Image-based Video Search Engine

Data Compression and Nearest Neighbour Search

EE3L11: Bachelor Graduation Project

Lars Hoogland and Matthijs Korevaar

Technische Universiteit Delft

TU Delft

Image-based Video Search Engine

Data Compression and Nearest Neighbour Search

by

L. Hoogland and M. Korevaar

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended on Monday June 20, 2022 at 13:30 PM.

Student number: 4490363, 5162467
Project duration: April 19, 2022 – June 24, 2022
Thesis committee:

Dr. ir. J. Dauwels,	TU Delft, supervisor
Prof. Dr. ir. O. Isabella,	TU Delft, jury chair
ing. R. van Puffelen,	TU Delft

TU Delft

Abstract

One of the main problems with Instance-level Image Retrieval in video data is that for longer query videos or large amount of image queries, comparing all of the query images to every extracted frame is time-inefficient. This thesis aims to solve this problem by implementing Nearest Neighbour Search (NNS) algorithms and data compression methods, significantly reducing total comparison time. In most NNS use cases, the reference data is provided before reaching the user, allowing methods such as ANNOY or HNSW to partition the data beforehand. However, little research has been done into partitioning the data during run-time. In this thesis, the use of Nearest Neighbor Search and Data Compression methods are discussed for the purposes of matching a query image to a query video, both of which are provided at run-time. The result is an implementation of several state-of-the-art NNS and data compression methods in a system which, based on the amount of query images and the amount of extracted keyframes, selects the optimal comparison method to be used, as well as its optimal parameters if applicable.

Preface

This thesis was written as part of the Bachelor Graduation Project (EE3L11) 2021-2022 to conclude the Bachelor Electrical Engineering at the Delft University of Technology. The project took a total of 10 weeks over the period of April to June 2022. The project was proposed by supervisor Justin Dauwels. Originally, the project was named ”Image Search Engine for Digital History”, but it was subsequently changed to its current state in week 4.2. The name of the project was then altered to ”Image-Based Video Search Engine”. We would like to thank our supervisor Justin Dauwels, daily supervisor Yuanyuan Yao and the Bachelor Graduation Project coordinator Ioan Lager for their support during the project. Finally, we would like to thank the members of the other subgroups [1], [2] of this project. Working together has been both an enjoyable and educational experience.

Lars Hoogland and Matthijs Korevaar
Delft, April 2022

Contents

1

Introduction

Image-Based Video Search Engine

This thesis is part of an ongoing research project “Engineering Historical Memory” [3]. This thesis builds upon the work of one of the Bachelor Graduation Project [4] groups of the TU Delft from academic year 2021-2022 on a Search Engine for Digital History [5], [6]. The goal of their research was to create a search engine that can detect whether an object appears in a database of images. Since then, the CAS group of the TU Delft has worked on improving the image search engine with a team of MSc students [7]. The next step in this project is to create a similar search engine that can detect instances of a desired object in a query video. This Image-Based Video Search Engine is to be designed for a variety of use cases, transcending the historical use case.

State-of-the-art Analysis

Instance-level image retrieval (IIR) is the problem of detecting an instance of an object that appears in an image and then retrieving images from a database that contain the same instance of this object. IIR and its application to video footage are both active fields of research. As video data and surveillance coverage are becoming ever more prevalent, developing information systems that can process and query this data is becoming increasingly important [8], as it is unfeasible to comb through all this footage by hand. Thus, there are endless applications for video-based IIR: such as person/vehicle identification, assistance in copyright claims, querying historical footage, etc.

Several examples exist of using IIR systems for finding images in a database of images [8], and for finding videos in a database of videos [9]. However, relatively little was found in using IIR systems for detecting images in a video. This suggests potential for research into the field. One of the rare implementations of a video-based IIR system is the work of A. Araujo *et al.* [10]. Their research focuses on reducing storage requirements when using IIR systems for video databases when using local feature matching by determining optimal descriptors. A similar existing system is Video Google [11]. The approach used by Video Google is to perform the search similar to how Google does text document retrieval. Frames of a video are evaluated based on two types of regions. One based on gradients and a second one based on how stationary objects are. Vector descriptors can be made based on these regions and evaluated using text retrieval methods. In this case an inverted file structure is used that stores the descriptors as visual words. Similar to how commonly occurring words (such as ’the’) are excluded in a text based scenario, the commonly occurring descriptors are put in a ’stop list’ that suppresses these occurrences. However, the reported system takes frames from the video as inputs. Both of these methods make use of local features for comparing the query images to the video dataset. Using local features leads to accurate results but in both papers time considerations are ignored.

Looking into existing video-based IIR systems for historic systems, only one implementation was found in the work of Condorelli *et al.* [12]. Their work focuses on detecting segments of video which contain lost cultural heritage in historical video footage. This method focuses on the accuracy of the resulting

video segments and the length of these segments as compared to the length of the original footage. However, similar to the existing systems explained above, the research does not take computing time into consideration.

From this State-of-the-art Analysis it can be concluded that research into video-based IIR systems has been done, but most research focuses on optimising accuracy of the system and barely any research focuses on the execution time of the system. Thus, there is a lot of potential for research into the field.

Problem scoping and bounding

Utilising IIR for finding images in a database of images is in itself already valuable. However, with the rapidly growing amount of visual content, not only the amount of images is increasing, but the amount of video material as well. Extending the IIR to videos opens up new possibilities to explore video material. From easier access by searching for images in a library of cultural heritage videos [9] to finding appearances of a companies products in extremist videos.

The main objective is to develop a system that is capable of retrieving occurrences of one or more desired objects in a set of given videos, based on a set of given images. The second objective is to document the process of developing this system. The first objective will be completed if it complies with the requirements as specified in Chapter 2. The second objective will be completed if it complies with the requirements as described in the BAP manual [4].

The main limitation of the project is the time constraint of 10 weeks. This is the time allocated to build the entire system and document the process. The system itself is limited to use content based methods only. This ensures that the content of the image is taken into account and no interpretations are made of the image. Text-based methods have the limitation of the terms that describe the image [13], which also limits search across cultures when these text-terms are not supported. Further constraints are placed on which methods to be used. Existing methods should be used to develop the search engine, in order to ensure the allocated time is put to good use on developing the system, rather than on improving existing methods. Lastly, the development is restricted to using Python [14] and its associated libraries and tools.

To create alternative systems to the Image-Based Video Search Engine, one could look into implementing different methods for each of the modules, specifically regarding Feature Extraction . This module is the slowest of the system, followed by the Keyframe extraction module. By looking into different methods for these modules, the speed of the system could be improved. Furthermore, the Feature Extraction module could be trained on different data in order to improve the performance of the system even further. Finally, further tuning of the module-specific parameters could also be done to improve the performance. The system-wide performance can be gauged by the Key Performance Indicators: mean average precision (mAP), recall, and the amount of time saved.

Problem statement

Based on the analysis, the problem scoping and the problem bounding, the following problem statement was derived:

Develop a system that can detect whether an instance of a desired object appears in a given video, based on a given set of images containing the desired object.

System Overview

The system

The system takes one or more query video(s) and one or more query image(s). These are then compared to each other and any timestamps where the image(s) appear in the video(s) are returned. Figure 1.1 shows this process.

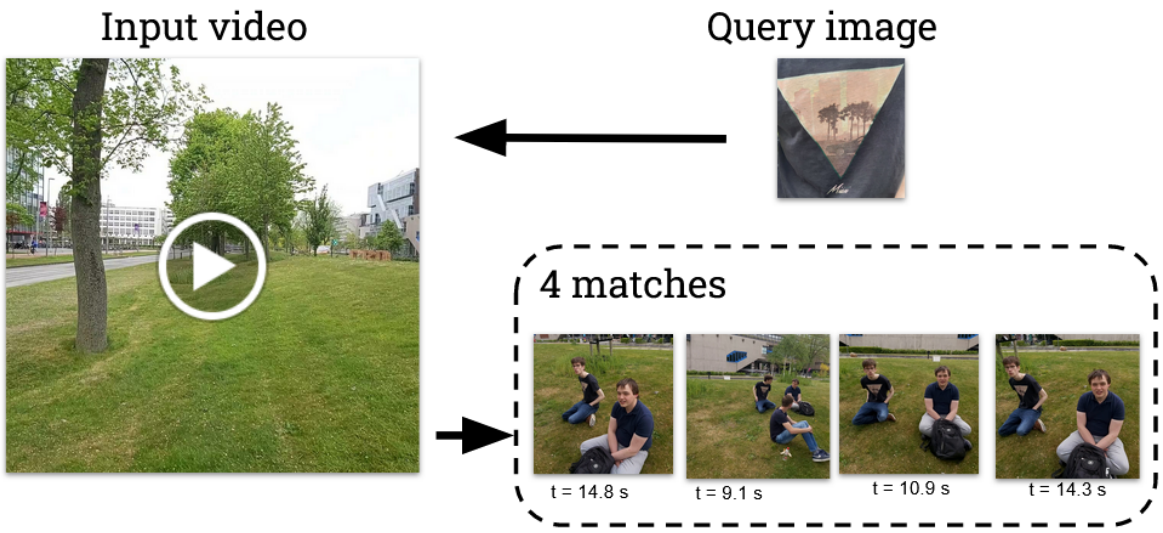


Figure 1.1: The system prototype

Subdivision of the System

According to the BAP thesis guidelines [4], each BAP group has to be split into three subgroups of two people each. As such, this Image-Based Video Search Engine has to be split into three modules. These modules were selected to be:

1. Key-Frame Extraction (KFE) [1], which cuts the query video down into frames and removes unnecessary frames.
2. Feature Extraction (FE) [2], which translates the keyframes and the query image into feature vectors.
3. Data Compression & Nearest Neighbour Search (DCNNS), which compares the extracted features of the keyframes to those of the query image.

The three modules will tackle the following subproblems respectively:

1. *Develop an algorithm that finds keyframes to reduce the amount of video frames to be evaluated.*
2. *Develop an algorithm that can extract the features of the keyframes and of the query images.*
3. *Develop an algorithm that can compare the features of the query image(s) to the features of the keyframes.*

This thesis will describe the Data Compression and Nearest Neighbour submodule.

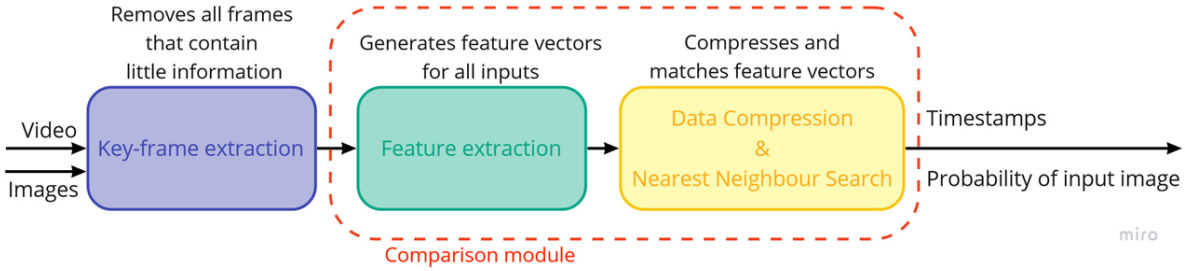


Figure 1.2: Pipeline of the complete Image-Based Video Search Engine

Nearest Neighbour Search

The third module is the Data Compression and Nearest Neighbour submodule. It focuses on subproblem 3 of the problem statement. In essence, the module consists of an algorithm that compares

the features of the keyframes and of the image queries that were extracted by the Feature Extraction module. The goal is to optimize this comparison by evaluating different methods and their strengths and weaknesses under varying conditions. These conditions include the amount of keyframes and the amount of query images.

Document Structure

This thesis describes the Data Compression and Nearest Neighbour submodule. In Chapter 2 the programme of requirements is explained. In Chapter 3, existing methods are analysed. In Chapter 4 the design of the module is explained. In Chapter 5 the implementation and validation of the prototype are explained. In Chapter 6 the results are discussed and the conclusion is presented.

2

Programme of requirements

The Programme of requirements lists the restrictions and functionality of the Image-Based Video Search Engine. The requirements are divided into mandatory requirements and trade-off requirements. The mandatory requirements must be met and specify the core of the system. The trade-off requirements lists requirements that improve the system and customer satisfaction when they are met, but they are not absolutely necessary for the system to function. Both sections are divided into functional requirements and non-functional requirements. The complete overview is shown below.

Mandatory Requirements

Functional Requirements

1. The search engine must detect whether an instance of a desired object appears in a given video, based on a given set of images containing the desired object.
2. The search engine must return the timestamp(s) where the object appears in the video.
3. Matching of images to frames must be based on visual content.

Non-Functional Requirements

4. The system must support video files of the type mp4.
5. The system must support image files of the type png.
6. The full implementation must be completed within 10 weeks by a group of 6 students.
7. The system must be written in Python version 3.9 or higher.
8. Conda version 4.10 or higher must be used for synchronising Python environments.
9. The engine must be able to be tested with hardware that is available to the group.
10. For a single query image, the engine should process a video shot at 30 frames per second in half the video duration.
11. Mean Average Precision must be at least 65%.

Trade-off Requirements

12. The engine should be able to handle multiple input videos.
13. The engine should be able to handle multiple input images.
14. Mean Average Precision should be as high as possible.
15. The codebase should be structured clearly and properly documented.
16. The supported number of image formats should be as high as possible.
17. The supported number of video formats should be as high as possible.

18. The execution time should be as low as possible .
19. The system should be able to process videos with a large duration.

The core functional requirements 1, 2 and 3 are the result of discussions with the project supervisor. Requirements 4 and 5 refer to the minimum file support. Requirement 6 follows from the BAP manual [4]. Requirements 7, 8 and 9 serve to limit the scope of the project. Requirements 10, 11 are important for measuring performance. The trade-off requirements specify requirements that lead to increasing customer satisfaction. Requirements 12 ,13, 16, 17, 19 refer to the scalability of the system. Requirements 14 and 18 again are important for measuring performance. Requirement 15 is important for future research.

Programme of Requirements for Data Compression & Nearest Neighbour Search

Aside from the requirements for the entire system, the Nearest Neighbour Search module has a set of requirements of its own. The requirements are not relevant for the entire system since they specify constraints on the implementation of this module. These requirements are listed below:

Mandatory Requirements

Functional Requirements

1. The extracted features of a query image must be compared to the extracted features of the keyframes.
2. The system should be optimised for features of length 2048.
3. From this comparison, an amount of nearest neighbours k must be returned.
4. This amount of nearest neighbours k should depend on the amount of query keyframes.
5. If data partitioning is required for a comparison method, its partitioning time must be taken into account for its performance as the data has to be partitioned during run-time.
6. Recall should be at least 50%.

Non-Functional Requirements

7. Existing implementations of nearest neighbor search methods must be used, rather than developing new methods.
8. The used nearest neighbor search method should be at least as fast as linear search in python.

Trade-off Requirements

9. The total comparison time for the NNS should be as low as possible.

The core functional module requirements 1, 3, 5 and 7 follow from discussions with the project supervisor. Module requirement 2 was determined based on discussions with the Feature Extraction subgroup [2]. Module requirement 4 is based on the wide amount of use cases the system will be used for. For longer videos, more found neighbours should be returned. Module requirements 6, 8 and 9 relate to the performance of the system.

3

Analysis of existing methods

To determine whether an object of interest actually appears in a video, a comparison needs to be made between the extracted feature vectors of the keyframes and those of the query images, as explained in Chapter 1. Defining the reference data (feature vectors of the keyframes) as a set of points **P** $\{x_1, x_2, ..., x_n\}$, the goal of this module is to find the k points $\{x_1, x_2, ..., x_k\}$ in this set that have the smallest distance to a query point **q** (the feature vector of the query image) [15]. The simplest and most precise way to do this, is to compare the feature vectors using a brute-force method: comparing every query vector to every keyframe vector. The main downside of this method is its speed: the comparison time grows linearly with the amount of queries and the amounts of keyframes. In this chapter two separate methodologies for reducing the comparison time will be analysed: Nearest Neighbour Search and Data compression.

3.1. Nearest Neigbour Search

Finding the closest matches between the query image(s) and keyframes can be done by evaluating which feature vectors of the keyframes have the smallest distance to the feature vector(s) of the query image(s). This process is called the Nearest Neighbour Search (NNS). Evaluating the distance between the feature vectors allows for finding the best matches by taking the matches with the best distance. There are multiple distance metrics that can be used and the what defines the best distance varies as well. Other NNS algorithms have been developed that can optimise the total comparison time by first partitioning the data points **P** $\{x_1, x_2, ..., x_n\}$ in such a way that they can be efficiently compared to the query point **q**. The main advantage of partitioning the data is that the search time is significantly lower than that of linear comparison. However, the partitioning costs extra time during run-time. Several implementations of NNS methods will be analysed. Rather than creating a new NNS method from scratch, the project supervisor suggested making use of existing NNS methods. The investigated methods are brute-force linear comparison [7], Approximate Nearest Neighbour Search Oh Yeah (ANNOY) [16], [17], Hierarchical Navigable Small World (HNSW) [18], [19] and several Facebook AI Similarity Search (FAISS) implementations [20].

3.1.1. Distance Metric

In order to determine the similarity between the feature vectors of the keyframes and those of the query image, a metric for determining the distance is required. The employed methods offer limited support for distance metrics. To be able to evaluate all metrics equally the same distance metric should be used for every method. The methods supported between all implementations are the euclidean distance, cosine similarity and the inner product [17], [19], [20]. These are defined by the following formulas [19], [21]:

$$Euclidean : ED(x, y) = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2} \tag{3.1}$$

$$Inner\ product : IP(x,y) = 1 - \sum_{i=1}^{n} x_i y_i \tag{3.2}$$

$$Cosine : CosD(x,y) = 1 - \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} x_i y_i^2}} \tag{3.3}$$

According to [22], a measure has to obey four properties to be considered a metric. Out of these three methods, only the euclidean distance obeys these properties: The distance is non-negative ($d(x,y) \geq 0$), indiscernible ($d(x,x) = 0$), symmetric ($d(x,y) = d(y,x)$) and adheres to the triangle inequality ($d(x,y) \leq d(x,z) + d(z,y)$). For the cosine similarity and inner product similarity, the indiscernibility property does not hold. Thus the distance from a point to itself is larger than zero, which makes it unreliable for queries that are exactly equal to one of the keyframes. This leaves the euclidean distance as the best metric to be used. Furthermore, information is given by [23] on the computation time for the euclidean and cosine similarity. The euclidean distance strongly outperforms the cosine similarity in computation time, while the accuracy is very similar. This could have a higher impact for large videos that contain many keyframes, since the need would arise to partition many feature vectors and thus many comparisons have to be made.

When using the Euclidean distance as a distance metric, it is important to first normalise the extracted features before comparing feature vectors. The reason for this is to ensure every feature is equally important for finding the nearest neighbour of a query point. Without normalisation, features that have a small variance will be weighted more than features that have larger variances. As neither the triplet network feature extraction described above, nor the methods used by the feature extraction group [2] produce normalized results, all results are normalized before comparison.

3.1.2. Linear comparison

Linear or Brute-force comparison is the simplest comparison method. Linear comparison compares the feature vector of every query to those of all the keyframes. As the name implies, the complexity grows linearly with the size of the data. This can be quantified as $O(n)$. The main benefit of using linear search is that it does not require data partitioning. As such, the total comparison time only depends on the search time.

3.1.3. ANNOY

ANNOY is a tree-based Nearest Neighbour Search method [16], [17]. It splits an n-dimensional space such that small regions are defined that hold a certain number of vectors. The length of the feature vector defines the n dimensions of the space. The splitting is done by finding the hyperplane between two randomly chosen points of the data to be partitioned. This process continues recursively until a specified amount of vectors is left in each node [16]. An example of this partitioning in a two-dimensional space can be found in Figure 3.1. Since the data is split in two segments every step, it allows for the construction of a binary tree. The tree corresponding to Figure 3.1 can be found in Figure 3.2. By constructing multiple trees and searching them at the same time a region can be determined of which the approximate nearest neighbours can be retrieved. The amount of trees is described by the parameter $forestsize$. A search in a binary tree reduces the complexity to $O(log(n))$. The complexity for ANNOY is lower than that for linear, which results in a faster search speed. However, there is additional cost for partitioning the data, as the tree can only be constructed once all the feature vectors have been extracted. The partitioning time is a one-time cost. After the data is partitioned, the search itself can be performed faster than linear search.

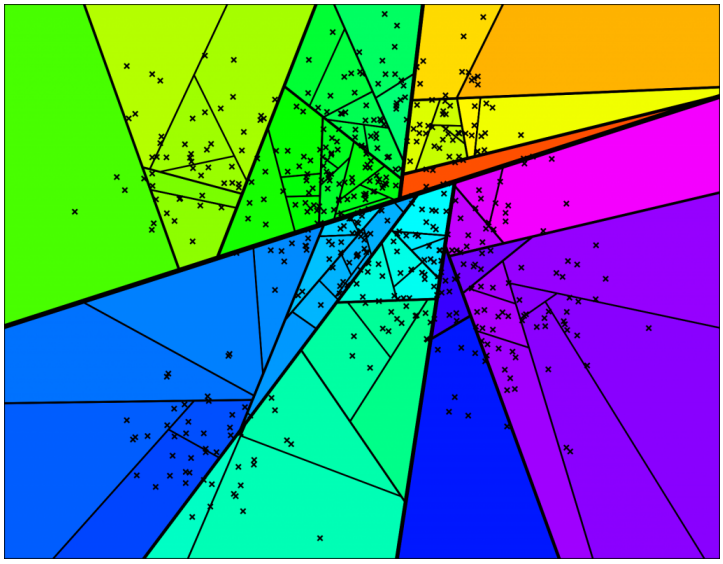


Figure 3.1: Data partitioning for ANNOY represented in 2D Space. Every cross represents a data point and every line represents a split in the constructed binary tree. Adapted from [16].

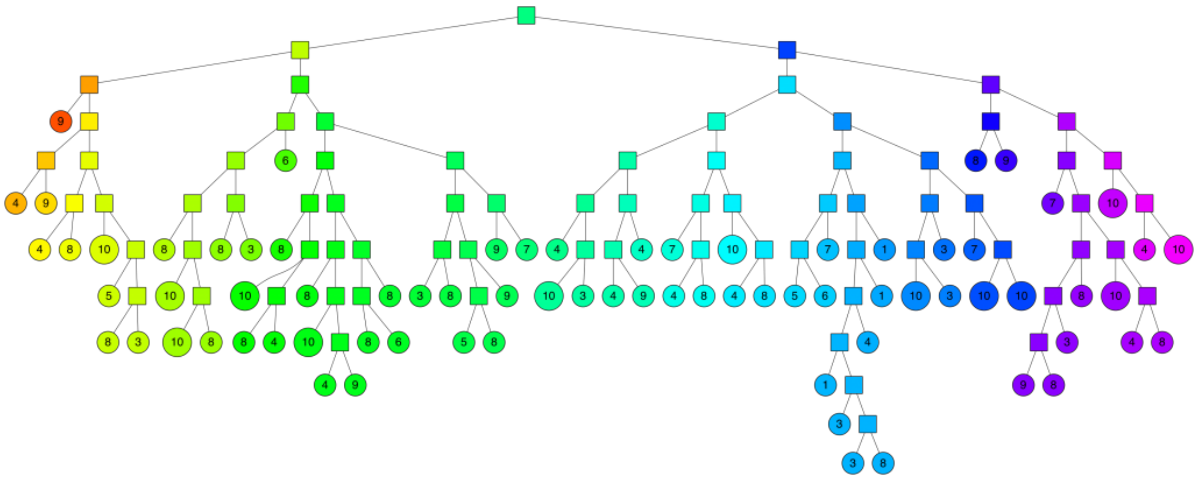


Figure 3.2: Corresponding binary tree for ANNOY. Adapted from [16].

3.1.4. HNSW

Another NNS method is Hierarchical Navigable Small World (HNSW) [18], [19]. HNSW is a graph-based method that is based on the Navigable Small World (NSW) algorithm. It partitions the data by finding the approximate nearest neighbours of a data point to be inserted and appending the data point to those nodes. The set of nearest neighbours to be found when partitioning the data this way can be expressed by a parameter ef_{const}. A similar parameter ef is used to define the amount of nearest neighbours to be found during the search. Furthermore, each node can have a number of connections up to a specified value M. The difference from the NSW algorithm comes due to constructing different layers to speed up the search. The structure is shown in Figure 3.3. Finding an item is done by first evaluating the links between larger clusters before going into them to find the vectors with the least distance. Figure 3.3 shows how this search is performed by having the red point as the entry, the green point the query and the read arrows defining the path that is taken. The time complexity for the HNSW is $O(log(n))$, which puts it on equal footing to ANNOY. The graph is constructed by adding vectors one after the other. An opportunity that arises from this is the ability to construct the graph dynamically. Using batches of feature vectors the graph can be extended parallel to the Feature Extraction module. The limiting condition for this approach to be be viable is that it should not reduce the processing speed of the Feature Extraction. By applying this method the relevant partitioning time would consist only of

the last batch of feature vectors to be added.

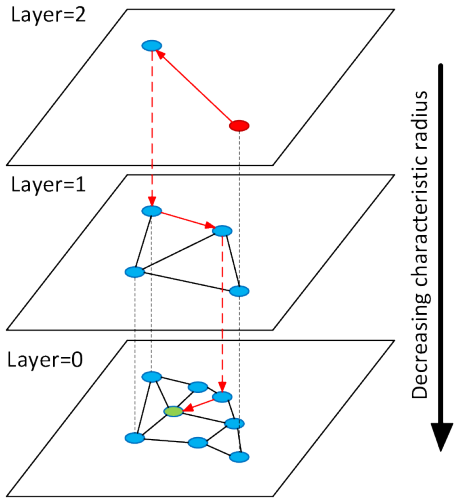


Figure 3.3: Representation of data partitioning for HNSW. Adapted from [18]

3.1.5. FAISS

Facebook AI Similarity Search (FAISS) [20], [24] was originally built to explore the use of the Graphical Processing Unit (GPU) for nearest neighbour search. Since then, the library has been extended and it now consists of a variety of methods for similarity search. It includes implementations for HNSW [18] and Inverted file (IVF) as used in the Video Google system [11]. The idea of IVF is to build a list containing all occurrences of a visual word [11]. The origin of IVF is in finding occurrences of a word in text files. The term visual word tries to put the feature vector in context of text files. Thus its meaning is the same as a feature vector. The IVF implementation in the FAISS library uses Voronoi cells to partition the data. The result of the data partitioning is shown in Figure 3.4. For IVF there are two parameters that can be tuned. The first one is the number of splits, which indicates how many Voronoi cells should be made. The second one is $nprobe$ which represents the number of Voronoi cells to be evaluated during a search.

Furthermore, FAISS allows for exploring data compression methods such as Product Quantization (PQ) [25] and Locality Sensitive Hashing (LSH) [26]. These methods will be described in Section 3.2.

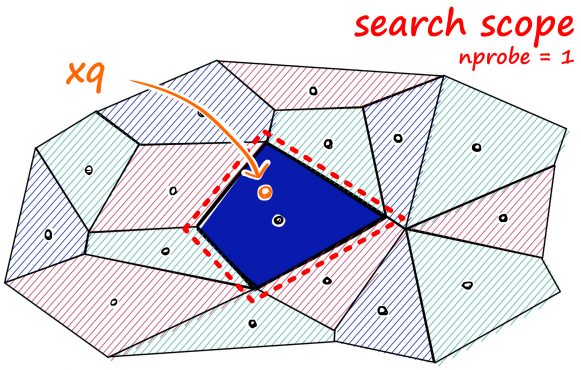


Figure 3.4: Data partitioning for IVF in 2D space using Voronoi cells. Adapted from [27]

3.2. Data compression

Another method of reducing the comparison time is by applying hashing. Hashing is a technique where the reference data $\{x_1, x_2, ...x_n\}$ and query point **q** are mapped to lower dimensional data (integers) $\{y_1, y_2, ..., y_n\}$ by means of a hashing function $y = f(x)$. This is done in such a way that similarity between any two items $\{x_i, x_j\}$ in the reference data is preserved such that their respective hashes $\{y_i, y_j\}$ are also similar. [28]. For the purposes of this video search engine, hashing would reduce the feature vectors of the extracted frames and those of the query images to lower dimensional data. After this compression, the lower dimensional data can be compared much more quickly, as compared to the high-dimensional feature vectors. Hashing has similar advantages and disadvantages to nearest neighbor search, as explained in Section 3.1: The search time is significantly lower than that of linear comparison, but the data has to be hashed before the comparison which takes extra time.

3.2.1. Locality Sensitive Hashing

Locality Sensitive Hashing (LSH) [26] is a form of hashing where the reference data $\{x_1, x_2, ...x_n\}$ is hashed into buckets $\{y_1, y_2, ..., y_m\}$ with $m < n$. This is performed in such a way that data points that are near each other are likely to be placed in the same bucket, while data points that are far from each other are likely to be placed in different buckets. Thus, unlike most hashing methods, Locality Sensitive Hashing tries to maximize hash collisions. It does this by making use of multiple hash functions, with one set of resulting buckets per hash function. The idea is that for different hashing functions, similar data points are hashed to the same bucket with a high probability. For LSH, there is one tweakable parameters n_{bits}, which represents the length of the resulting hashes.

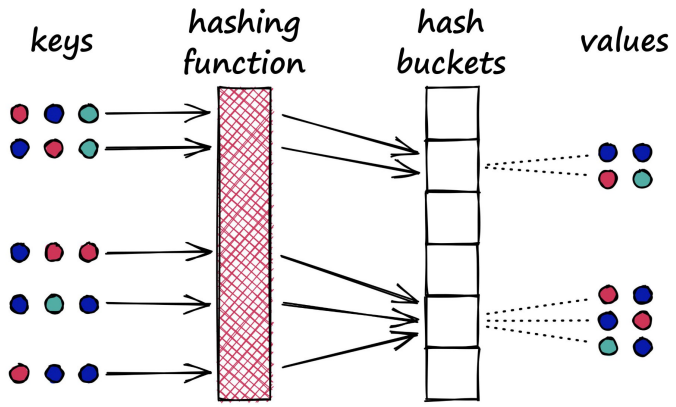


Figure 3.5: Visualisation of Locality Sensitive Hashing. Adapted from [29].

3.2.2. Product Quantization

Product Quantization (PQ) is a form of data compression where every vector x in the reference data is split into m shorter subvectors $\{u_1, u_2, ...u_m\}$. Afterwards, a small set of n centroids is defined and every subvector u is then assigned to its nearest centroid. This process is displayed in Figure 3.6. As such, a large vector x is compressed into a smaller vector y with dimension m of which every element u can only take on a value in a small range equal to the values of the determined centroids. As such, vector y will take up much less memory than vector x. For Product Quantization there are two main parameters that can be tuned. The first is the number of splits n_{splits} which dictates into how many subvectors each vector is split. The second is the bitlength which represents the amount of predetermined centroids. For bitlength l there are 2^l defined centroids.

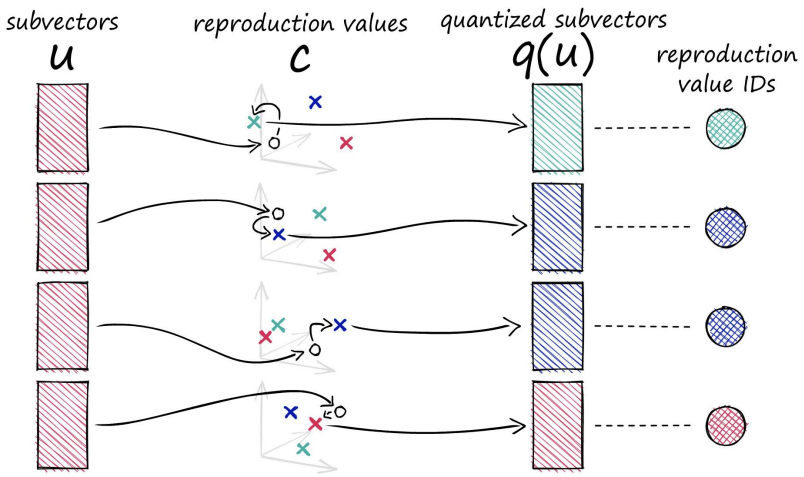


Figure 3.6: Visualisation of Product Quantization. Every subvector u is assigned to it s nearest centroid. Adapted from [30].

4

Design

4.1. Performance metrics

In order to select the optimal nearest neighbor and data compression methods to be used, their performances have to be compared. The methods are compared on three main metrics: total comparison time, mean average precision (mAP) and recall. The rest of this section will explain these metrics.

4.1.1. Total comparison time

As explained in chapter 3, some form of data partitioning is required for every NNS method except for linear comparison. The time required to partition the data is defined as the partitioning time $t_{partition}$. It is important to note that for most other nearest neighbor problems, this partitioning of the data can be done before the search has to be performed. As such, for those applications, only the search time per query is relevant for the performance of the nearest neighbor method. For the purposes of this video search engine however, the partitioning can not be performed beforehand as the keyframe data is not available before run-time. Thus, $t_{partition}$ should be taken into account when assessing performance. The partitioning of the data is required only once per query video, and as such does not depend on the amount of query images to be compared. Thus, a linear equation can be deduced for the total comparison time:

$$t_{total} = t_{partition} + t_{search} \times n_{queries} \tag{4.1}$$

where t_{total} is the total comparison time, t_{search} is the search time per query, $n_{queries}$ is the amount of queries and $t_{partition}$ is the time required for data partitioning.

4.1.2. Mean Average Precision

In order to determine the performance of the Nearest Neighbor Search methods, the Mean Average Precision (mAP) metric is used. This is a widely used metric that takes into account both the precision of a method and the order of the list of results it returns. It is defined as

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i \tag{4.2}$$

where mAP is the mean Average Precision, N is the number of queries and AP_i is the average precision for query i. In order to explain what the Average Precision is, first the relevance and precision metrics have to be explained. Relevance is an indicator that is equal to 1 for a relevant result and 0 for an irrelevant result. As such, the relevance for correctly retrieved nearest neighbors is equal to 1 and for incorrect retrieved nearest neighbors the relevance is zero. Precision is defined by the following equation:

$$Precision = \frac{TP}{TP + FP} \tag{4.3}$$

where TP is the amount of true positive results and FP is the amount of false positive results. Thus, precision is a metric that describes how many of the retrieved results of the NNS method are accurate.

This is used to calculate the average precision, which, unlike the name suggests, is not the average value of the precision for a set of queries. Instead, the average precision AP_i is described by the following equation:

$$AP_i = \frac{1}{GTP} \sum_{n=1}^{k} P@n \times rel@n \tag{4.4}$$

where AP_i is the average precision for query i for k nearest neighbors, GTP is the total number of Ground Truth Positives, k is the total amount of nearest neighbors, n is the index of the n-th nearest neighbor, P@n is the precision of the n-th nearest neighbor and rel@n is the relevance of the n-th nearest neighbor. The total number of Ground Truth Positives is equal to the expected amount of correct results and thus it is equal to k.

4.1.3. Recall

However, mAP is not the only relevant performance metric for the nearest neighbor search. A query object could appear many times within a query video and if only a subset of these appearances are detected, the mAP will be sufficient, but the system will not fulfill the recall requirements specified in 2. As such, a metric is needed that takes into account the ratio of positive results found by the method to the total amount of positive results in the dataset. The recall metric does just this:

$$recall = \frac{TP}{TP + FN} \tag{4.5}$$

where TP is the amount of True Positive results and FN is the amount of False Negative results. For the purposes of a Nearest Neighbor Search method, the recall for a single query image thus describes the ratio between the amount of correctly retrieved nearest neighbors for that query image and the total amount of correct nearest neighbors in the dataset for that query image. To minimalize the influence of outliers, the mean of the recall for 100 query images is used.

4.2. Dataset

In order to determine mAP and recall, knowledge of whether any retrieved result is relevant or not is required. As it is near impossible to manually label the relevance of all of the retrieved nearest neighbor keyframes in a large video, a pre-labeled dataset of feature vectors should be used instead. Considering it is difficult to find datasets of labeled feature vectors of the required length specified in Chapter 2, while large numbers of labeled datasets of images are publically available, the decision was made to extract features from a pre-labeled dataset of images instead, in which every image would represent one keyframe. Considering the Video Search Engine should be applicable to a broad range of applications, this dataset should be large enough to represent longer videos (2,5 hours). The keyframe extraction subgroup derived a lower bound of 85% compression ratio [1], which suggests that at maximum 15% of the total amount of frames in a video would be marked as keyframes. As such we can determine the expected amount of keyframes using the following relationship:

$$n_{keyframes} = \frac{t_{video} * f}{1 - C_{ratio}}$$

where $n_{keyframes}$ is the expected amount of extracted keyframes, T_{video} is the length of the video in seconds, f is the framerate of the video in frames per second and C_{ratio} is the compression ratio. Assuming a video of 2,5 hours, recorded at 30 fps and utilizing the lower bound of 85% compression ratio, the expected amount of extracted keyframes can be calculated to be 40500. As such, a labeled dataset that contains at least 40500 images should be used. Considering the hardware restrictions as described in chapter 2, these images should not be too large as the duration of the feature extraction is dependent on the size of the images. For these reasons, the CIFAR-10 dataset [31] was used. This dataset consists of 60.000 labeled images with size 32x32 pixels, which are split into 50.000 training images and 10.000 test images. The training images are categorized in 10 classes of 5.000 images each, and the test images are categorized in 10 classes of 1.000 images each.

Feature vectors of length 2048 were extracted for all of the images by making use of a triplet network implementation that was provided by one of the master students working on the Search Engine for Digital History [7]. Thus, a dataset of 50.000 labeled feature vectors, corresponding to the keyframes of a

video of 3.08 hours long was created. Datasets for shorter videos could then be simulated by creating a random subset of this dataset. This was done by reducing the amount of images per class, while preserving the distribution of the classes as to not favor one of the classes.

4.3. Pre-selection of methods

To determine which of the methods described in Chapter 3 are viable for real-time data partitioning, their performances have to be compared for videos of different lengths. This was done by measuring their partitioning time, average search time per query, mAP and recall for comparing 100 query images to three pre-set amounts of extracted frames. These values are 270, 8100 and 50000 and correspond to videos of length 1 minute, 30 minutes and 3.2 hours respectively. The results are shown in Tables 4.1, 4.2 and 4.3 respectively. To reiterate from the requirements in Chapter 2: the mAP should be at least 65% and the recall should be minimally 50%. Furthermore, the search time should be as low as possible. For values that require parameters as inputs, the values were based on their respective manuals and code examples.

Table 4.1: Comparison of performance for different NNS methods for comparing 100 query images to 270 keyframes for k = 10%

Method	Partitioning time (ms)	Search time per query (ms)	mAP	recall
L2	0	0.98	0.72	0.76
ANNOY [1]	48.04	0.5	0.72	0.76
HNSW [2]	28.19	0.09	0.71	0.75
HNSW batch [3]	21.02	0.08	0.72	0.76
FAISS FLAT L2	12.18|453.41[4]	0.33|0.0	0.72|0.72	0.76|0.76
FAISS HNSW [2]	5.00	0.01	0.72	0.76
FAISS IVF [5]	22.02|468.68	0.0|0.02	0.74|0.74	0.76|0.76
FAISS PQ [6]	70.06	0.0	0.72	0.76
FAISS LSH [7]	53.05	0.02	0.72	0.76

[1] Parameters: forest size = 10.
[2] Parameters: M=8, ef construct = 100, ef= k.
[3] Parameters: M=8, ef construct = 100, ef= k, batch size = 10% of the frames. Note that the partitioning time is equal to the construction time of only the last batch
[4] Left value specifies CPU performance, right value specifies GPU performance.
[5] Parameters: number of splits = 8, nprobe = 1.
[6] Parameters: number of vector splits = 8, number of bits = 8.
[7] Parameters: bitlength = 25% of the feature vector length.

Table 4.2: Comparison of performance for different NNS methods for comparing 100 query images to 8100 keyframes for k = 10%

Method	Partitioning time (ms)	Search time per query (ms)	mAP	recall
L2	0	43.59	0.7	0.75
ANNOY [1]	1038.43	3.91	0.71	0.75
HNSW [2]	1192.34	0.76	0.7	0.75
HNSW batch [3]	1118.12	0.75	0.7	0.75
FAISS FLAT L2	13.01|516.25[4]	0.3|0.3	0.7|0.7	0.75|0.75
FAISS HNSW [2]	202.22	0.18	0.70	0.75
FAISS IVF [5]	105.09|594.22	0.15|0.13	0.72|0.72	0.75|0.75
FAISS PQ [6]	1120.82	0.04	0.71	0.76
FAISS LSH [7]	81.07	0.04	0.7	0.75

[1] Parameters: forest size = 10.
[2] Parameters: M=8, ef construct = 100, ef= k.
[3] Parameters: M=8, ef construct = 100, ef= k, batch size = 10% of the frames. Note that the partitioning time is equal to the construction time of only the last batch
[4] Left value specifies CPU performance, right value specifies GPU performance.
[5] Parameters: number of splits = 8, nprobe = 1.
[6] Parameters: number of vector splits = 8, number of bits = 8.
[7] Parameters: bitlength = 25% of the feature vector length.

Table 4.3: Comparison of performance for different NNS methods for comparing 100 query images to 50000 frames for k = 10%

Method	Partitioning time (ms)	Search time per query (ms)	mAP	recall
L2	0	282.33	0.63	0.68
ANNOY [1]	6885.22	18.6	0.63	0.68
HNSW [2]	9514.22	5.52	0.63	0.68
HNSW batch [3]	4036.03	5.54	0.63	0.68
FAISS FLAT L2	84.08|558.6[4]	1.57|0.1	0.63|0.68	0.68|0.29
FAISS HNSW [2]	5907.23	20.13	0.63	0.68
FAISS IVF [5]	495.45|857.87	1.48|0.39	0.68|0.73	0.71|0.31
FAISS PQ [6]	6999.24	0.31	0.64	0.69
FAISS LSH [7]	291.26	0.24	0.62	0.67

[1] Parameters: forest size = 10.
[2] Parameters: M=8, ef construct = 100, ef= k.
[3] Parameters: M=8, ef construct = 100, ef= k, batch size = 10% of the frames. Note that the partitioning time is equal to the construction time of only the last batch
[4] Left value specifies CPU performance, right value specifies GPU performance.
[5] Parameters: number of splits = 8, nprobe = 1.
[6] Parameters: number of vector splits = 8, number of bits = 8.
[7] Parameters: bitlength = 25% of the feature vector length.

From the data shown in the tables it becomes clear that the mAP and recall are consistent for almost all of the methods with some minor outliers. However, it is important to note that the values for mAP and recall are not equal to 1, which would be expected for linear search at 50000 keyframes. The reason for this is related to the dataset: As explained in section 4.2, the dataset consists of 10 classes of images. Each class has 5000 corresponding images and 10% of 50000 keyframes equals finding 5000 nearest neighbours. Thus, all images belonging to a specific class should be found if k is set at 10% and thus both mAP and recall should be equal to the optimal value of 1. The reason for the deviations from ideal mAP and recall can be traced back to the feature extraction. If the features are not extracted perfectly then this error will propagate into the NNS. As such the mAP and recall for the linear method at 50000 keyframes can be regarded as the best performance available. Higher mAP and recall values in Table 4.1 and 4.2 are a result of taking a random subset of the dataset. The quality of the random samples

varies and as such a more favourable subset was created from the dataset.

The only outliers in the results are the GPU implementations of the FAISS IVF and FAISS FLAT L2 methods at 50000 keyframes. The reason for this is that GPU implementation in FAISS have an upper limit for $k = 2048$ nearest neighbours [20]. If k is equal to 10% of the keyframes, this issue would become apparent for videos that contain over 20480 keyframes. Because the amount of nearest neighbours found is limited, the recall has ma maximum amount of $\frac{2048}{5000} \approx 0.41$. On the other hand, the mAP is higher than the mAP for the other methods. The reason for this is that for searching a smaller amount of nearest neighbours, less false positives are detected. Since the mAP and recall are very similar between methods, the leading performance indicator is the time required for the search.

As described in section 4.1.1, the total comparison time is dependent not only on the search time and partitioning time, but also on the amount of queries. As such, the fastest method will vary for varying amounts of queries. The total comparison times for varying amounts of queries are displayed in Figure 4.1. The figures are adjusted such that they only display the effective time range. Methods may fall out of this scope due to the high partitioning time (e.g. ANNOY).
From the intersection points of the plots in Figure 4.1 a selection can be made of all methods that prove to be valuable:

- For 270 frames, linear is the optimal method, but at roughly nine queries it is overtaken by FAISS HNSW. For the scope of queries it stays the best method, but FAISS IVF CPU should also be taken into account since it comes close to FAISS HNSW and would overtake for query amounts larger than 1000.
- For 8100 frames, the only two methods of notice are FAISS FLAT L2 CPU and FAISS LSH. Their intersection is at roughly 260 queries.
- For 50000 frames, the same pattern occurs, but the intersection point is at roughly 155 queries. Furthermore, FAISS FLAT L2 GPU has the ability to overtake FAISS LSH for queries more than 1000. The eleven methods can be reduced to six promising methods with their conditions summarized in Table 4.4

Table 4.4: Promising methods and their performance restrictions

	Conditions	
Method	Amount of frames	Amount of queries
L2	low	low
FAISS FLAT L2 CPU	medium - high	low - medium
FAISS FLAT L2 GPU	high	very high
FAISS HNSW	low	low - high
FAISS IVF CPU	low	very high
FAISS LSH	medium - high	medium - high

4.4. Method optimisation

With a selection of the best performing nearest neighbour methods, the next step is to optimise the total comparison time of these methods as much as possible, while still fulfilling the mAP and recall requirements. This consists of optimising the amount of nearest neighbours to be found and method-specific parameters for methods that have them. These methods are FAISS HNSW, FAISS IVF and FAISS LSH.

4.4.1. Determining optimal amount of nearest neighbours

During the pre-selection of the methods, the amount of nearest neighbours k was set at 10% as based on theory. However, the value of k is a trade-off: lower k values have shorter search times, lower recall and higher mAP values, while higher k values have longer search times, higher recall and lower mAP values. In order to determine the optimal value of k, the search time, mAP and recall for the preselected methods were measured for the pre-selected amounts of frames by varying k. The results of these measurements can be found in Figures 4.2, 4.3 and 4.4. The raw data can be found on

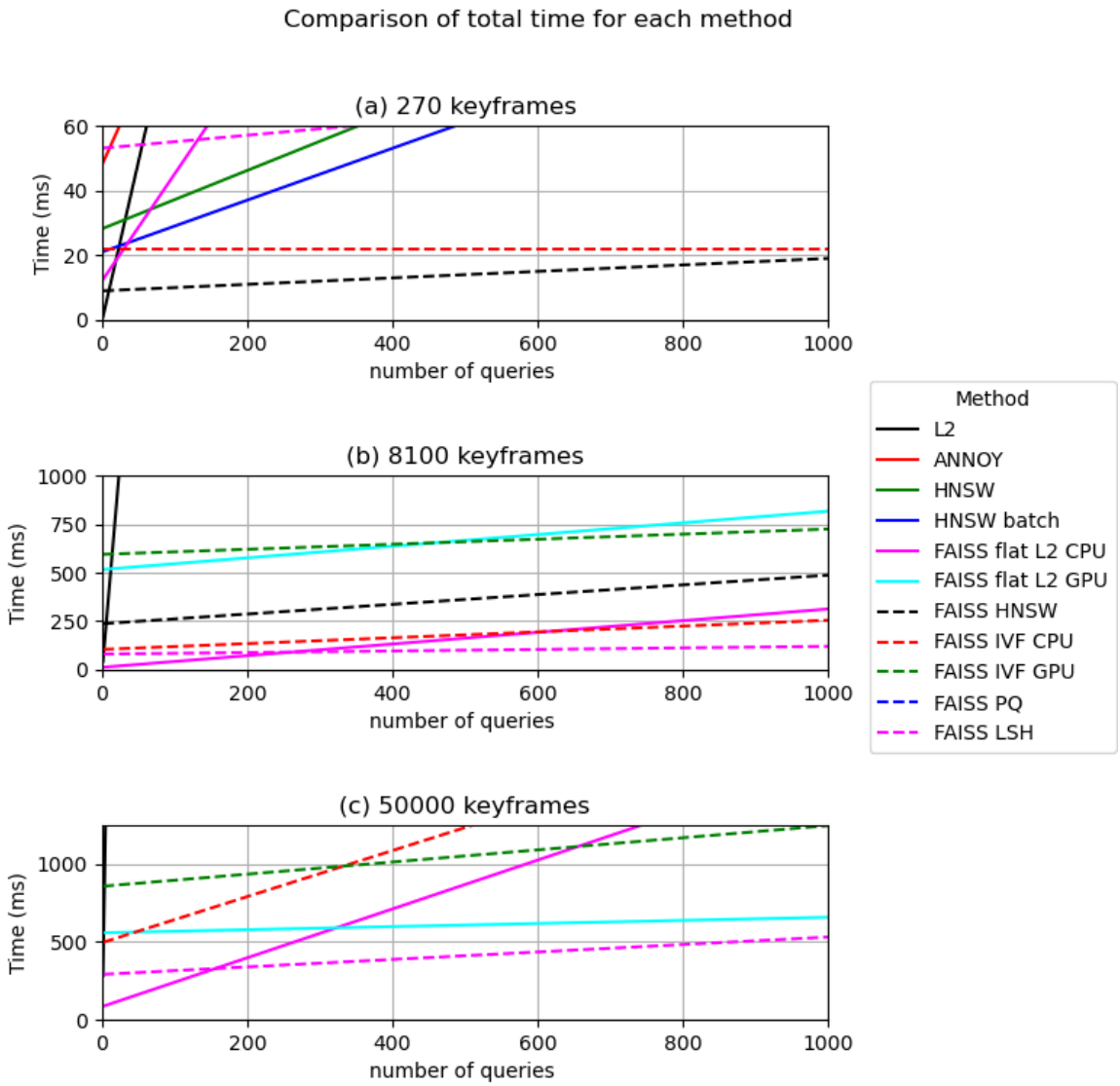


Figure 4.1: Total comparison times for comparing varying amounts of queries against (a) 270, (b) 8100 and (c) 50000 keyframes

the GitHub [32]. In these figures, the minimum requirements for the mAP and recall are displayed as horizontal dotted lines.

From these figures it becomes apparent that a range of values for k can be chosen that fulfills both the mAP and recall requirements. The bounds are as follows:

- 50000 queries: k_{max} for mAP requirement = 8.5%, k_{min} for recall requirement = 7%
- 8100 queries: k_{max} for mAP requirement= 10.8%, k_{min} for recall requirement = 6.5%
- 270 queries: k_{max} for mAP requirement= 11.3%, k_{min} for recall requirement= 6.4%

As the mAP and recall requirements are met for any point within this range, the decision was made to select k=7%, in order to maximise the mAP. This value of k=7% is plotted as a vertical line in Figures 4.2, 4.3 and 4.4.

This value for k does not hold for videos of all lengths however. For short videos, where only a small

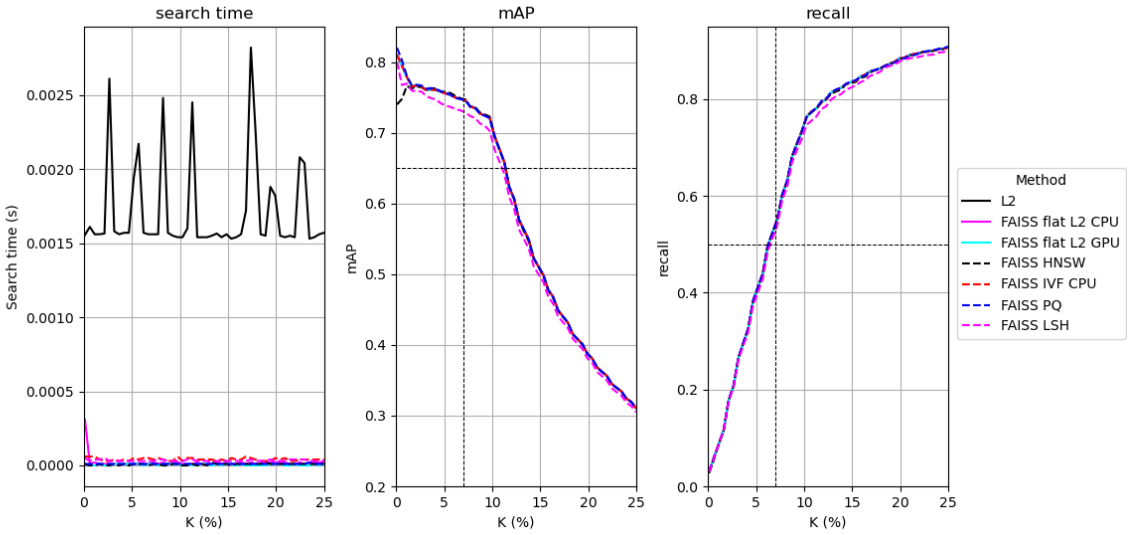


Figure 4.2: Search time, mAP and recall for varying k for a dataset of 270 keyframes for varying methods

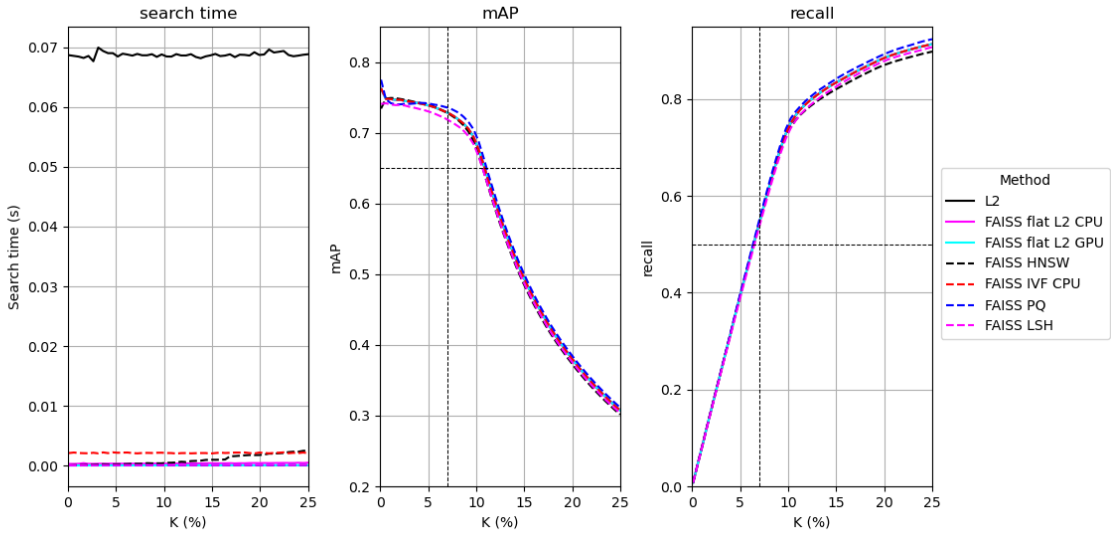


Figure 4.3: Search time, mAP and recall for varying k for a dataset of 8100 keyframes for varying methods

amount of keyframes is extracted, performance will decrease substantially if the value for k is too low, as only one or two nearest neighbours will be found. This can cause the performance to drop below the thresholds specified in the programme of requirements 2. To combat this, a threshold for query video length needs to be determined, below which using $k = 7\%$ is unsatisfactory. This is done by calculating the mAP and recall for $k = 7\%$ for a range of short videos. The results can be found in table 4.5. For videos of 30 frames and longer, the recall satisfies the minimum requirement of 50% as specified in Chapter 2. Because a random subset is taken, as described in Section 4.2, this subset might be favoured and thus produce higher results than a fairly balanced set. To have a bit of leeway for the recall, the number of frames should be set at 40. This means that for videos where less than 40 keyframes are extracted, every keyframe should be evaluated, rather than just the $k = 7\%$. However, this results in a big drop of the mAP, since most of the returned results are not relevant. To accommodate for this, a final evaluation can be made, based on the distance of the matches, to discard non-relevant matches. In conclusion, for 40 or less keyframes k should be set at 100% of the

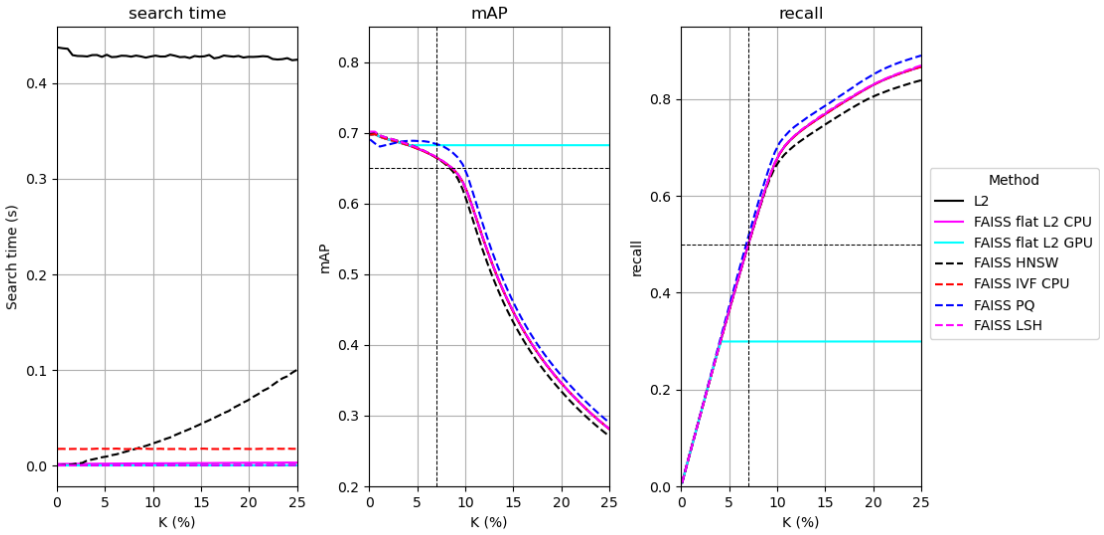


Figure 4.4: Search time, mAP and recall for varying k for a dataset of 50000 keyframes for varying methods

keyframes and for more than 40 it should be set at 7%.

Table 4.5: mAP and recall for linear comparison for small amounts of frames

number of frames	k at 7%	mAP	recall
10	1	0.78	0.78
20	1	0.74	0.37
30	2	0.75	0.50
40	3	0.76	0.58
50	4	0.76	0.62

4.4.2. FAISS HNSW: Determining optimal parameters

Now that k has been determined, the method-specific parameters of FAISS HNSW have to be optimized. As described in section 3.1.5, these consist of the construction efficiency parameter ef_{const} and the amount of connections M for every point in the graph. In order to determine the optimal value for these parameters, hyperparameter optimization was performed for the dataset of 270 keyframes it performed well on during the pre-selection process. The hyperparameter optimization was implemented by making use of Optuna [33]. This optimization was done by maximizing mAP and recall while minimizing partitioning time and total query time. The results of this optimization study can be found on the GitHub repository [32]. From these results, the trials that did not fulfill the mAP and recall requirements were filtered out. For the remaining trials, the total comparison time for varying amount of queries was calculated as described in section 4.1.1. Then, the optimal trial for the total range of 1-1000 queries was determined and its parameters noted. The optimal parameters for varying amount of queries can be found in table 4.6.

Table 4.6: Optimal parameters for FAISS HNSW for varying amount of queries for 270 keyframes

Amount of queries	nprobe	splits	mAp	recall	time per query	partitioning time
1-1000	58	24	0.758143	0.557670	0.000003	0.002003

4.4.3. FAISS IVF: Determining optimal parameters

For FAISS IVF a similar optimization study was performed to find the optimal values for the amount of splits and nprobe. Individual optimization studies were performed for both datasets is performed well on during the pre-selection process: 270 and 8100 frames. The results of these optimization studies can be found on the GitHub repository [32]. From these results the same methodology was applied for finding the optimal parameters for varying amount of queries. These optimal parameters can be found in Table 4.7.

Table 4.7: Optimal parameters for IVF for varying amount of queries for a dataset of 270 keyframes

Amount of queries	nprobe	splits	mAp	recall	time per query	partitioning time
1-122	1	2	0.757584	0.555556	0.000011	0.003002
123-1000	1	5	0.766144	0.557222	0.000003	0.004003

Table 4.8: Optimal parameters for IVF for varying amount of queries for a dataset of 8100 keyframes

Amount of queries	nprobe	splits	mAp	recall	time per query	partitioning time
1-4	3	4	0.745071	0.552691	0.000455	0.073066
5-10	1	4	0.748259	0.548764	0.000158	0.074066
11-89	1	4	0.748259	0.548764	0.000158	0.074068
90-258	1	6	0.752843	0.550838	0.000112	0.078071
259-508	1	8	0.761731	0.554189	0.000085	0.085076
509-1000	1	12	0.727798	0.527254	0.000063	0.096262

4.4.4. FAISS LSH: Determining optimal parameters

For FAISS LSH, again, a similar optimization study was performed to find the optimal values for the bitlength percentage for all three datasets. The results of this optimization study can be found on the GitHub repository [32]. A notable difference between the results of this study and those for the other FAISS methods is that one trial dominated the other trials for every single amount of queries. As such, instead of a table, a graph for every dataset is included. The result can be found in Table 4.9, 4.10 and 4.11

Table 4.9: Optimal parameters for FAISS LSH for varying amount of queries for 270 keyframes

Amount of queries	bitlength percentage	mAP	recall	time per query	partitioning time
1-1000	0.04	0.676598	0.509063	0.000002	0.002001

Table 4.10: Optimal parameters for FAISS LSH for varying amount of queries for 8100 keyframes

Amount of queries	bitlength percentage	mAP	recall	time per query	partitioning time
1-1000	0.09	0.689293	0.517708	0.000022	0.028025

Table 4.11: Optimal parameters for FAISS LSH for varying amount of queries for 50000 keyframes

Amount of queries	bitlength percentage	mAP	recall	time per query	partitioning time
1-1000	0.09	0.691888	0.518794	0.000130	0.134121

From these figures it is apparent that a bitlength percentage of 0.04 is optimal for the dataset of 270 frames, while a bitlength percentage of 0.09 is optimal for both larger datasets.

4.5. Final method selection

With all of the pre-selected methods optimised, the optimal method can be selected for varying queries for the three frame amounts. This is done by plotting the total comparison time versus the amount of queries for all six methods. These plots can be found in Figure 4.5. Compared to the performance of

these methods in Figure 4.1 the optimization has led to a lower total time for all six methods, while mAP and recall requirements are still met. From this plot, the optimal method for every combination of queries

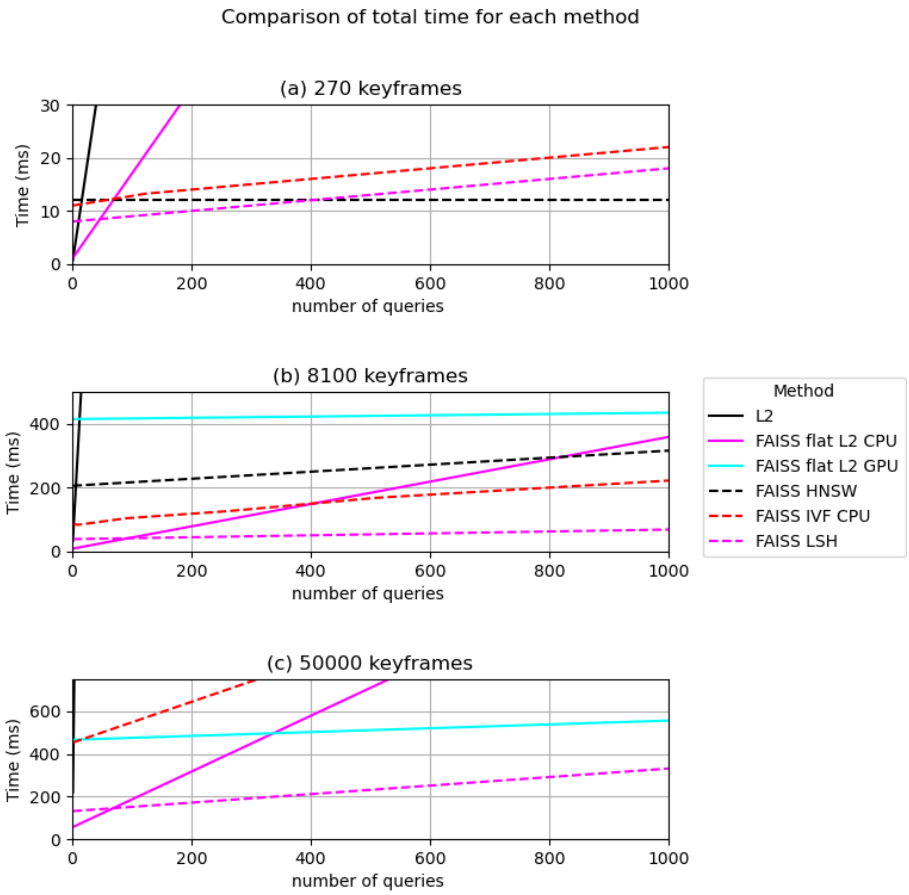


Figure 4.5: Total comparison time for varying number of queries for optimized methods against (a) 270, (b) 8100 and (c) 50000 keyframes

and the three set amounts of frames can be found by inspecting the breakpoints. These methods and their respective breakpoints can be found in Table 4.12. From this table it becomes apparent that only the methods Linear, FAISS L2 CPU, FAISS LSH and FAISS HNSW will be selected by the model for the pre-determined amounts of frames. For differing amounts of frames, the method-specific parameters are linearly interpolated from the data in Tables 4.6 - 4.11.

Table 4.12: Optimal method breakpoints for (a) 270, (b) 8100 and (c) 50000 keyframes

(a) Breakpoints at 270 frames

number of queries	method
1-2	linear
3-47	FAISS L2 CPU
48-400	FAISS LSH
401-1000	FAISS HNSW

(b) Breakpoints at 8100 frames

number of queries	method
1-94	FAISS L2 CPU
95-1000	FAISS LSH

(c) Breakpoints at 50000 frames

number of queries	method
1-69	FAISS L2 CPU
70-1000	FAISS LSH

5

Prototype implementation and validation

This prototype implementation chapter consists of two parts. The first part considers the implementation and validation of the Nearest Neighbour Search module, and the second part describes the implementation of the complete Image-based Video Search Engine, which combines the three modules. The code for the implementation of the full system and the Nearest Neighbour Search Module can be found on GitHub [32] that was used for developing the Image-Based Video Search Engine.

5.1. Nearest Neighbour Module

For the implementation of the Nearest Neighbour Search, three restrictions have to be taken into account. Two of the restrictions are related to the inputs and outputs of the module. These ensure that there are no miscommunications between the modules. The final restriction is a result of the measurements described in Chapter 4.

- The input should consist of two arrays. One array contains the extracted features of the keyframes, and the other one contains the extracted features of the query image(s).
- The output should consist of two arrays. One contains the k sorted nearest neighbour frames' indices, with the indices corresponding to the indices of the keyframes. The other one contains the respective distances that were calculated.
- A selector must be used to select the optimal method to be used, based on the amounts of keyframes and queries.

The implemented prototype first selects the NNS method to be used, based on the total number of keyframes and queries. Next, the selected method is called and the correct parameters for the corresponding method are determined. These parameters includes the k nearest neighbours to be found and any method specific parameters if applicable. Then, the selected method returns the indices and distances of the k nearest neighbours, thus complying with the final restriction.

5.1.1. Selector module

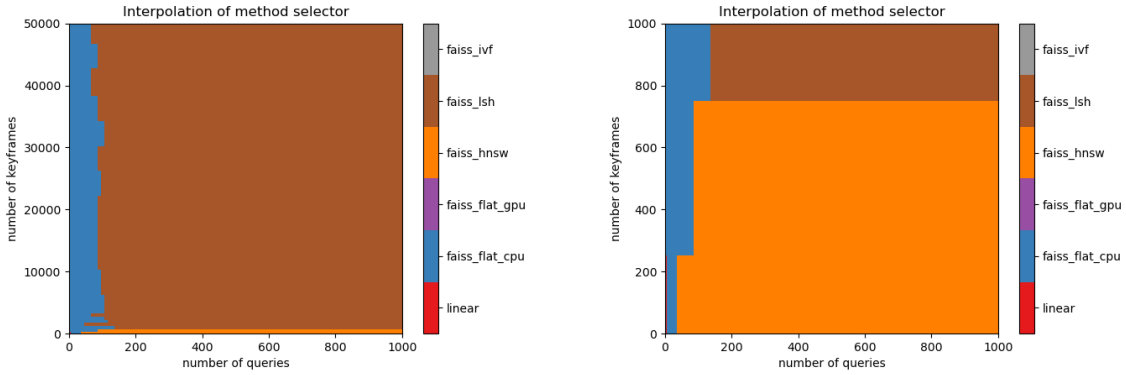
The heart of the NNS module is the selector module: the module which selects which NNS or data partitioning method to use. Based on the results of Table 4.12, such a selector could be implemented by making use of nearest neighbour interpolation for the methods. However, the selector would perform suboptimally due to the large ranges where interpolation would be required. To reduce the ranges where interpolation is required and to improve the selector, the optimal method was calculated for a varying amount of keyframes over 2 ranges, namely:

1. Every 4050 keyframes (corresponding to 15 minutes of video as per the relation described in Section 4.2) over the entire range of the dataset.
2. Every 270 keyframes (corresponding to 1 minutes of video as per the relation described in Section 4.2) over the range of the first 4050 keyframes (corresponding to the first 15 minutes of video).

The goal of inspecting the first range is to improve performance in the area in which the method selection is rather consistent. As such, large intervals of keyframe amounts were chosen. The goal of inspecting the second range is to improve quality in the range where the methods vary more and as such a smaller interval was chosen. The values for these ranges were added to the selector and nearest neighbour interpolation was used to determine the optimal methods for varying keyframe amounts. The resulting behaviour of the selector is shown in 5.1.

From these figures four approximate regions can be defined:

- For single queries in short videos (1 - 200 keyframes), the selector selects linear comparison as the fastest method. Note that this range is very difficult to spot in Subfigure (a) of Figure 5.1 and thus a close-up was added in Subfigure (b).
- For small amounts of queries (2-100) in videos of any length and for larger amounts of queries (200-100) in very short videos, the selector selects FAISS linear comparison on CPU-basis as the fastest method.
- For larger amounts of queries (100-1000) in videos of up to 700 keyframes (2.6 minutes) the selector selects FAISS HNSW as the fastest method.
- For larger amount of queries (100-1000) in videos of 700+ keyframes, the selector selects FAISS LSH as the fastest method.



(a) Optimal methods over the entire range of keyframes

(b) Close-up of plot (a): Optimal methods over a range of 1000 keyframes

Figure 5.1: Methods selected by the method selector for varying amounts of queries and keyframes

5.1.2. Validation of the selector module

In order to test the validity of the selector module, two aspects have to be tested:

- The mAP and recall of the NNS methods selected by the selector module,
- The speed of the selected method compared to other methods

The first aspect can be tested by inspecting the mAP and recall for the method calculations in section 4.5. The outliers for these calculations are displayed in Table 5.1. Only at 50000 keyframes do all four of the methods fail to meet the requirements of 50% recall and 65% mAP. As explained in 4.3, in the mAP there is a propagation of a inaccuracy in the quality of extracted features. As for the recall, for which all four methods fail to perform, with k at 7% the maximum possible recall is equal to $\frac{0.07*50000}{5000} = 0.70$. This maximum would be reached only if the features had been extracted perfectly. Thus the problem stemming from the inaccurately extracted features propagates here aswell.

To test the speed of the selected methods, ten random combinations of keyframe and query image amounts were tested in order to determine which method selected by the selector. Afterwards, these selected methods were compared to the actual fastest method. The results can be found in Figure 5.2. Due to the number of frames, the dataset cannot be equally split according to Section 4.2. For this

Table 5.1: Conditions where mAP and recall are not met for number of queries = 100

method	number of frames	mAP	recall
Linear	50000	0.671	0.496
FAISS FLAT CPU	50000	0.671	0.497
FAISS HNSW	50000	0.671	0.497
FAISS LSH	50000	0.647	0.489

Table 5.2: Comparison of the method selected by the selector to the fastest method.

number of frames	number of queries	selected method	fastest method	total time (ms)	mAP	recall
15519	322	FAISS LSH	FAISS LSH	70.06	0.66	0.50
44	506	FAISS HNSW	FAISS HNSW	4.00	0.65	0.51
28959	123	FAISS LSH	FAISS LSH	100.09	0.66	0.50
9806	75	FAISS flat CPU	FAISS flat CPU	36.03	0.69	0.51
41357	661	FAISS LSH	FAISS LSH	214.19	0.68	0.51
23857	838	FAISS LSH	FAISS LSH	119.11	0.68	0.51
6924	565	FAISS LSH	FAISS LSH	40.04	0.68	0.51
7993	2	FAISS flat CPU	FAISS flat CPU	12.01	0.99	0.73
3192	448	**FAISS HNSW**	**FAISS LSH**	22.02	0.66	0.50
932	588	FAISS LSH	FAISS LSH	12.01	0.68	0.51

reason the equal distribution of classes in the subset was dropped. This also means that the dataset now reflects a somewhat more realistic scenario. The amount of keyframes that actually contain the object of interest will now vary, whereas previously this was always equal to 10% of the keyframes. First of all, looking at the mAP and recall, the system still meets the requirements of 65% mAP and 50% recall. As for the performance of the method selector, it is performing quite well. The only mistake was made for 3192 keyframes and 448 query images as highlighted in Table 5.1. Looking at Figure 5.1 this same selection of the HNSW method can be seen.

5.1.3. Bookkeeping module

In order to comply with system requirement 2 of the programme of requirements in Chapter 2, the video search engine must return the timestamp where the query object appears in the video. Thus far, this module focuses solely on finding the frames that are the nearest neighbours of the query image, but no time stamp is returned yet. For this reason, an extra module is required that:

- Filters out the found nearest neighbour keyframes that have high distances to the query image.
- Converts the index of the left-over nearest neighbour keyframes to a timestamp based on the frame rate of the query video.

This filter is based on the distances that are returned by the Nearest Neighbour Search. The optimal cutoff for the filter was empirically established to be a distance of 1.13, based on the results of the datasets as explained in the Feature Extraction thesis [2]. However, this minimum distance does not guarantee perfect results. Therefore, the filter strength is designed to be adjustable by the user. This allows for a trade-off between returning few, but accurate results and returning more, less accurate results. The timestamp conversion is done by dividing the frame number, as explained in [1], by the frame rate.

5.2. Image-Based Video Search Engine

5.2.1. Implementation

For the prototype of the entire system it is important that it is both easy to use and fast in execution. Otherwise, it is not attractive for other people to use or improve on. To that end the following constraints for the prototype are chosen:

- The three modules will each reside in their own Python sub package, making them easy to devel-

op/maintain individually;

- The final prototype will run as a Python application on a host machine. The end-user will be able to upload the video and one or more search images via the application.
- The input to the complete system consists of: a single input video (in mp4 format), and one or more query images (in JPG format);
- The output of the complete system contains the timestamps in which the object of the query image(s) appears.

In- and Outputs
In order to ensure smooth development between the different modules, the in- and outputs of each module will be defined. The scheme can be found in Figure 5.2.

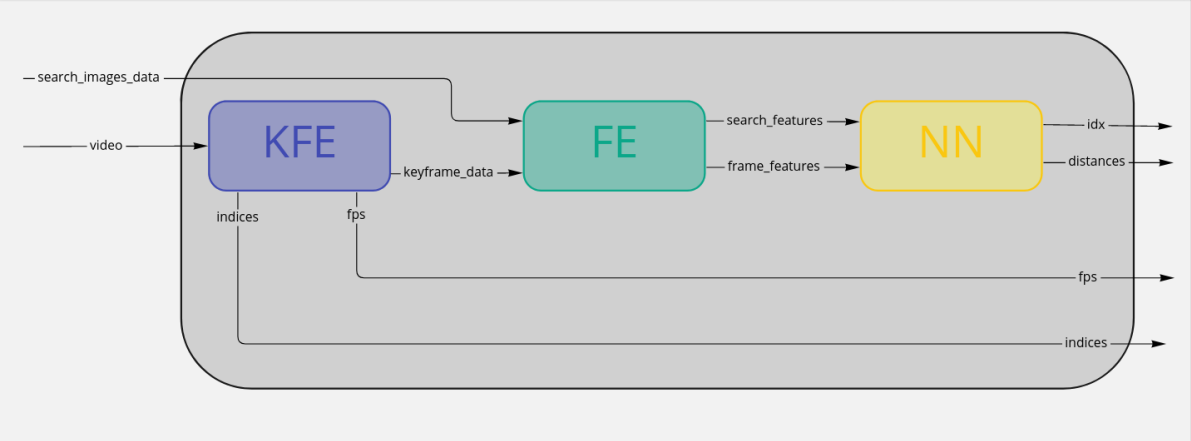


Figure 5.2: The in- and outputs of all the modules.

Timings
One of the most important performance metrics of the system is the execution time. In order to optimise the total execution time, the running time of all three modules will be measured so that the modules can be fine-tuned. The most time consuming module will most likely be the Feature Extraction module (FE) (if all frames of the query video will be used). For each of the images received by FE it will run a lot of calculations. A way to reduce this execution time is to reduce the number of images it has to process. This reduction is performed by the Keyframe Extraction module (KFE). The goal of KFE is to reduce the execution time of FE without increasing the total execution time or losing visual content. This is equivalent to minimising Equation 5.1:

$$t_{sum} = t_{KFE} + t_{FE} + t_{NN} \tag{5.1}$$

where $t_{[\cdot]}$ corresponds to the execution time of module $[\cdot]$.

For the measurement of the system the overhead of the Python application will not be considered, so only the time measurement of the individual modules will be performed. For each of the modules the time will be measured individually and the total time t_{total} (from the start to the end of the script) of the prototype will be measured. The total execution time is equal to:

$$t_{total} = t_{sum} + t_{overhead} \tag{5.2}$$

where $t_{overhead}$ is the extra time (overhead) of loading in the query images and other calculations performed outside of the modules t_{KFE}, t_{FE} and t_{NN}.

To fulfill system requirement 10, the total time should be converted to a ratio. This can be done using Equation 5.3.

$$\frac{t_{total}}{t_{video}} \leq 0.5 \tag{5.3}$$

where t_{video} is the length of the video.

Precision
The precision is just as important as the execution time. The system could execute really fast, but if the results do not show the correct timestamps of the video then the small execution time has no value. For that reason the mean Average Precision (mAP) will be used to evaluate the precision of the complete system. The mAP will be calculated using Equation 4.2.

The average precision AP_i that is used for the mAP calculation is obtained using the following formula:

$$AP@k = \frac{1}{k}\sum_{n}^{k} P@n \times rel@n \tag{5.4}$$

where k refers to the total number of timestamps at the output of the system (which is 7% of the keyframes as described in section 4.4.1, n refers to the rank of the timestamp at the output, $P@n$ refers to the precision@n and $rel@n$ is the relevance function at n. The relevance function equals either one or zero: $rel@n = 1$ if the timestamp at rank n is relevant and $rel@k = 0$ otherwise. The precision@n can be calculated using Equation 4.3.

5.2.2. Validation

Timings
In Table 5.3 the timing results can be found for the 'easy' dataset. For all of the test scenarios, system requirement 10 is satisfied, as can be seen in column 'Ratio'. The column 'Video' shows the name of the query video in combination with the query image, so *Battuta1_1* corresponds to the first query video of Battuta and the first query image. Table 5.1 found in [2] also shows the different query videos with the amount of available query images. Due to time constraints, not all videos and query images were evaluated.

Precision
The performance of the system will not only be described using the ratio between the execution time of the system and the duration of the query video, but also with the help of the mean Average Precision. The mAP was calculated following Subsection 5.2.1. The mAP calculations can be found in Tables 5.4 and 5.5. The tables show the mAP with and without applying the distance filter. The filter reduces the amount of results by discarding all matches above a certain distance threshold and retaining only those above that threshold. By using the filter, the mAP of the system increases significantly.

Table 5.3: Time measurements of the prototype for various query videos and images from the 'easy' dataset. The *Ratio* is defined as in Eq. 5.3. A resize of $1024 \times 576p$ was used (as chosen by the Feature Extraction module [2].

Video	t_{video} [s]	t_{KFE} [s]	t_{FE} [s]	t_{NN} [s]	t_{total} [s]	Ratio
Battuta1_1	261	28.25	75.50	0.0023	103.80	**0.40**
Battuta1_2	261	27.73	76.84	0.0018	104.63	**0.40**
Batutta1_3	261	28.17	76.29	0.0019	104.55	**0.40**
Battuta1_4	261	28.57	75.98	0.0016	104.57	**0.40**
Battuta2_1	188	39.61	56.97	0.0018	96.68	**0.51**
He1_1	274	26.66	56.31	0.0047	83.06	**0.30**
He1_2	274	27.04	57.35	0.0019	84.43	**0.31**
He1_3	274	26.68	58.41	0.0022	85.18	**0.31**
He2_1	113	1.56	30.84	0.0049	32.43	**0.29**
He3_1	187	13.33	23.53	0.0070	36.89	**0.20**
He3_2	187	13.48	24.44	0.0018	37.92	**0.20**
He3_3	187	13.98	23.70	0.0050	37.70	**0.20**
Polo1_1	323	32.55	71.37	0.0022	103.95	**0.32**

Table 5.4: mAP results of the prototype for the 'easy' dataset.

	mAP	
	Without Filter	With Filter
Battuta1	0.74	1
Battuta2	0.59	1
Battuta3	0.31	1
Battuta4	0.16	0.67

Table 5.5: mAP results of the prototype for the 'hard' dataset.

	mAP	
	Without Filter	With Filter
Ewi1	0.125	No matches
Ewi2	0.25	1
Dutch mailbox	0.29	1

6

Conclusion

6.1. Nearest Neighbour Module

The requirements from Chapter 2 have largely been fulfilled and will be discussed in this section. To reiterate the problem statement from Chapter 1:

- *Develop an algorithm that can compare the features of the query image(s) to the features of the keyframes.*

Module requirements 1, 2 and 3 are fulfilled since the module, as described in Section 5.1.1, performs the search and returns the nearest neighbours of the query image. Module requirement 4 is met since the amount of nearest neighbours returned is equal to 7% of the amount of keyframes as described in Section 4.4.1. In compliance with Module requirement 5, the partitioning times are taken into account for designing the module as described in Section 4.1.1. From the results obtained in Chapter 5, it is clear that Module requirement 6, describing the recall, is met for almost all implemented methods with a rare exception for large amounts of keyframes. These results are caused by the lacking quality of the extracted features by the triplet network, as described in Section 5.1.2. Module requirement 7 is fulfilled since the system uses FAISS implementations and linear search in python to perform the comparison and no new methods were developed. Module requirement 8 is met, as Figure 4.5 shows that the other methods used are faster than linear search.

As the mandatory requirements are met, the trade-off requirements can be inspected. Module requirement 9 is fulfilled by use of an optimal method selector as described in section 5.1.2. The Nearest Neighbour Search module adheres to the requirements and functions well. There are a few minor problems as discussed in Section 5.1.2 that can be resolved to improve the module.

6.1.1. Discussion

The performance meets the specified requirements, but can be further improved upon. As the features of the CIFAR-10 dataset were extracted by making use of a triplet network, the feature vectors are not completely accurate. As such, even linear methods which should have perfect recall (because it compares all of the data) and mAP (because it is a balanced dataset in which any search where k is smaller than 10% should return only correct results) have values for the mAP and recall that are less than ideal.
Due to the time constraints of the Bachelor Graduation Project, as described in chapter 2, only pre-built implementations of NNS methods were considered, rather than writing new methods or optimizing the code of existing methods. Additionally, considering the hardware constraints of the group, the hyper-parameter optimization for the FAISS methods was performed for only 25^n trials, where n is equal to the amount of input parameters the method has. Even for this low amount of trials, this took hours to run on the available hardware and as such the decision was made not to increase the amount of trials. Finally, the validation of the selector module was performed based on only ten frame-query combinations because of time constraints. For a more thorough analysis, this validation should be done on a

larger amount of datapoints.

6.2. Image-Based Video Search Engine

The requirements from Chapter 2 have been fulfilled and will be discussed in this section. To reiterate the problem statement from Chapter 1:

- *Develop a system that can detect whether an instance of a desired object appears in a given video, based on a given set of images containing the desired object.*

The core functional requirements 1, 2 and 3 are met as explained in section 5.2. Requirements 4, 5, 7 and 8 have been fulfilled and the results can be found on the GitHub [32]. Requirements 6 and 9 have both been fulfilled, because a prototype has been developed that can run on a laptop with requirements specified in the Programme of Requirements as described in [2]. Requirement 10 and by extension requirement 18 have been met as described in section 5.2. Requirement 11 and 14 have been fulfilled through the implementation of the filter module as described in Subsection 5.1.3

The trade-off requirements specify requirements that lead to increasing customer satisfaction as they are fulfilled. Requirements 12 and 13 are met as the prototype is able to deal with multiple query videos and images. This can be seen in the GitHub [32]. Requirement 15 has been fulfilled to the best of our ability. Requirements 16 and 17 are based on the video and image reading libraries used, which support a variety of formats. These are OpenCV [34] and Pillow (PIL) [35] respectively and the supported formats can be found in their respective documentations. Requirement 19 has been tested but is not fully met. The system works for videos up to 1 hour and 20 minutes.

6.2.1. Discussion

The system meets almost all of the stated requirements, but it can not handle longer videos. This is caused by the Keyframe Extraction Module requiring large amounts of memory, that the specified hardware rig does not possess. This is further explained in the Keyframe Extraction thesis [1].

6.2.2. Future Work

For future research and future BAP groups working on this project, investigating faster Keyframe Extraction and Feature Extraction implementations could significantly speed up the system. For a 6-person group working on the next generation of the project, the team could be split into three students working on the Keyframe Extraction module and three students working on the Feature Extraction module, while re-using the Data Compression and Nearest Neighbour Search implementation explained in this thesis. Additionally, this Image-Based Video Seach Engine was designed for the general use case and performance can be improved by focusing on a specific use case. For such a situation, selecting a Feature Extraction network that was trained for that use case will yield even better results.

Bibliography

[1] R. Bos and L. Zheng, *Image-based video search engine: Keyframe extraction*.

[2] M. van Oort and K. Hoogeveen, *Image-based video search engine: Feature extraction*.

[3] A. Nanetti, *Engineering historical memory*. [Online]. Available: http://engineeringhistoricalmemory.com.

[4] I. E. Lager, V. Scholten, E. Bol, C. Richie, and S. Izadkhast, *Bachelor graduation project manual*, 2022.

[5] M. Deutman, P. Groet, and O. van Hooff, "Image search engine for digital history, a standard approach," [Online]. Available: http://resolver.tudelft.nl/uuid:d0b96e9b-d383-448e-9342-db0b2560b560.

[6] M. van Geerenstein, P. van Mastrigt, and L. Vergroesen, "Image search engine for digital history, a deep-learning approach," [Online]. Available: http://resolver.tudelft.nl/uuid:f1a2902b-14be-416c-ae1a-ce4f179a0425.

[7] Y. Yao, "[unpublished manuscript]," [Online]. Available: https://github.com/YYao-42/ImgSearch.

[8] W. Chen, Y. Liu, W. Wang, *et al.*, "Deep image retrieval: A survey," *ArXiv*, 2021.

[9] F. Condorelli, F. Rinaudo, F. Salvadore, and S. Tagliaventi, "A neural networks approach to detecting lost heritage in historical video," *ISPRS International Journal of Geo-Information*, vol. 9, no. 5, 2020, ISSN: 2220-9964. DOI: 10.3390/ijgi9050297. [Online]. Available: https://www.mdpi.com/2220-9964/9/5/297.

[10] A. Araujo, M. Makar, V. Chandrasekhar, *et al.*, "Efficient video search using image queries," in *2014 IEEE International Conference on Image Processing (ICIP)*, 2014, pp. 3082–3086. DOI: 10.1109/ICIP.2014.7025623.

[11] Sivic and Zisserman, "Video google: A text retrieval approach to object matching in videos," in *Proceedings Ninth IEEE International Conference on Computer Vision*, 2003, 1470–1477 vol.2. DOI: 10.1109/ICCV.2003.1238663.

[12] F. Condorelli, F. Rinaudo, F. Salvadore, and S. Tagliaventi, "A neural networks approach to detecting lost heritage in historical video," *ISPRS International Journal of Geo-Information*, vol. 9, no. 5, p. 297, 2020.

[13] B. Luo, X. Wang, and X. Tang, "World-Wide-Web-based image search engine using text and image content features," in *Internet Imaging IV*, S. Santini and R. Schettini, Eds., International Society for Optics and Photonics, vol. 5018, SPIE, 2003, pp. 123–130. DOI: 10.1117/12.476329. [Online]. Available: https://doi.org/10.1117/12.476329.

[14] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009, ISBN: 1441412697.

[15] F. Groh, L. Ruppert, P. Wieschollek, and H. Lensch, "Ggnn: Graph-based gpu nearest neighbor search," *IEEE Transactions on Big Data*, pp. 1–1, 2022. DOI: 10.1109/TBDATA.2022.3161156.

[16] E. Bernhardsson, "Nearest neighbors and vector models – part 2 – algorithms and data structures," Oct. 2015. [Online]. Available: https://erikbern.com/2015/10/01/nearest-neighbors-and-vector-models-part-2-how-to-search-in-high-dimensional-spaces.html.

[17] E. Bernhardsson, *Annoy: Approximate nearest neighbors in c++/python*, Python package version 1.17.0, 2022. [Online]. Available: https://pypi.org/project/annoy/.

[18] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 4, pp. 824–836, 2018.

[19] Y. Malkov, *Hnswlib - fast approximate nearest neighbor search*, Python package version 0.6.2, 2022. [Online]. Available: https://pypi.org/project/hnswlib/.

[20] J. Johnson, M. Douze, and H. Jégou, *Faiss*, 2022. [Online]. Available: https://github.com/facebookresearch/faiss.

[21] V. Prasatha, H. A. A. Alfeilate, A. Hassanate, *et al.*, "Effects of distance measure choice on knn classifier performance-a review," *arXiv preprint arXiv:1708.04321*, p. 56, 2017.

[22] J. Han, M. Kamber, and J. Pei, "2 - getting to know your data," in *Data Mining (Third Edition)*, ser. The Morgan Kaufmann Series in Data Management Systems, J. Han, M. Kamber, and J. Pei, Eds., Third Edition, Boston: Morgan Kaufmann, 2012, pp. 39–82, ISBN: 978-0-12-381479-1. DOI: https://doi.org/10.1016/B978-0-12-381479-1.00002-2. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123814791000022.

[23] T. Rosa, R. Primartha, and A. Wijaya, "Comparison of distance measurement methods on k-nearest neighbor algorithm for classification," *vol*, vol. 172, pp. 358–361, 2020.

[24] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535–547, 2019.

[25] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117–128, 2011. DOI: 10.1109/TPAMI.2010.57.

[26] O. Jafari, P. Maurya, P. Nagarkar, K. M. Islam, and C. Crushev, "A survey on locality sensitive hashing algorithms and their applications," *arXiv preprint arXiv:2102.08942*, 2021.

[27] J. Briggs, "Nearest neighbor indexes for similarity search," [Online]. Available: https://www.pinecone.io/learn/vector-indexes/.

[28] J. Wang, T. Zhang, J. Song, N. Sebe, and H. T. Shen, "A survey on learning to hash," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 769–790, 2018. DOI: 10.1109/TPAMI.2017.2699960.

[29] J. Briggs, "Locality sensitive hashing (lsh): The illustrated guide," [Online]. Available: https://www.pinecone.io/learn/locality-sensitive-hashing/.

[30] ——, "Similarity search, product quantization 101," [Online]. Available: https://www.pinecone.io/learn/product-quantization/.

[31] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[32] R. Bos, L. Zheng, A. Hoogeveen, M. van Oort, L. Hoogland, and M. Korevaar, *Image-based video search engine*. [Online]. Available: https://github.com/aron-hoogeveen/ibvse.

[33] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[34] O. Team, *Opencv: Image file reading and writing*. [Online]. Available: https://docs.opencv.org/4.x/d4/da8/group__imgcodecs.html.

[35] A. Clark, *Pillow: Image file formats*. [Online]. Available: https://pillow.readthedocs.io/en/stable/handbook/image-file-formats.html.