap index, it is possible and desirable to have many of such bitmap indexes across various columns to perform these bitwise logical operation between the different bit arrays.

There are other advantages of using bitmap indexes besides the simplicity of its structure. The bit arrays like this can be compressed [12] making the structure space efficient. If compressed they do not take much space on disk and the read operations can be done very quickly. One disadvantage of bitmap index is that if there are high number of distinct values (more than several thousands) then we will need to have same number of columns in our bitmap index which might not be an efficient approach [13]. This is also one of the deciding factors whether to use a bitmap index or not.

2.2 Face Video Retrieval

In recent years, there have been a lot of studies going on in the field of face video retrieval [14–17]. [14] provides a single image based face matching method with partial occlusion and expression change allowing robustness to pose, expression, illumination and background clutter. [15] is a person retrieval system by comparing the Chi-square distance between sets of faces which are represented as distributions in the form of histogram to retrieve a ranked list of shots of a queried person. These implementations mainly focus on spotting a person in a single video and do not address a solution to

find the shots of a given person in a large database of videos. [17] tries to address this problem by proposing a heterogeneous hashing-based retrieval where two entirely heterogeneous spaces (e.g., Euclidean space and Riemannian manifold) are embedded into a common discriminant Hamming space. In their study, the query face is a point in Euclidean space and the video clip is modeled as a point on Riemannian manifold. While this addresses the problem of querying in a large scale databases, it is still only applicable for a single person search. In this paper, we implement a generic approach towards face video retrieval, where we can add the number of faces in the query to find out if they appear together in a particular instance using bitmap index. This makes our approach different from the above implementations.

In recent years there has been huge improvement in face detection and recognition. For a given input image, face recognition system typically runs face detection first to isolate the faces present in the image. Once the faces are detected, each face is preprocessed and represented into a low-dimensional representation (embedding). This low-dimensional representation is key for efficient classification. In this process, the face representation needs to be resilient to intra-personal image variation such as age, expressions and styling. These representations are mapping of face images into a compact Euclidean space, which are trained usually through deep learning network [4].

The breakthrough in deep learning based networks occurred when AlexNet won the ImageNet competition [18]. The networks for face recognition learn multiple levels of representations that correspond to different levels of abstraction, showing strong invariance to the face pose, lighting and expression changes. Inspired by

the extraordinary success on the ImageNet challenge, there has been significant number of deep learning research studies based on convolutional neural network (CNN) architectures [19], such as VGGNet [20], GoogleNet [21], ResNet [22] and SENet [23] which are widely used as the baseline model in face recognition. Still researchers are trying to improve learning in these models even more and use them for multiple applications. For example, the introduction of large margin cosine loss (LMCL) [24] function aims to maximize inter-class variance while minimizing intra-class variance. In [24], it is shown that LMCL achieves consistent state of the art results on the benchmark datasets. ArcFace [25] introduces another loss function named additive angular margin loss function that aims to obtain rich discriminative features for face recognition. The performance of ArcFace achieves promising results using their loss function that has high discriminative power to obtain the deeply learned features. ArcFace outperforms current state of art approaches on LFW, YTF, CALFW and CPLFW datasets.

Hyperface [26] is a framework based on CNNs for simultaneous face detection, facial landmarks localization, head pose estimation and gender recognition from a given image using the synergy among features. The authors show that learning correlated features simultaneously can boost the performance of individual tasks. Yang et al. [27] propose to neural aggregation network that produces a compact and fixed-dimension feature representation for recognition. The network consists of two modules: embedding module and aggregation module for adaptive aggregation. Their results show that the network automatically learns to advocate high-quality face images while repelling low-quality ones such as blurred, occluded and improperly

exposed faces. This process helps consistently outperform naive aggregation methods while achieving the state-of-the-art accuracy.

While these studies explore the boundaries of deep learning based networks to solve problems related to face recognition, other research studies such as [3, 5, 28] have published their own pre-trained networks to accelerate the face recognition research community with the motive to perform more research and create more useful applications that rely on face recognition.

In this thesis, we use Dlib [5], which is a state of the art library for face detection and tracking. Dlib has a pretrained neural network model for this purpose and its model consists of ResNet network [22], which has a building block as shown in Figure 2.1.

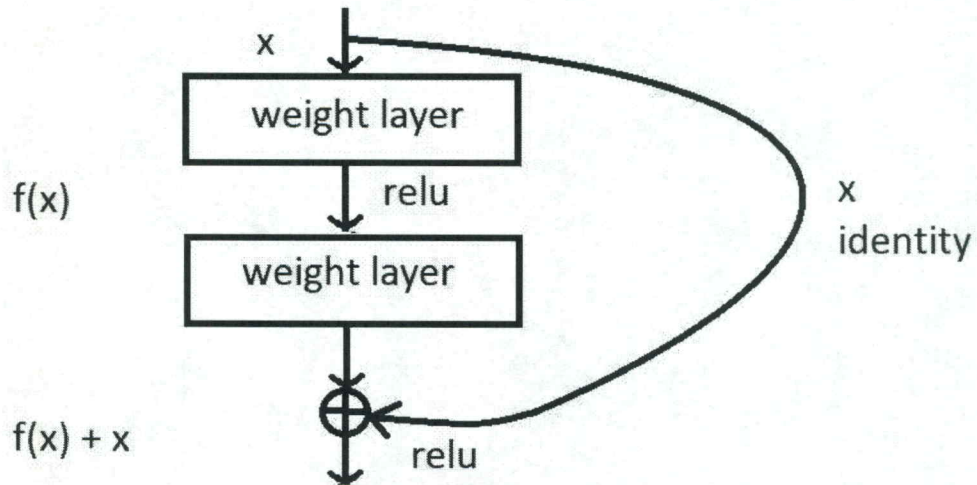


Figure 2.1: Resnet Building block

A brief review of the steps involved while tracking faces using Dlib library are explained in the following section.

2.2.1 Pre-processing the input

The robustness of detection can be improved by preprocessing the input image. For example, the lightning condition for every image or each frame of video will not be optimal, so it is a good option to preprocess the input which helps to obtain higher accuracy. The preprocessing mostly includes operations such as the gamma correction for enhancing the image quality (Figure 2.2).



Figure 2.2: Gamma correction

2.2.2 Detecting Faces

This is a crucial part of the process since we do not want to miss out on a possible face that can be queried later. Among numerous approaches, Histogram Oriented Gradient (HOG) is one of the most popular [29] feature extracting approaches. This technique counts occurrences of gradient orientation in localized portions of an image - detection window, or region of interest (ROI), which is similar to that of edge

orientation histograms, but differs in that it is computed on a dense grid of uniformly spaced cells and uses overlapping local contrast normalization for improved accuracy. This then is applied to a sliding window detector over an image or frame and HOG descriptor is calculated for each position. Each HOG descriptor that is computed is fed to a pre-trained SVM classifier to find the part of our image that looks the most similar to a known HOG pattern (Figure 2.3).

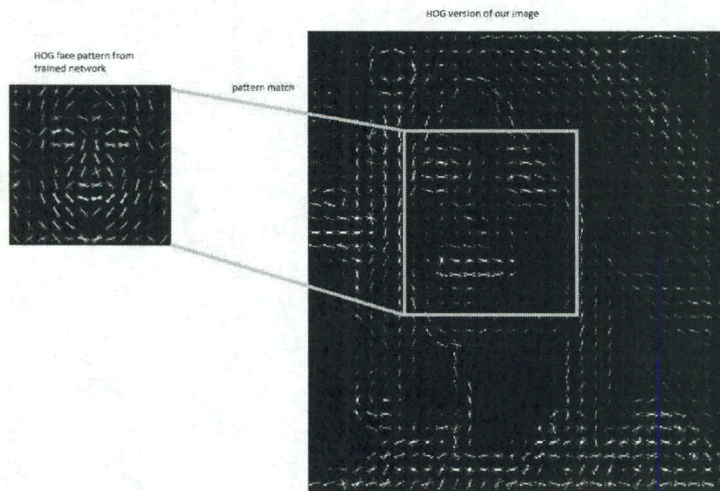


Figure 2.3: HOG computation

2.2.3 Detecting Facial Landmarks and Aligning Faces

After detecting faces, important facial structures on the face are detected using shape prediction methods. [30] is one of the popular algorithms used to detect facial landmarks in real-time with high quality predictions. Also, it is better to align the faces with some reference template because a tilt or a turn of face may cause

the representation or the coordinates of landmarks to be shifted by a certain factor.

Figure 2.4 shows an example of aligning faces.

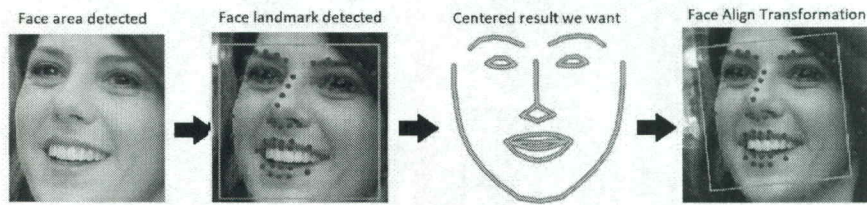


Figure 2.4: Landmark detection with Affine Transformation

2.2.4 Extracting Embeddings

Extracting features that are similar for the same faces and can distinguish different faces is important. Generally, a face is represented as a feature vector and the distance between feature vectors is used to determine the similarity between faces. Feature extraction is usually done through training a deep neural network (a convolutional neural network) with the objective of minimizing the distance between the output feature vectors of the same face and maximizing the distance between different faces.

After training a model, the input image can be fed to the model to generate a feature vector corresponding to the person in the image. An example is provided in Figure 2.5.



```
[0.7934545 0.26255756 0.16659186 0.24863714 0.60531085 0.5592173  
0.24518489 0.6186619 0.00363487 0.09207082 0.47543688 0.13274324  
0.20213747 0.45695136 0.94635125 0.25691449 0.44013757 0.05081216  
0.19786438 0.20172843 0.56587849 0.16020489 0.83626032 0.47261184  
0.78066301 0.2447687 0.68320858 0.66539402 0.41572631 0.43541266  
0.19787807 0.49727512 0.31513145 0.81041694 0.62371707 0.44377604  
0.17760107 0.76743997 0.71814297 0.56604348 0.78434479 0.54418729  
0.75392101 0.87756698 0.52627776 0.94469383 0.97583139 0.53477271  
0.61022198 0.1397516 0.82350748 0.86637927 0.38382302 0.53266077  
0.57908278 0.53134402 0.08668357 0.26358521 0.54902421 0.65237925  
0.68296196 0.42882182 0.46705074 0.55802962 0.34448255 0.63758427  
0.91616792 0.68093308 0.30593387 0.77115668 0.17912108 0.29391654  
0.11499991 0.31574968 0.15901544 0.24734581 0.11729496 0.50664648  
0.66213121 0.0913827 0.54324182 0.39840328 0.64488995 0.28856211  
0.43305913 0.68535637 0.9239709 0.44710139 0.80105124 0.38961319  
0.07740444 0.78075551 0.3578415 0.47529117 0.89762679 0.18405672  
0.50706335 0.3909206 0.25100457 0.52295131 0.53879282 0.64700939  
0.36707598 0.20636555 0.06775962 0.96870149 0.61984931 0.33262885  
0.19905535 0.93268371 0.18701782 0.60037088 0.12646802 0.01128069  
0.6484856 0.52183142 0.35351333 0.27450675 0.60118823 0.15141763  
0.3002626 0.52006357 0.64021841 0.24151184 0.33063946 0.03642889  
0.1846083 0.95715698]
```

Figure 2.5: 128d vector embedding generated from given image

2.2.5 Similarity Search

Traditional databases are made up of structured tables containing symbolic information where tables are viewed as relations and their columns are viewed as attributes [31]. For example, to represent an image collection, we would need to create a table having one row per indexed photo. Each row would have its own image identifier and can be linked to entries from another tables as well.

There are different techniques to generate high dimensional descriptors (for text, images etc.) that are much more powerful and flexible than fixed symbolic representation. But, the problem is that the traditional databases can be queried efficiently using SQL, which do not support high dimensional descriptor representations.

In similarity search, the descriptors (vector representation) are designed to produce similar vectors for similar images where similar vectors are defined as those that are nearby in space [32]. For example, the images of faces of the same person will have a smaller distance as compared to that of the images of a different person.

So, for similarity search, a query vector should return the list of database objects that are nearest to the given vector in terms of a distance measure.

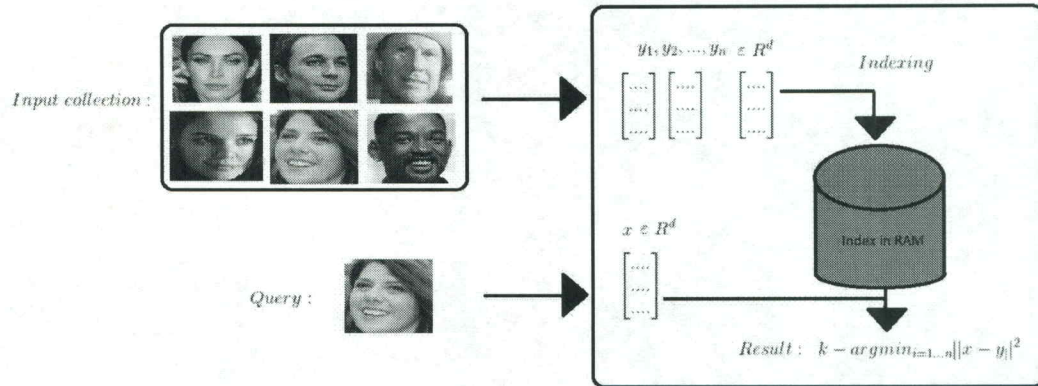


Figure 2.6: Similarity search block diagram

Figure 2.6 represents a block diagram that describes the similarity search process. First, the collection of input is loaded into the memory. Then the query object is compared with each of the input to return the ‘ N ’ similar objects [33]. However, this is a brute-force method that computes all the similarities – exactly and exhaustively. Implementing such algorithms is not efficient and can impact the performance of the system.

[34] is a state of the art similarity search library that spans a wide spectrum of usage trade-offs with optimized memory usage and GPU implementation for the most relevant indexing methods.

In summary, face recognition/tracking systems has a flow block diagram as shown in Figure 2.7 [28].

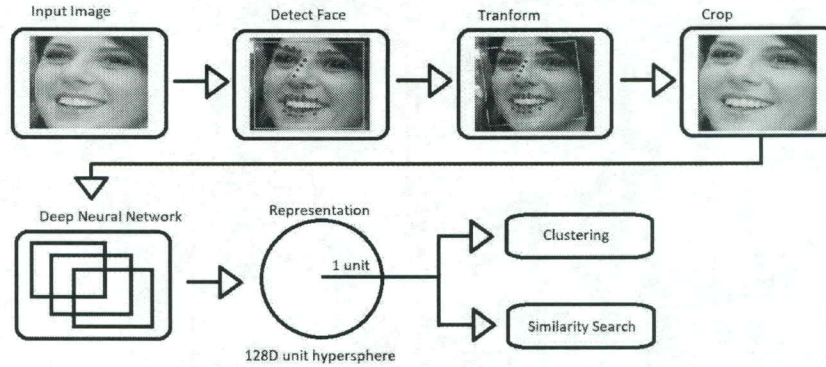


Figure 2.7: Face Recognition/Tracking block diagram

2.3 Indexing for Temporal data

A video database features a spatio-temporal context. Queries having temporal or spatio-temporal query conditions are difficult to handle using traditional index structures. To perform such queries, special index structures, such as semantic sequence state graph (S^3G) [9, 10], have been developed. S^3G represents object locations as a state and link states when objects change locations. Spatio-temporal retrieval can be imitated using SQL patterns. Jain et al. [35, 36] developed strings based on a grammar to represent a spatio temporal context and show how SQL patterns can be used for spatio-temporal querying. The downside, however, is that it cannot benefit from existing index structures in relational databases. In this paper, we focus on temporal querying using a bitmap index. Spatial context will be added in our future work. Our temporal querying is based on linear temporal logic [37]. For example, eventually/globally operator checks if the condition becomes true at a point in the path. Next operator checks whether the condition becomes true in the next time point.

Bettaiah and Aygun used gaming controller to specify spatio-temporal queries that include 'next' and 'eventually' operators [38, 39].

2.4 Summary

There are effective face recognition and similarity search algorithms that extract feature vectors for faces and run quick similarity search. Our goal is to develop a type of face retrieval method based on temporal context. We plan to utilize bitmap index for achieving temporal querying.

CHAPTER 3

METHOD

The first step in performing an indexing structure for temporal querying is to acquire video dataset. After the dataset is obtained, video parsing is to be done which involves running face detection and tracking on each video and clustering them with respect to faces [40] in order to maintain a mapping of face versus time interval. Using this information, first a bitmap of person versus video is constructed followed by person versus timings bitmap which are maintained in a proper file structure. Once the indexes are constructed, the structure can be used to perform queries like co-appearance, next and eventual. Figure 3.1 summarizes the system in a block diagram.

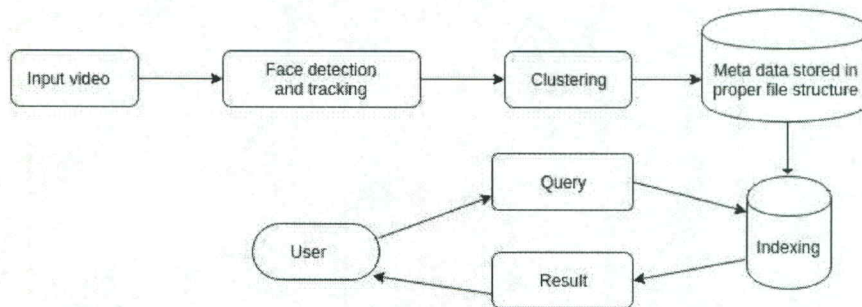


Figure 3.1: System block diagram

3.1 Dataset

For querying faces in videos, we could not find a labeled standard publicly available dataset. Therefore, we created our own dataset using a face detection and recognition library. Nevertheless, we had to manually track the appearance and timings of each person in the video for verifying outputs of the face detection/recognition. We have collected videos of movies and TV shows from multiple sources. Videos varied in length and quality so that a variety of experiments could be run based on those properties. Eventually, we generated a dataset of 30 videos with varying duration from 20 seconds to 30 minutes. We have built this dataset for evaluating our method for temporal querying.

3.2 Video parsing

Index is built by first detecting and recognizing who appear in the videos and then identifying intervals they appear. In this research, Dlib's [5] pre-trained face detector based on ResNet network with 29 convolutional layers [41,42] is used to detect and track faces while sampling the videos.

Figure 3.2 is an example of the facial landmark detector implemented in Dlib that produces 68 landmark points that map to specific facial structures [30]. The Dlib's facial recognition library maps an image of a human face to a 128 dimensional vector space where images of the same person are near to each other and images from different people are far apart. Once 128d feature vector is available, face recognition is accomplished by a similarity or distance function such as Euclidean.

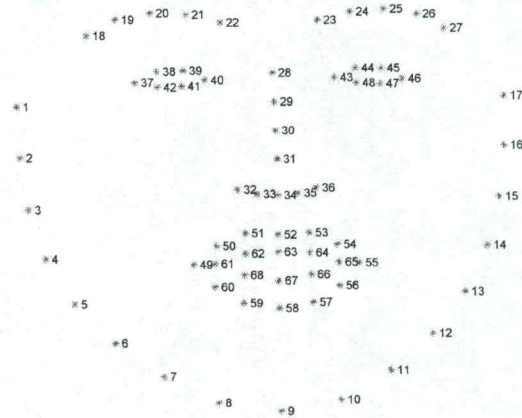


Figure 3.2: 68-landmark representation of face



$$\longrightarrow \mathbf{p}_i = \begin{pmatrix} \mathbf{128d} \\ 0.3412 \\ 0.9323 \\ 0.4421 \\ \vdots \\ 0.8378 \end{pmatrix}$$

Figure 3.3: 128d vector representation of face of person p_i

Figure 3.4 shows an example of the face detection and the corresponding feature vector. After detecting faces, the time intervals in which they appear are tracked. As the number of frames in a video can be very high, running face detection on each frame is not an optimal approach. It is necessary to partition the video into shots for efficient indexing and retrieval of video. Shot transition detection is used to split up a video

into basic temporal units called shots [43]. A shot is a series of interrelated consecutive pictures taken contiguously and representing a continuous action in time and space [44].

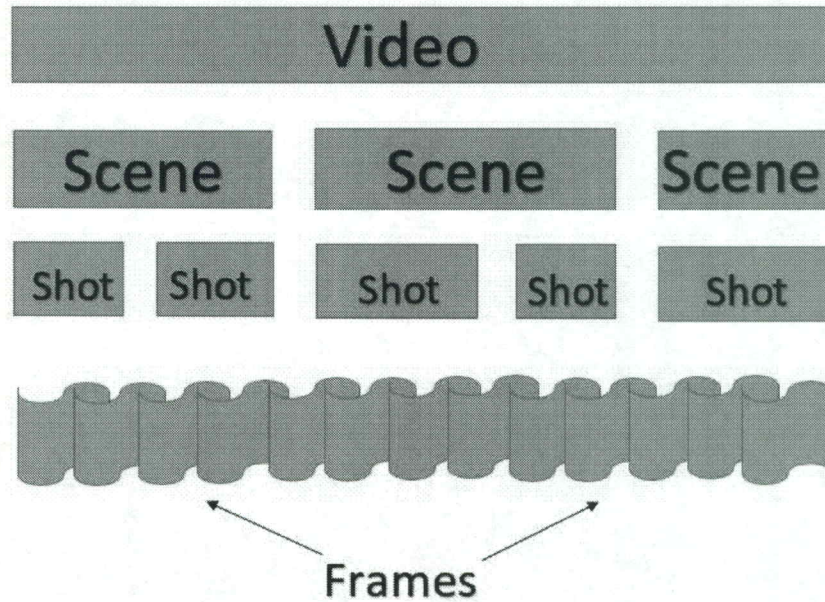


Figure 3.4: Structure of a video

Displaced frame difference [45] is used to find shots and record its duration in the video. After finding the shots and their durations, we detect and track faces appearing only in those shots rather than tracking each frame. Once we have the face track, we apply clustering to obtain unique set of faces in the video and their corresponding timeline of appearance as shown in Figure 3.5. This information is then used to create an indexing structure which will be discussed in the Section 3.3. For clustering, dlib's chinese whisper module is used to cluster unknown faces using a

graph data structure. It runs on linear time and is fast and recommended for large datasets with unknown number of classes [46].

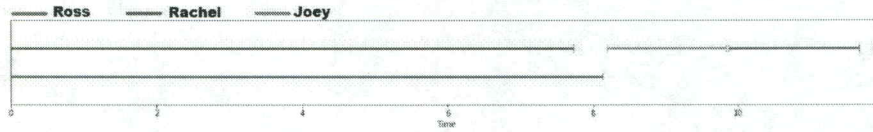


Figure 3.5: Timeline for appearance of people in a video

3.3 Creation of Bitmap index

Our main objective is to use bitmap index for temporal querying. Since videos are continuous, we need to come up with a way to properly structure the bitmap index. For this purpose, we build a two-level bitmap index. The first level maintains whether people appear in the videos or not, whereas the second level maintains the time intervals of people who appeared in a specific video.

In the first level, a bitmap index of person versus videos ($p2v$) is maintained where i^{th} person is denoted as p_i and his or her appearance is determined by the i^{th} row in $p2v$ matrix. Person p_i is represented as a 128-dimensional feature vector as explained in Section 3.2.

Each column in $p2v$ matrix corresponds to the j^{th} video. So, $p2v(i, j)$ corresponds to the bit for the appearance of person p_i in video v_j and is set to 1 if p_i appears in v_j , otherwise, it is set to 0.

Figure 3.6 is a sample $p2v$ matrix which maintains records of whether a person appears in a given video or not.

$$p2v = \left(\begin{array}{c|cccccc} \mathbf{p} & \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3 & \dots & \mathbf{v}_n & \mathbf{v}_{n+1} \\ \hline p_1 & 1 & 1 & 0 & \dots & 1 & 1 \\ p_2 & 0 & 1 & 0 & \dots & 1 & 0 \\ p_3 & 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \dots & \vdots & \vdots \\ p_k & 1 & 0 & 0 & \dots & 1 & 0 \\ p_{k+1} & 0 & 0 & 0 & \dots & 0 & 0 \end{array} \right)$$

Figure 3.6: Person versus video mapping

In the second level, a similar indexing scheme is maintained for each individual video v_j . First, the intervals only where faces appear are filtered out (since intervals without face are not used for further computation). Each interval has its start and end times. These intervals are sorted in ascending order with respect to their start times. Then $p2t^j$ matrix for video v_j is constructed where column t_r represents the r^{th} interval and $p2t^j(i, r)$ corresponds to the bit for the appearance of person p_i in interval t_r and is set to 1 if p_i appears in t_r , otherwise, it is set to 0. The following is a sample matrix:

$$p2t^j = \left(\begin{array}{c|ccccc} \mathbf{p} & \mathbf{t}_1 & \mathbf{t}_2 & \mathbf{t}_3 & \dots & \mathbf{t}_m \\ \hline p_1 & 1 & 1 & 0 & \dots & 1 \\ p_2 & 0 & 1 & 0 & \dots & 1 \\ p_3 & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \dots & \vdots \\ p_j & 1 & 0 & 0 & \dots & 1 \end{array} \right)$$

Figure 3.7: Person versus timing mapping for each video

Figure 3.8 is a sample $p2t$ matrix which maintains records of whether the person appears in a given time interval in that a particular video or not.

$$p2t^n = \left(\begin{array}{c|cccccccc} \mathbf{p} & \mathbf{t}_1 & \mathbf{t}_2 & \mathbf{t}_3 & \mathbf{t}_4 & \mathbf{t}_5 & \mathbf{t}_6 & \mathbf{t}_7 & \mathbf{t}_8 \\ \hline p_1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ p_2 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \end{array} \right)$$

Figure 3.8: Person versus timing mapping for each video

Whenever a new video (v_{n+1}) is added to the database, a new column is added to the $p2v$ matrix, and a new $p2t$ matrix is created for the video. For updating $p2v$ matrix, we run a similarity search against the p column for each person appearing in the video in the $p2v$ matrix. If the similarity score is within the threshold, it indicates that we already have a previous record of that person (e.g., p_i) and need to set $p2v(i, n + 1) = 1$. If the similarity score is beyond the threshold, we add a new $k + 1^{th}$ row (assuming k people in the database so far) and assign $p2v(k + 1, n + 1) = 1$. For any other video v_j , $p2v(k + 1, j)$ is set to 0. Table 3.1 summarizes symbols used in the following sections.

Table 3.1: Table of Symbols

Symbol	Description
$p2v$	person versus video mapping
$p2t^j$	person versus time interval mapping for video v_j
$p2v_k$	person p_k 's row in $p2v$
$p2t_k^j$	person p_k 's row in $p2t^j$
$v_{(i,k)}$	bitmap after computing ' $p2v_i$ & $p2v_k$ '
$\overline{v}_{(i,k)}$	the list of videos where p_i and p_k appear together
$t_{(i,k)}^j$	bitmap after computing ' $p2t_i^j$ & $p2t_k^j$ '
$t_{(i,k)}^j$	the list of time intervals having p_i and p_k together in video v_j
η	bitmap for intersection of intervals between $p2t_i^j$ and $p2t_k^j$
β_k	bitmap corresponding to $p2t_k^j$ after removing η from $p2t_k^j$
$\tilde{\beta}_k$	left shift β_k by one bit
$\alpha_{(i,k)}^j$	bitmap indicating if p_k appears right after p_i in video v_j
$\overline{\alpha}_{(i,k)}^j$	actual intervals where p_k appears right after p_i in video v_j

3.4 Querying in Videos

For querying purposes, the user provides images of people rather than the names of people. Our system extracts 128-dimensional feature vector from the image and runs a similarity search against all faces (column p) in $p2v$ matrix. Once the person is found, we can find the intervals in which this person appears in each video using the corresponding $p2t^j$ matrix for that video. The following example of $p2v_k$ shows sample row of $p2v$ for person p_k . The $p2t_k^j$ is the time interval mapping for person p_k in video v_j .

$$p2v_k = \begin{matrix} & \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3 & \mathbf{v}_4 & \dots & \mathbf{v}_n \\ \left(\begin{array}{cccccc} 1 & 0 & 1 & 0 & \dots & 1 \end{array} \right) \end{matrix}$$

$$p2t_k^j = \begin{matrix} & \mathbf{t}_1 & \mathbf{t}_2 & \mathbf{t}_3 & \mathbf{t}_4 & \dots & \mathbf{t}_m \\ \left(\begin{array}{cccccc} 0 & 1 & 1 & 0 & \dots & 1 \end{array} \right) \end{matrix}$$

To find videos where two or more people appear together, rows of $p2v$ for each person are extracted and bitwise 'AND' is applied.

$$p2v_i = [1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ \dots \ 1]$$

$$\&$$

$$p2v_k = [1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ \dots \ 1]$$

\downarrow

$$v_{(i,k)} = \begin{matrix} & \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3 & \mathbf{v}_4 & \mathbf{v}_5 & \mathbf{v}_6 & \dots & \mathbf{v}_n \\ \left[\begin{array}{cccccc} 1 & 0 & 0 & 0 & 1 & 0 & \dots & 1 \end{array} \right] \end{matrix}$$

$$\overline{v_{(i,k)}} = [\mathbf{v}_1, \mathbf{v}_5 \dots, \mathbf{v}_n]$$

Once the list of videos where a person(s) appears is filtered, the system can compute the intervals in which the queried person(s) appears using the $p2t^j$ mappings for the person in each video. This can help to formulate numerous sequence of events that can be used as additional conditions for a query. Since we are using the bitmap index, the computation for these conditions can be done using bit-wise operations. Our system supports the following types of queries: 1) (co-occurrence) find intervals where person A and person B appear together, 2) (next) find videos where a person appears right after another person, 3) (prior) find videos where a person appears before another person appears, and 4) (eventually) find videos where a person appears sometime after another person appears.

3.4.1 Co-appearance Query

Finding when a person appears or time intervals in which they appear in each video is readily available by $p2t^j$ bitmap index. Using these, time intervals when multiple people appear together can be found.

Co-appearance in a video does not necessarily mean people appear at the same time in the video. Using $p2t^j$ matrix for a common video v_j , a similar conjunction (AND,&) for rows of these people is applied while extracting the time intervals that these people appear together. For example, consider $p2t_i^j$ and $p2t_k^j$ represent the bitmaps for time intervals for a specific video (v_j) for people p_i and p_k . Their conjunction yields bitmaps that indicate the interval of co-appearance represented by $t_{(i,k)}^j$. We then find the indices of set bits of $t_{(i,k)}^j$, which will be the actual time

intervals where the people appear together, represented by $\overline{t_{(i,k)}^j}$. Sample steps involved while calculating the intervals are provided below.

$$\begin{array}{c}
 \boxed{
 \begin{array}{l}
 p2t_i^j = [1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ \dots \ 1] \\
 \& \\
 p2t_k^j = [1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ \dots \ 1]
 \end{array}
 } \\
 \downarrow \\
 \begin{array}{l}
 \mathbf{t}_1 \quad \mathbf{t}_2 \quad \mathbf{t}_3 \quad \mathbf{t}_4 \quad \mathbf{t}_5 \quad \mathbf{t}_6 \quad \mathbf{t}_7 \quad \dots \quad \mathbf{t}_n \\
 t_{(i,k)}^j = [1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ \dots \ 1] \\
 \overline{t_{(i,k)}^j} = [\mathbf{t}_1, \mathbf{t}_6 \dots, \mathbf{t}_n]
 \end{array}
 \end{array}$$

3.4.2 Next-appearance Query

The *next* query can be used to find whether a person appears after another person. For example, in a video v_j , we want to check if person ' p_k ' appears right after person ' p_i ' (not in the same frame). This is achieved by first removing the co-appearing time intervals (η) from $p2t_k^j$. The result is then left-shifted by one bit ($\tilde{\beta}_k$), followed by computing bitwise 'AND' with $p2t_i^j$:

$$\begin{aligned}
 \eta &\leftarrow p2t_k^j \& p2t_i^j \\
 \beta_k &\leftarrow p2t_k^j \oplus \eta \\
 \alpha_{(i,k)}^j &\leftarrow p2t_{i,j} \& \tilde{\beta}_k
 \end{aligned}$$

where ' η ' indicates the bitmap for co-appearing intervals which is removed from $p2t_k^j$ using the bitwise 'XOR' operator in the second step to get ' β_k '. The left-shift operation $\tilde{\beta}_k$ is indicated by left-arrow over the bitmap variable as β_k .

If $\alpha_{(i,k)}^j$ returns a bitmap with non-zero value, it means that there is at least a sequence in the video where p_k appears right after p_i . The actual intervals can be calculated using the indices of $\alpha_{(i,k)}^j$, represented by $\overline{\alpha_{(i,k)}^j}$. The following shows sample calculations.

$$\begin{array}{c}
 \boxed{
 \begin{array}{l}
 p2t_i^j = [1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ \dots \ 1] \\
 \quad \quad \quad \& \\
 p2t_k^j = [1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ \dots \ 1]
 \end{array}
 } \\
 \downarrow \& \\
 \eta = [1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ \dots \ 1] \\
 \boxed{
 \begin{array}{l}
 [1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ \dots \ 1] \\
 \beta_k = [1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ \dots \ 1] \\
 \quad \quad \quad \oplus
 \end{array}
 } \\
 \downarrow \\
 \beta_k = [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ \dots \ 0] \\
 \boxed{
 \begin{array}{l}
 \tilde{\beta}_k = [0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ \dots \ 0] \\
 \quad \quad \quad \& \\
 p2t_k^j = [1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ \dots \ 1]
 \end{array}
 } \\
 \downarrow \\
 \alpha_{(i,k)}^j = \begin{array}{c} \mathbf{t_1} \ \mathbf{t_2} \ \mathbf{t_3} \ \mathbf{t_4} \ \mathbf{t_5} \ \mathbf{t_6} \ \mathbf{t_7} \ \dots \ \mathbf{t_n} \\ [0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ \dots \ 0] \end{array} \\
 \overline{\alpha_{(i,k)}^j} = [\mathbf{t_6}]
 \end{array}$$

Figure 3.9: Allen's relations to illustrate co-appearance, next-appearance and eventual-appearance

3.4.3 Prior-appearance Query

The prior-appearance query is used to find whether a person appears before another person appears. This can be done by just comparing two bitmaps. For example, if $\beta_k > p2t_i^j$, person p_k appears before p_i .

3.4.4 Eventual-appearance Query

Similar to the prior-appearance query, eventual-appearance query can be used to determine if a person appears some time after the other person. This can be done by just comparing two bitmaps. For example if $p2t_i^j > \beta_k$, person p_k appears sometime after p_i .

We can also relate these queries with respect to Allen's interval algebra, which is basically a calculus for temporal reasoning, introduced by James F. Allen in 1983. This consists of thirteen basic relations between time intervals that are distinct, exhaustive, and qualitative [47]. Figure 3.10 shows all the possible relations that two definite intervals can have. We use these relations to illustrate whether they satisfy co-appearance, next-appearance or eventual-appearance. Each relation is defined graphically by a diagram which relates two definite intervals A and B.

3.5 Summary

In this section, we firstly explained the face recognition method. We explained our method that utilizes the bitmap index. There are two bitmap index structures:

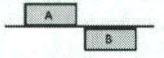
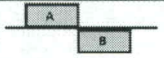
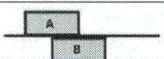

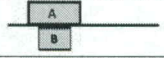
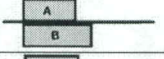
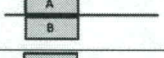

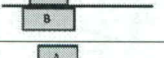

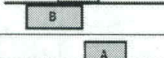
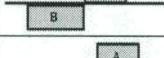
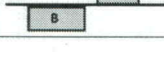
Relation	Illustration	Co-appearance	Next	Eventually	Interpretation
Precedes		No	No	Yes	B is preceded by A
Meets		No	Yes	Yes	A ends when B begins
Overlaps		Yes	Yes	Yes	A overlaps B
Finished by		Yes	No	No	A is finished by B
Contains		Yes	No	No	B during A
Starts		Yes	Yes	Yes	A starts B
Equals		Yes	No	No	A equals B
Started by		Yes	No	No	B is started by A
During		Yes	Yes	Yes	A during B
Finishes		Yes	No	No	A finishes B
Overlapped by		Yes	No	No	B is overlapped by A
Met by		No	No	No	B ends when A begins
Preceded by		No	No	No	A is preceded by B

Figure 3.10: Allen's relations to illustrate co-appearance, next-appearance and eventual-appearance

person versus video and person vs time intervals in a video. We explain how these bitmap index structures are used for co-appearance, next, and eventual/prior queries.

CHAPTER 4

EXPERIMENTS AND RESULTS

We have developed a user interface to build our temporal queries. Then, multiple experiments were performed using the dataset noted in Section 3.1 to evaluate the level of accuracy and efficiency achieved by the system. Also, results from some additional strategies are included that can be used to further improve the system. All experiments were performed on a quad-core Intel(R) Core(TM) i5-7200U 2.50GHz CPU.

4.1 User Interface for Temporal Querying of Faces in Video

We have developed a web application for our experiments. The videos are processed and relevant information from videos is stored in a specific folder structure. Using the bitmap index structures, the application performs queries based on the user's request. The front page of the application is provided in Figure 4.1. The user needs to click on the 'Add Person' button for each person which uploads their full face images. Once the images are provided by the user, the user selects the query type using one of co-appearance, next or eventual buttons as shown in Figure 4.2. Then the user clicks the 'Search' button to initiate the query processing. The images in

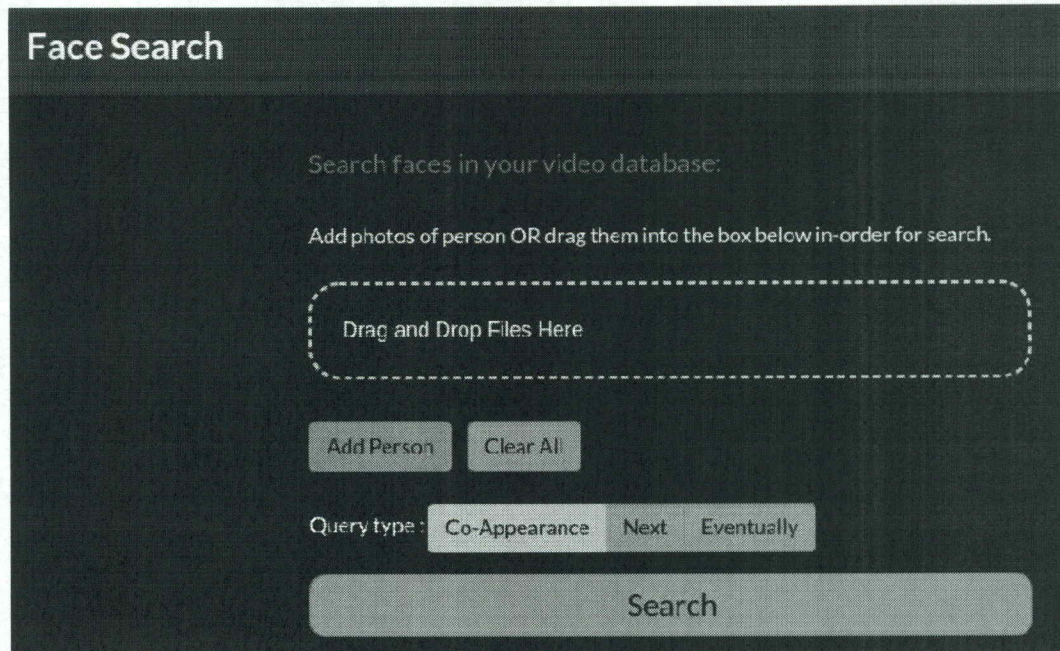


Figure 4.1: Home page of the application

the screenshots are the characters from TV series Big Bang theory. These images are provided as 'fair use' to show how the interface looks like when the images are uploaded.

The system shows the progress of the query using the progress as shown in Figure 4.3. Once the query is processed, the thumbnails of videos that satisfy the query are displayed (Figure 4.4). Whenever the user clicks on a particular video, the time intervals that satisfy the query conditions are highlighted as red on a timeline under the video as displayed in Figure 4.5.

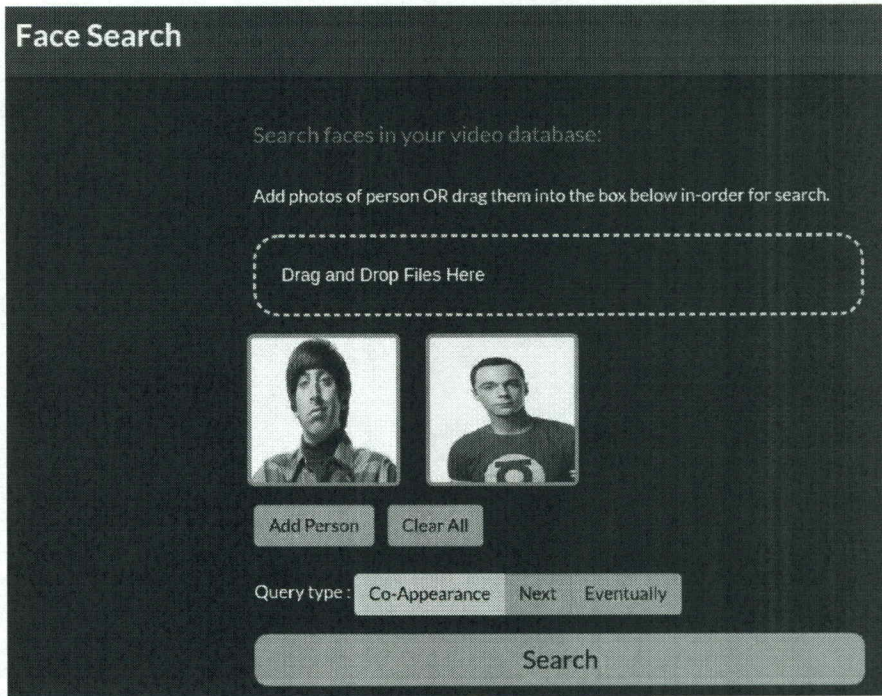


Figure 4.2: Query Type Selection

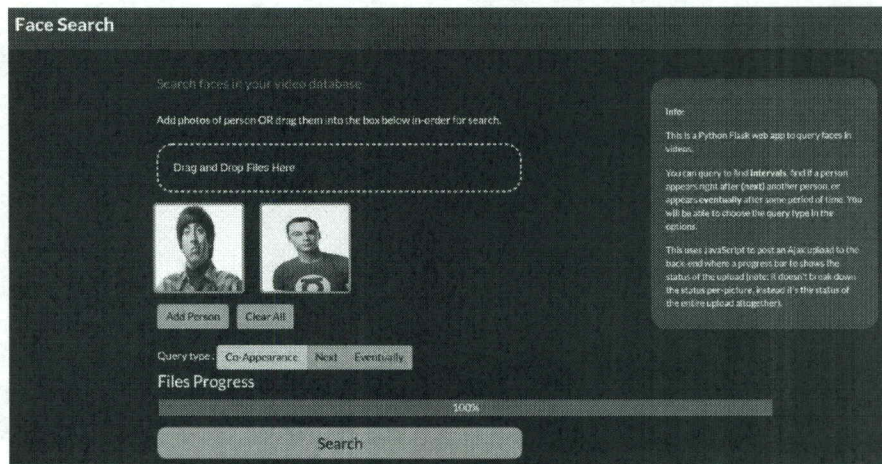


Figure 4.3: Apply Search



Figure 4.4: Filtered list of videos

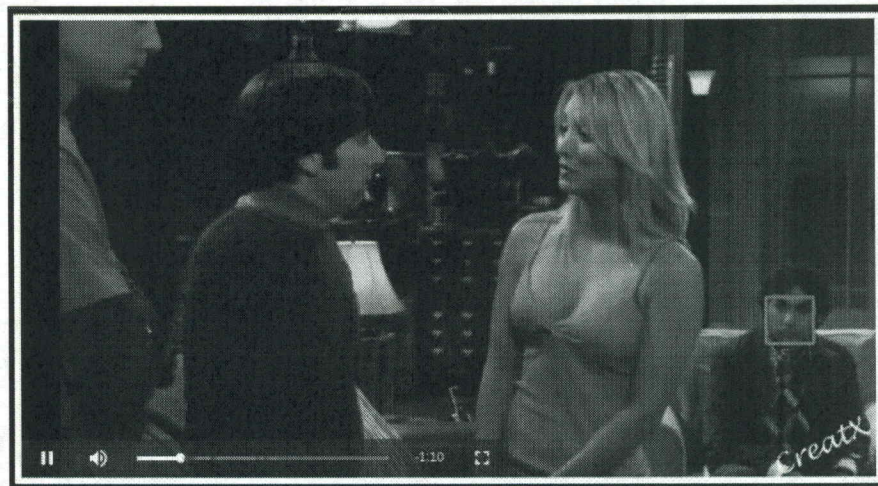


Figure 4.5: Highlighted time segments

4.2 Accuracy

The overall accuracy of the system depends on the proper functioning of face detection and similarity search for different types of videos. We use Dlib library for face detection/tracking and Facebook’s FAISS library for similarity search. Dlib is one of the most popular face recognition libraries and has 99.38% accuracy [48] for the

Labeled Faces in the Wild database (LFW), which is the de facto benchmark dataset for face verification. Also, FAISS has one of the most optimal search implementation and can handle more than a billion data points [34]. Both Dlib and FAISS libraries are state-of-the-art technologies and have been used in many research projects and applications.

While running our tests on different types of videos, we found that the accuracy of Dlib library is heavily impacted by the the quality of video. For videos having less than 360p resolution, detection of faces deteriorated and this affected the 'p2t' mappings for that video. This resulted in many missed intervals for the query. So, in our experiments, we found that if the library is able to detect the faces, then the system works as expected, returning exact intervals for the queries. If it is not able to properly detect faces, then it results in many missed intervals and false recognition of faces.

After face recognition, the similarity search is run to find the record of the person. FAISS returns the top k similar vectors from the database such that the label of the most similar database entry is used to label the input vector. But, for this, we need to set a threshold such that if the similarity value is below a certain threshold [49], we need to treat it as a new label which was never introduced to the database. To find the optimal value of this threshold, we evaluated the face verification performance on a range of distance threshold values. Since the dataset contained much more negative pairs than the positive pairs, we used F1 score as our accuracy metric [50]. The result is shown in Figure 4.6.

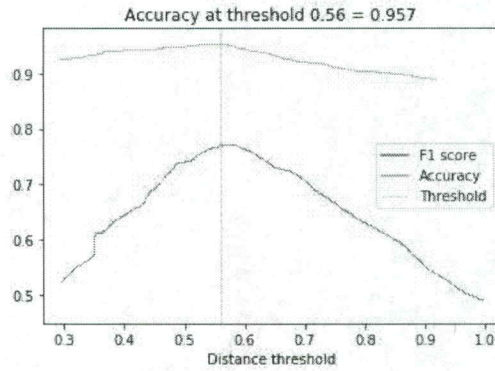


Figure 4.6: Accuracy vs threshold distance

The results showed that the accuracy was maximum at 0.56 threshold value. Using this threshold, the process of finding whether a person already exists in the database can be done with higher level of confidence rather than just using the top nearest result from the similarity search. Also, this procedure can be parallelized and scale to large databases [51]. Once we have the results from face detection and similarity search, we can move ahead to process the requesting query.

4.3 Comparison of Temporal Querying

Our main approach to compute temporal queries is based on using bit manipulation strategies with a notion that bitwise operations would normally be faster than other operators. In our experiments, the average number of time intervals in each video was 20. Using these intervals, we generate their corresponding bitmaps having average length of 20.

Since we could not find any specific method for querying for a database in this format, we came up with an alternate technique to compare performance. Rather than using bitmaps, this alternate technique uses lists. For example, for each person, the list of videos he or she appears is maintained.

While performing queries, there are two stages: i) finding relevant videos and ii) the indices of intervals where they appear together, and returning the actual intervals.

4.3.1 Finding videos

Using $p2v$ mapping, the videos in which a person appears is found. Firstly, a similarity search is run against column p and the most similar vector to the given query image is found. If the similarity score is within a threshold, this indicates that the person is found and then all the videos the person appears are extracted (i.e., bitmap row). FAISS [34] library can find faces in less than 1-second for SIFT1M dataset [52] having of one million data points, by computing 10000 batch searches on a 22.8GHz Intel Xeon E5-2680v2 with 4 Maxwell Titan X GPUs on CUDA 8.0. In our dataset, we have 150 people in 30 videos and to perform the similarity search for a person, it took 3.51ms for each person.

After filtering out the bitmaps for each person ($p2v_i$ and $p2v_k$), we compute $v_{(i,k)}$. The timings involved when calculating this operation with respect to varying number of people is reported in Table 4.1. We then extract $\overline{v_{(i,k)}}$ from $v_{(i,k)}$, which takes $5.93\mu s$. The timings shown in the above table are with respect to bitmaps with length as the number of videos, which is 30 in our case.

No. of people	Total Avg. time (μ s)	
	Bitmap method	Alternate method
2	0.050	0.004
3	0.082	0.004
4	0.116	0.007

Table 4.1: Timings involved to calculate $v_{(i,k)}$

4.3.2 Checking and computing intervals

After finding the videos, each video can be processed and the corresponding intervals can be returned. For queries involving two or more people, there are basically three cases that are encountered.

1. No co-appearance in any video
2. Co-appearance in video, but no common time interval for co-appearance
3. Co-appearance in video and at least one common interval for co-appearance

The average time required to return the results where there is no co-appearance is almost similar in both the approaches. So, we consider only the experiments which has co-appearance.

i) Co-appearance query

Using bitmap method, finding co-appearance in each video involves bitwise ‘AND’ between $p2t_i^j$ and $p2t_k^j$ followed by finding indices of each set bit – which represents the time interval of appearance. The timings for each operation is reported in Table 4.2.

Operation	Avg. time (μ s)
$p2t_i^j$ & $p2t_k^j$ ($t_{(i,k)}^j$)	0.055
Finding indices ($t_{(i,k)}^j$)	0.258
Total	0.313

Table 4.2: Timings involved to compute $t_{(i,k)}^j$ and $\overline{t_{(i,k)}^j}$

In list approach, the time interval of appearance for each person is available and the intersection between those sets of intervals can be computed using a linear time algorithm. The average time taken using this approach was 0.376μ s.

No. of people	Total Avg. time (μ s)	
	Bitmap method	Alternate method
2	0.313	0.376
3	0.360	0.678
4	0.403	1.071
5	0.439	1.343

Table 4.3: Timing comparison between Bitmap and List method to find Co-Appearance between varying number of people

The results from Table 4.2 show that in case of bitmap approach, the time taken to perform ‘AND’ operation is really fast but finding the indices is costlier in comparison. Finding indices depends on the type of implementation and we used the ‘find’ method under the BitString class in Python 3. In future, if a faster version of this function is released, then it will reduce the overall time taken to return the intervals. For now, the cost to compute the queries for two people using bitmap and list approach is almost similar. But the difference is seen when the number of people in the query increases. From Table 4.3, we see that for bitwise method, the overall time increases with small steps whereas in linear

approach, bigger steps in comparison. In bitwise method, all operations prior to finding indices are the bitwise operations, which are usually very fast.

ii) Next-appearance query

The timings involved for computing next-appearance using bitmap approach is reported in Table Table 4.4. Finding next appearance for two disjoint set of intervals is complex than finding co-appearance. A simple and efficient approach would be to check if the end time of p_i comes is the same as the start time of p_j . If it exists, then p_j appears right after p_i . Since, for each interval of p_i , we traverse all the time intervals of p_j , this algorithm's run time complexity is $O(n^2)$ – which is quadratic time. We can also return the timings of the occurrences as well. The average timing involved to perform these calculations is $7.03 \mu s$.

Operation	Avg. time (μs)
AND (η)	0.04153
XOR (β_k)	0.0455
Left shift ($\tilde{\beta}_k$)	0.0440
AND ($\alpha_{i,k}^j$)	0.04153
Comparison ($\alpha_{i,k}^j > 0$)	0.0332
Finding indices of ($\overline{\alpha_{i,k}^j}$)	0.327
Total	0.53276

Table 4.4: Timings involved to compute Next-Appearance between two people using Bitmap approach

In Table Table 4.4, we see a big difference while computing next-appearance. In case of bitmap approach, the overall time required to compute is $0.53276\mu s$ which is very fast, compared to $7.03\mu s$ in the other approach.

iii) Eventual-appearance query

Checking if a person A appears eventually after person B can be done easily. If the start time of last interval for person B is greater than the end time of first interval for person A, then the condition is satisfied, else it is not. For our dataset, using bitmap approach, the operations involved is similar to Next-appearance query with the exception that we do not have to find the indices. So the average time to compute the query, using bitmap approach is $0.20576\mu s$. But using the other approach, it took $0.189\mu s$ which is faster than the bitmap approach, but only by a small margin. Prior-appearance query takes similar time.

4.3.3 Pre-indexing

Clustering faces in videos and using those unique faces to build bitmap index is an automated process. But usually, due to several factors like resolution of video, alignment of the image etc., the 128-dimensional feature vector generated for a person in a video may have a value with a deviation of higher magnitude in another video [53]. This leads the system to treat vectors of same person as different.

In the Figure 4.7, the green line represents the distance compared with a reference image whereas the blue line represents the distance between two images of same person in different videos. Even though the person is the same in the videos, the blue line could cross the threshold and consider these faces as different.

To minimize this effect, we came up with an approach such that, before processing the videos we collect standard high resolution images of the people appearing in those videos and compute their feature vectors. Then we add those feature vectors

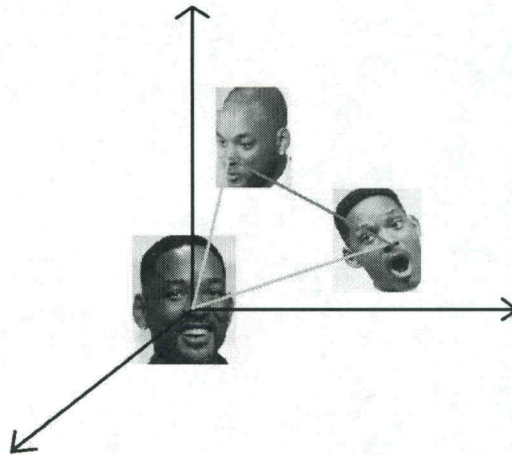


Figure 4.7: Distance computed from a reference image

to the reference table and label the vector. This table acts as a proper reference vector for that person (a proper reference vector is an image descriptor which has resolution larger than 21 X 21 pixels [54] with affine transformation). So when we parse the video and look to update the $p2v$ mapping for each person in the video, we do a similarity search against this reference table to see if there is a vector similar to that person. If it exists, we can update its corresponding video column and if it does not, then it indicates that we do not have a reference vector for that person and hence need to treat the person as a new entity for the system. This approach increased the performance of the system significantly where the people and their interval of appearance in videos was detected with high accuracy. The only problem here would be to collect proper images of the people before processing the video.

4.4 Summary

This chapter analyzes the performance of queries (co-appearance, next and eventual) using the bitmap method and the list method along with asserting the advantages of ‘pre-indexing’. Based on the results, it was found that computing co-appearance and eventual query between two people in videos could be performed using other methods with comparable execution time. However, in case of co-appearance query, if the number of people in the query increased, the bitmap approach yielded the intervals more rapidly. And for next-appearance query, the bitmap approach performed much faster than other approaches.

CHAPTER 5

CONCLUSION

In this thesis, we present an approach for indexing faces in videos which can be used for temporal querying. The indexes are generated using the bitmap index where one can utilize this index to perform queries to confirm if multiple people appear together in a video, determine if a person appears right after another person, or establish if a person appears sometime after the appearance of another person with a simple and generic computational steps. All these queries can be performed using the same indexing scheme employing bit manipulation logic with an assumption that overall cost for computing the queries should be faster than list intersection techniques.

To validate our hypothesis, labeled dataset was generated manually which consisted of 30 videos. Each video was parsed to find the faces and record their corresponding time of appearances. Using this information, the indexing structure was created as explained in Chapter 3. During this index generation process, it was found that face tracking only worked well particularly in high resolution videos (typically greater than 360p) otherwise yielding in many misses. Also, after face tracking, a threshold distance hyper-parameter was introduced that should be satisfied in order to verify the person with our database with higher confidence.

Once the index was created, queries were performed and their timings were tracked for each operation. The results from the experiment verified that the bitmap indexing scheme can be used to perform temporal querying of faces with a generalized approach with respect to the number of query elements and type of query while maintaining a very high level of efficiency.

5.1 Future Works

There are few strategies that we could adopt in order to increase the performance of our system. One of them is by reducing the time taken to parse the video for tracking faces [55]. A robust and fast video parser with index generator will definitely make the system much more user friendly.

Another future work will involve implementing improved version of face tracking that especially supports low resolution videos. We also plan on adding spatial context to our queries, which will help to create more advanced search capabilities from the content. Additionally, we plan to apply this approach for temporal CCTV videos analytics [56]. By exploring indexed faces in CCTV videos that focus on capturing faces (e.g., for ATM confirmation), we will investigate how effectively the behaviors of suspicious people can be identified and confirmed over time. In addition, the same structure of indices allows the user to build additional components to the query like co-appearance, next-appearance, prior-appearance and eventual-appearance.

We also plan on parallelizing our computational steps and testing it in a big data environment [57]. In a practical scenario, a multimedia database could have a lot of videos that could extend to millions of hours of recordings. Performing search

operation in such database will be a challenging task and one of the possible solution would be to parallelize the bit manipulation steps in a larger cluster.

CHAPTER 6

REFERENCES

- [1] Anders Brodersen, Salvatore Scellato, and Mirjam Wattenhofer. Youtube around the world: geographic popularity of videos. In *Proceedings of the 21st international conference on World Wide Web*, pages 241–250. ACM, 2012.
- [2] Muhammad Nabeel Asghar, Fiaz Hussain, and Rob Manton. Video indexing: a survey. *Int. Jour. of Computer and Information Technology*, 3(01), 2014.
- [3] Yaniv Taigman, Ming Yang, Marc’Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proc. of the IEEE Conf. on computer vision and pattern recognition*, pages 1701–1708, 2014.
- [4] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proc. of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.
- [5] Davis E King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10(Jul):1755–1758, 2009.
- [6] Walid Aref, Moustafa Hammad, Ann Christine Catlin, Ihab Ilyas, Thanaa Ghanem, Ahmed Elmagarmid, and Mirette Marzouk. Video query processing in the vdbms testbed for video database research. In *Proceedings of the 1st ACM international workshop on Multimedia databases*, pages 25–32. ACM, 2003.
- [7] R. Aygun and W. Benesova. Multimedia retrieval that works. In *2018 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, pages 63–68, April 2018.
- [8] Z. Chen, Y. Wen, J. Cao, W. Zheng, J. Chang, Yinjun Wu, G. Ma, M. Hakmaoui, and G. Peng. A survey of bitmap index compression algorithms for big data. *Tsinghua Science and Technology*, 20(1):100–115, Feb 2015.
- [9] M. Naik, V. Jain, and R. S. Aygun. S3g: A semantic sequence state graph for indexing spatio-temporal data - a tennis video database application. In *2008 IEEE International Conference on Semantic Computing*, pages 66–73, Aug 2008.

- [10] M. M. Naik, M. Sigdel, and R. S. Aygun. Spatio-temporal querying recurrent multimedia databases using a semantic sequence state graph. *Multimedia Systems*, 18(3):263–281, Jun 2012.
- [11] Kurt Stockinger and Kesheng Wu. Bitmap indices for data warehouses. In *Data Warehouses and OLAP: Concepts, Architectures and Solutions*, pages 157–178. IGI Global, 2007.
- [12] Chee-Yong Chan and Yannis E Ioannidis. Bitmap index design and evaluation. In *ACM SIGMOD Record*, volume 27, pages 355–366. ACM, 1998.
- [13] Elizabeth O’Neil, Patrick O’Neil, and Kesheng Wu. Bitmap index design choices and their performance implications. In *11th International Database Engineering and Applications Symposium (IDEAS 2007)*, pages 72–84. IEEE, 2007.
- [14] Ognjen Arandjelovic and Andrew Zisserman. Automatic face recognition for film character retrieval in feature-length films. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 860–867. IEEE, 2005.
- [15] Josef Sivic, Mark Everingham, and Andrew Zisserman. Person spotting: video shot retrieval for face sets. In *International conference on image and video retrieval*, pages 226–236. Springer, 2005.
- [16] Caifeng Shan. Face recognition and retrieval in video. In *Video Search and Mining*, pages 235–260. Springer, 2010.
- [17] Yan Li, Ruiping Wang, Zhiwu Huang, Shiguang Shan, and Xilin Chen. Face video retrieval with image query via hashing across euclidean space and riemannian manifold. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4758–4767, 2015.
- [18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [19] Mei Wang and Weihong Deng. Deep face recognition: a survey. *arXiv preprint arXiv:1804.06655*, 2018.
- [20] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [21] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

- [22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [23] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.
- [24] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu. Cosface: Large margin cosine loss for deep face recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5265–5274, 2018.
- [25] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. *arXiv preprint arXiv:1801.07698*, 2018.
- [26] Rajeev Ranjan, Vishal M Patel, and Rama Chellappa. Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(1):121–135, 2019.
- [27] Jialong Yang, Peiran Ren, Dongqing Zhang, Dong Chen, Fang Wen, Hongdong Li, and Gang Hua. Neural aggregation network for video face recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4362–4371, 2017.
- [28] Brandon Amos, Bartosz Ludwiczuk, Mahadev Satyanarayanan, et al. Openface: A general-purpose face recognition library with mobile applications. *CMU School of Computer Science*, 2016.
- [29] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.
- [30] Vahid Kazemi and Josephine Sullivan. One millisecond face alignment with an ensemble of regression trees. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1867–1874, 2014.
- [31] Matthias Jarke and Jurgen Koch. Query optimization in database systems. *ACM Computing surveys (CSUR)*, 16(2):111–152, 1984.
- [32] Yoshiharu Ishikawa, Ravishankar Subramanya, and Christos Faloutsos. Mindreader: Querying databases through multiple examples. 1998.
- [33] Rakesh Agrawal, Christos Faloutsos, and Arun Swami. Efficient similarity search in sequence databases. In *International conference on foundations of data organization and algorithms*, pages 69–84. Springer, 1993.

- [34] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*, 2017.
- [35] V. Jain and R. Aygun. Smart: A grammar - based semantic video modeling and representation. In *IEEE SoutheastCon 2008*, pages 247–251, April 2008.
- [36] Vani Jain and Ramazan Savas Aygün. Spatio-temporal querying of video content using sql for quantizable video databases. *Journal of Multimedia*, 4:215–227, 2009.
- [37] Moshe Y. Vardi. Branching vs. linear time: Final showdown. In *Proceedings of the 7th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS 2001, pages 1–22, London, UK, UK, 2001. Springer-Verlag.
- [38] Vineetha Bettaiah and Ramazan S Aygun. Query-by-gaming: Interactive spatio-temporal querying and retrieval using gaming controller. *Journal of Visual Languages & Computing*, 29:63–76, 2015.
- [39] Ramazan S. Aygun and Vineetha Bettaiah. *Query-by-Gaming*, pages 1–10. Springer International Publishing, Cham, 2017.
- [40] Minyoung Kim, Sanjiv Kumar, Vladimir Pavlovic, and Henry Rowley. Face tracking and recognition with visual constraints in real-world videos. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2008.
- [41] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, et al. Deep face recognition. In *bmvc*, volume 1, page 6, 2015.
- [42] Hong-Wei Ng and Stefan Winkler. A data-driven approach to cleaning large face datasets. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 343–347. IEEE, 2014.
- [43] Xinbo Gao, Jie Li, and Yang Shi. A video shot boundary detection algorithm based on feature tracking. In *International Conference on Rough Sets and Knowledge Technology*, pages 651–658. Springer, 2006.
- [44] Hsueh-Hsien Chang, Hong-Tzer Yang, and Ching-Lung Lin. Computer supported cooperative work in design iv. *Chap. Load Identification in Neural Networks for a Non-intrusive Monitoring of Industrial Electrical Loads*, pages 664–674, 2007.
- [45] Min-Ho Park, Rae-Hong Park, and Sang Wook Lee. Efficient shot boundary detection for action movies using blockwise motion-based features. In *Int. Symposium on Visual Computing*, pages 478–485. Springer, 2005.
- [46] Chris Biemann. Chinese whispers: an efficient graph clustering algorithm and its application to natural language processing problems. In *Proc. of the first workshop on graph based methods for natural language processing*, pages 73–80. Assoc. for Computational Linguistics, 2006.

- [47] James F Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.
- [48] Xiang Xu, Ha A Le, Pengfei Dou, Yuhang Wu, and Ioannis A Kakadiaris. Evaluation of a 3d-aided pose invariant 2d face recognition system. In *2017 IEEE International Joint Conference on Biometrics (IJCB)*, pages 446–455. IEEE, 2017.
- [49] Qin Lv, Moses Charikar, and Kai Li. Image similarity search with compact data structures. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 208–217. ACM, 2004.
- [50] László A Jeni, Jeffrey F Cohn, and Fernando De La Torre. Facing imbalanced data—recommendations for the use of performance metrics. In *2013 Humaine Association Conference on Affective Computing and Intelligent Interaction*, pages 245–251. IEEE, 2013.
- [51] Narayanan Sundaram, Aizana Turmukhametova, Nadathur Satish, Todd Mostak, Piotr Indyk, Samuel Madden, and Pradeep Dubey. Streaming similarity search over one billion tweets using parallel locality-sensitive hashing. *Proceedings of the VLDB Endowment*, 6(14):1930–1941, 2013.
- [52] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2011.
- [53] Wilman WW Zou and Pong C Yuen. Very low resolution face recognition problem. *IEEE Transactions on image processing*, 21(1):327–340, 2012.
- [54] Tomasz Marciniak, Agata Chmielewska, Radoslaw Weychan, Marianna Parzych, and Adam Dabrowski. Influence of low resolution of images on reliability of face detection and recognition. *Multimedia Tools and Applications*, 74(12):4329–4349, 2015.
- [55] Si Liu, Changhu Wang, Ruihe Qian, Han Yu, Renda Bao, and Yao Sun. Surveillance video parsing with single frame supervision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 413–421, 2017.
- [56] Matthew PJ Ashby. The value of cctv surveillance cameras as an investigative tool: An empirical analysis. *European Journal on Criminal Policy and Research*, 23(3):441–459, 2017.
- [57] Wei-Chou Chen, Shian-Shyong Tseng, Lu-Ping Chang, Tzung-Pei Hong, and Mon-Fong Jiang. A parallelized indexing method for large-scale case-based reasoning. *Expert Systems with Applications*, 23(2):95–102, 2002.