Image-Based Video Search Engine

Feature Extraction

Aron Hoogeveen
and Max van Oort

EE3L11 - Bachelor Graduation Project
June 13, 2022

Group H2

TU Delft

Image-Based Video Search Engine

Feature Extraction

by

K.A. Hoogeveen and M.J.F. van Oort

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended publicly on Monday June 20, 2022 at 11:00 AM.

Student number: 4550897, 5113164
Project duration: April 19, 2022 – June 24, 2022
Thesis committee:

Prof. dr. N. Llombart,	TU Delft, jury chair
Dr. ir. J. Dauwels,	TU Delft, supervisor/ jury member 1
MSc L. Pakula,	TU Delft, jury member 2

TU Delft

Abstract

One of the main problems with Instance-Level Image Retrieval in video data is that for query videos with multiple objects of the same instance, extracting features from keyframes of this query video is time consuming. This thesis aims to solve this problem by implementing a Convolutional Neural Network based approach, which significantly reduces the extraction time and increases the accuracy. After analysing multiple methods, Second-Order Loss and Attention for image Retrieval (SOLAR) was found to be the most promising method. SOLAR will be tested based on three performance metrics: the mean average precision, the recall and the extraction time per image. The performance will be evaluated based on a selection of videos and images from a dataset provided by Dr. Andrea Nanetti from the Engineering Historical Memory project. For this dataset SOLAR achieved a mean average precision of 83 %, a recall of 85 % and an extraction time of 0.73 seconds per image. To conclude, SOLAR is specialised in detecting and describing landmarks due to its pre-trained model, but for a more general case the backbone model should be trained differently which will increase the accuracy. Future work could also include speed improvement by looking at object detection methods.

Preface

This thesis was written as part of the Bachelor Graduation Project to conclude our Bachelor Electrical Engineering at the Delft University of Technology. We worked on the project for 10 weeks in the period from April to June 2022. The project was proposed by our supervisor Justin Dauwels. Originally, the project was named “Image Search Engine for Digital History”, but it was subsequently changed to its current state during the second week of our project. The name of the project was then changed to “Image-Based Video Search Engine”.

We would like to thank our supervisor Justin Dauwels, our 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. We have always enjoyed working together as a team.

K.A. Hoogeveen and M.J.F. van Oort
Delft, June 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.* [9]. 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 duration 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 [12], 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 [13] 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

Subdivision of the System

According to the 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), which translates the keyframes and the query image into feature vectors.
3. Data Compression & Nearest Neighbour Search (DCNNS) [2], 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 image(s).*
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 Feature Extraction module.

Feature Extraction

The second module is the Feature Extraction module. It focuses on subproblem 2 of the problem statement. In essence, the module consists of an algorithm that extracts the features of the image queries and of the keyframes that were obtained by the Keyframe Extraction module. The goal is to optimise this extraction by evaluating different methods and their strengths and weaknesses under varying conditions. These conditions include the resolution of the keyframes and the query images.

Document Structure

This thesis describes the Feature Extraction module. In Chapter 2 the Programme of Requirements is given. In this chapter both the mandatory and trade-off requirements for the whole system and for the Feature Extraction module are enumerated. In Chapter 3, recent surveys about the state-of-the-art methods are summarised and three methods are analysed in more detail. In Chapter 4 the design process is explained. This includes the explanation of the performance metrics, the evaluation of the three methods and the purpose of rescaling. In Chapter 5 the implementation and validation of the chosen method are explained. Furthermore, the implementation and the validation of the prototype of the complete system will be evaluated. In Chapter 6 the conclusion is presented and the results of both the Feature Extraction module and the complete video search engine are discussed.

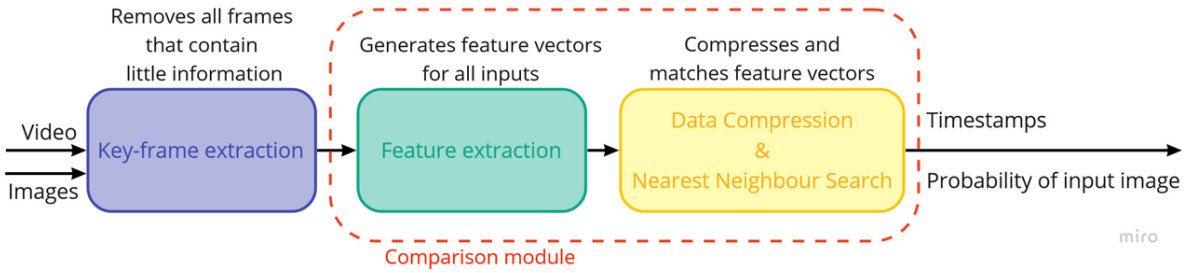


Figure 1.1: Pipeline of the complete Image-Based Video Search Engine

2

Programme of Requirements

The Programme of Requirements lists the restrictions and functionality of the Image-Based Video Search Engine. The requirements are divided in 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 when they are met. Both sections are divided in 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 be able to process a video shot at 30 frames per second in half the duration of the video.
11. Mean Average Precision must be at least 65 %.

Trade-off Requirements

12. The engine should be able to handle multiple query videos.
13. The engine should be able to handle multiple query 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.

PoR of Feature Extraction Module

The requirements below are made specifically for the Feature Extraction module. It is good to mention that these requirements hold for query images with a resolution of $1280 \times 720p$ and a minimal *GTX 1050 Mobile* GPU (4 GB of VRAM). With a higher resolution, the precision will most likely be higher but the execution time will also be higher. Furthermore, the execution time will be lower for a better GPU. That's why the requirements are based on two constants. The Key Performance Indicators (KPI) are mean Average Precision (mAP), recall and extraction time per image. The requirements below also include these KPI's.

Mandatory Requirements

Functional Requirements

1. The system must extract features from query images and keyframes.
2. The system must output feature descriptor vectors.

Non-Functional Requirements

3. Existing methods must be used to extract features from images.
4. In the license of the code of the existing methods it should state that the code is free to use for at least academic purposes.
5. The existing methods should make use of a pre-trained model.
6. The extraction time should be lower than or equal to 0.5 seconds per image.
7. The mAP should be equal to or higher than 80 %.
8. The recall should be equal to or higher than 80 %.
9. The performance of the system should be evaluated independent of the other modules.
10. The system should be tested with a diverse dataset.

Trade-Off Requirements

11. The extraction time per image should be as low as possible.
12. The resolution of the images should be as high as possible (to prevent information loss).
13. The map should be as high as possible.
14. The recall should be as high as possible.

3

Analysis

Earlier image matching approaches focused on handcrafted feature descriptors [14] [15]. Currently, state-of-the-art models use Convolutional Neural Networks (CNN) for extracting features, since these networks achieve much higher performance than the classical handcrafted approaches [16]. These are networks that learn the features themselves through deep learning, opposed to using handcrafted features.

Although most state-of-the-art methods use CNN for extracting features [17] [18] [19] this does not mean that handcrafted features have no value in current research. CNN models work better when the dataset they are trained on becomes larger [20]. In cases where the available data is limited it makes sense to use handcrafted features for better performance [21].

There are multiple ways to perform feature extraction. Aly *et al.* [22] clearly explain the two main ways. One way is to extract features at a global level: a single feature vector is extracted for the whole image. The benefit of this approach is that it is relatively fast, since you do not regard the contents of the image, but look at it as a whole. The other way is local feature extraction. With local feature extraction, keypoints are computed around which local feature vectors are extracted. This is in general slower, since you first need to apply an algorithm to determine the keypoints, after which you extract feature vectors for each of those keypoints. The latter method is more accurate at a cost of execution time.

This paper focuses on a deep learning approach for feature extraction, because it outperforms the handcrafted approach as described above. Furthermore, the amount of data will be large enough for a Image-Based Video Search Engine use case. Three methods based on a CNN backbone are described in the next sections. These methods are Detect-then-Describe Network (D2-Net), Local Feature TRansform (LoFTR) and Second-Order Loss and Attention for image Retrieval (SOLAR). D2-Net and LoFTR are both local feature extractors and SOLAR is a global feature extractor. For each method a general definition and the advantages will be given. The advantages will be illustrated with a figure that visualises the advantage.

3.1. D2-Net

D2-Net is a trainable CNN for joint description and detection of local features [23]. This method is designed to work under challenging conditions. Most methods work with a *detect-then-describe* approach, but D2-Net uses a *describe-and-detect* approach. Figure 3.1 visualises the comparison between the two different approaches. Part (a) of this figure shows different variants of the two-stage detect-then-describe approach. Part (b) shows the D2 approach which uses a single CNN. This means that the set of feature maps are first computed via a Deep Convolutional Neural Network (CNN). From these feature maps the descriptors are computed and the keypoints are detected. In the paper it is stated that this approach requires significantly less memory than dense methods. D2-Net also performs comparably well under challenging conditions such as day-night illumination changes and weakly textured scenes.

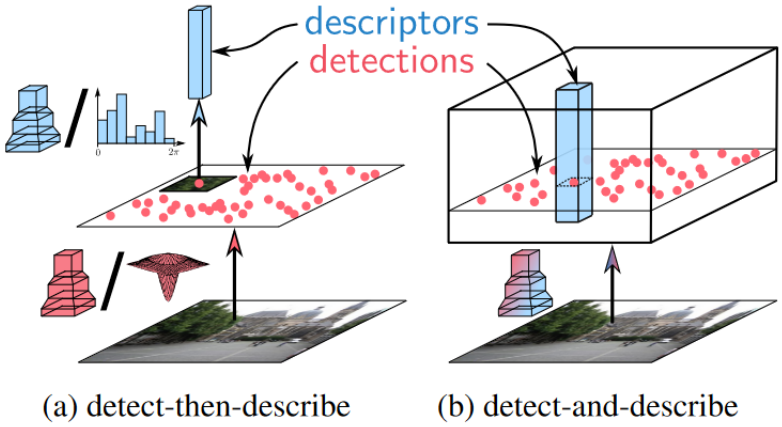


Figure 3.1: Comparison between different approaches for feature detection and description. Adapted from [23].

3.2. LoFTR

LoFTR is a detector-free local image feature matching method [24]. In contrast to other methods performing image feature detection, description and matching sequentially, LoFTR first establishes dense matches at a coarse level and later refines the good matches at a fine level. The advantage of LoFTR to state-of-the-art methods is that LoFTR is detector-free which will result in a higher performance at low-texture areas. Figure 3.2 shows the effect of a detector-free method. LoFTR is compared to SuperGlue [25] which is a neural network that matches two sets of local features by jointly finding correspondences and rejecting non-matchable points. It can be seen that LoFTR is capable of finding correspondences on the texture-less wall and the floor with repetitive patterns, where detector-based methods struggle to find repeatable interest points.

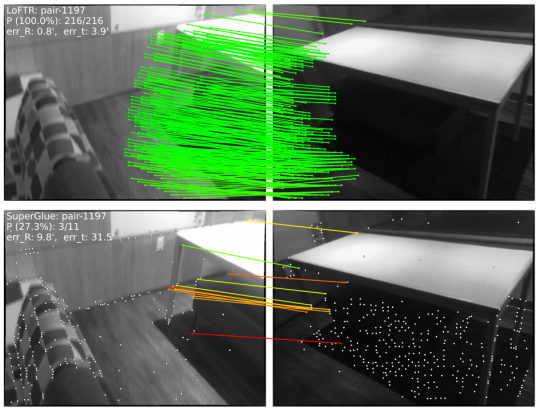


Figure 3.2: Comparison between the proposed method LoFTR and the detector-based method SuperGlue. Adapted from [24].

3.3. SOLAR

SOLAR is a global descriptor that applies second-order information through both spatial attention and descriptor similarity for image retrieval on a large scale [19]. As the name SOLAR (Second Order Loss and Attention for image Retrieval) suggests, it combines second-order spatial attention in descriptor learning with second-order descriptor loss for learning global image representation for retrieval.

Figure 3.3 shows an illustration of how SOLAR in general works. On the left it can be seen that the optimal relative feature contribution is learned spatially. The colours of the stars correspond to the borders of the smaller frames showing the attention for that location. The right part of the figure shows that SOLAR uses second-order similarity in the descriptor space to make the distance between clusters consistent.

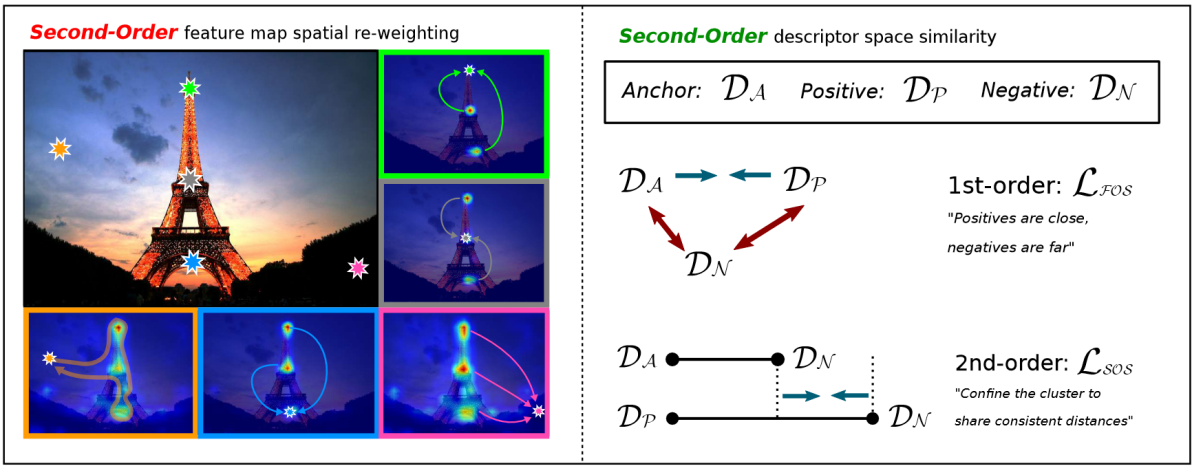


Figure 3.3: Illustration of SOLAR. Adapted from [19].

4

Design Process

4.1. Performance Metrics

To evaluate the performance of the different extraction methods, multiple performance metrics are used. One of the performance metrics is called mean Average Precision (mAP). The mAP is a popular metric mostly used to measure the performance of models which perform object detection tasks [26]. For object detection tasks the mAP is calculated in a way which can not be used for content matching tasks. However, to fulfill module requirement 7 a different formula can be used to measure the mAP. The general formula to evaluate different extraction methods can be seen below:

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i \tag{4.1}$$

where N is the number of iterations the system has been executed and AP_i is the average precision. The average precision for content matching is described by the following formula [27]:

$$AP_i = \frac{1}{GTP}\sum_{k}^{n} P@k \times rel@k \tag{4.2}$$

where GTP refers to the total number of ground truth positives, n refers to the total number of frames you are interested in, $P@k$ refers to the precision@k and $rel@k$ is the relevance function at k. k gives the rank in the resulting data, so $P@k$ means the precision at rank k for example. The relevance function equals either one or zero. $rel@k = 1$ if the frame at rank k is relevant and $rel@k = 0$ otherwise. The precision is then described by the following formula:

$$P@k = \frac{relevant\ frames@k}{retrieved\ frames@k} \tag{4.3}$$

where $relevant\ frames@k$ is the number of ground truth positives at rank k and $retrieved\ frames@k$ is the number of evaluated frames at rank k.

When measuring the performance of an extraction method, two scenarios can be taken into account. The first scenario is when the method should extract features of one query image and the keyframes of one query video. In that case it can be assumed that the mAP is equal to the AP, because the method extract the same features during each run. This means that it would be unnecessary to measure the average precision on multiple runs, because it would give the same result as with one run. The second scenario is when the method should extract features of multiple query images and the keyframes of one query video. In this case the mAP is not equal to the AP, because the method will extract different

features from each query image. For each query image an average precision will be calculated and the mAP is then calculated using Equation 4.1.

The average precision is not the only metric to evaluate the performance. The average precision is used to measure the quality of the ranking of the relevant frames, but the quantity of the relevant frames within the total number of ground truth positives is also important. To measure the quantity and to fulfill module requirement 8, the recall is used [28]. In the paper the recall is described as the ability of a model to find all relevant cases. It is the percentage of correct positive predictions among all given ground truths. This recall can be calculated with the following formula:

$$R@k = \frac{relevant\ frames@k}{GTP} \tag{4.4}$$

To illustrate how the AP and the recall are calculated, a possible scenario will be given. Imagine that one wants to search for a formula 1 car in a specific video. First the keyframes will be extracted from the video and these frames will be labeled and stored in a set G. How the labeling is done will be discussed in Subsection 5.1. Figure 4.1 shows the user query image and the labeled keyframes. In this case a Ferrari is searched for and the key frames contain both Ferrari and Red Bull cars.

Figure 4.1: The user query Q and the labeled set of key frames G.

After the keyframe extraction the features of the user query Q (the formula 1 car) and the keyframes will be extracted. Lastly, the features are matched and the result will be stored in a set G'. Figure 4.2 shows the sorted key frames for this specific case. It can be seen that the first, second and fourth frame are relevant frames and that the third frame is not relevant. Using Equation 4.3 the precision of the ordered set of key frames can be calculated at a certain rank. The recall can also be calculated using Equation 4.4. The results of these calculations can be seen in Table 4.1.

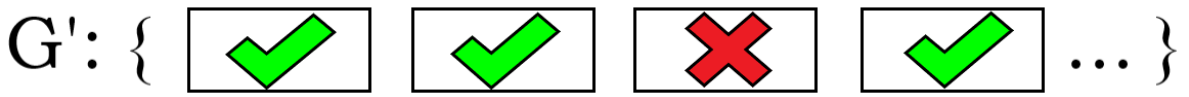


Figure 4.2: The sorted key frames G'.

Table 4.1: Precision and recall at rank k.

k	1	2	3	4
P@k	1	1	0.67	0.75
R@k	0.33	0.67	0.67	1

With the help of the precision, the average precision can be calculated. The average precision can be computed using Equation 4.2. It can be assumed that the total number of ground truth positives (GTP) is three and the reviewed number of frames (n) is equal to the amount of GTP's. Then the resulting average precision (which is equal to the mean average precision) for this scenario is calculated as

follows:

$$AP@3 = \frac{1}{3}\sum_{1}^{3} P@k \times rel@k = \frac{1}{3}((1 \times 1) + (1 \times 1) + (0.67 \times 0)) = 0.67$$

The average precision and recall will be used in the next section to evaluate the performance of three different methods to extract features. However, the precision and recall are not the only important aspects of the feature extraction part. The third important part of the performance is the execution time. This is also stated in module requirement 6 of the Programme of Requirements, so time, precision and recall will determine which method is the most promising for the final design.

4.2. Evaluated Methods

As described in Chapter 3, a lot of research is done on feature extraction. To fulfill the Programme of Requirements, not all methods that have been tested in the research papers can be used. According to module requirement 3 and as a result of system requirement 6, the method should be publicly available at GitHub, because it is not realistic to design a completely new method in a period of 10 weeks. Unfortunately, most recent methods are not publicly available so these methods are not evaluated. Furthermore, in the license of the methods it should be stated that the method can be used for research purposes to fulfill module requirement 4. Another thing to consider is that the available code should be written in Python, to fulfill system requirement 7.

Considering the requirements, three methods were chosen to evaluate the performance. These methods are D2-Net, Local Feature Matching with Transformers (LoFTR) and Second-Order Loss and Attention for Image Retrieval (SOLAR). For each method, a basic implementation was made. Some methods were easier to implement than others. LoFTR is both a feature extraction and feature matching method, so it would take more time to make a basic implementation using only the feature extraction part.

The three methods are evaluated based on the performance metrics described in the previous chapter, so the average precision, recall and the execution time will be measured. The three methods will be used to extract features from one query video and three query images. The query video and the query images are selected from the dataset which will be described in Section 5.1. The query video has a length of 4 minutes and 21 seconds and a resolution of $1280 \times 720p$. From the query video 2 frames per second will be extracted, so in total 524 frames per video. These frames will be labeled by hand as relevant or not relevant based on the object in the query image. How the labeling is done, will be explained in Subsection 5.1. The query images also have a resolution of $1280 \times 720p$. Furthermore, a linear comparison is used to match the feature vectors from the frames of the video and the query image. A linear comparison is used, because it has a precision of 1 and it can be easily implemented. Now, the methods are also evaluated independent of the other modules of the whole system, which was needed to fulfill module requirement 9.

D2-Net

Figure 4.3 shows the pipeline of the D2 network. A CNN (F) is used to extract feature maps from an input image (I). The CNN plays two roles. First, to create the local descriptors d_{ij} which are obtained by passing through all the n feature maps (D^k) at a spatial location (i, j). Second, to obtain the detections by performing a non-local-maximum suppression on a feature map followed by a non-maximum suppression across each descriptor. The right part of the figure shows the soft detection module which is used for training. This is not applicable for this project, because pre-trained models are used to fulfill module requirement 5.

In comparison to Quality Aware Template Matching (QATM) [29], Autoencoders [30] and Siamese network [31], D2-Net showed the most promising test results [6]. This was stated in the thesis of the previous Bachelor Graduation Project (BGP) group (2021). The test results showed that the precision of D2-Net was 1.00 and that the extraction time was 3.1 seconds per image. A precision of 1.00 is pretty impressive, but an extraction time of 3.1 seconds per image is not fast enough. However, a basic implementation was still made to test it with a different dataset. This was done using a part of the code of the paper which was publicly available at GitHub [32].

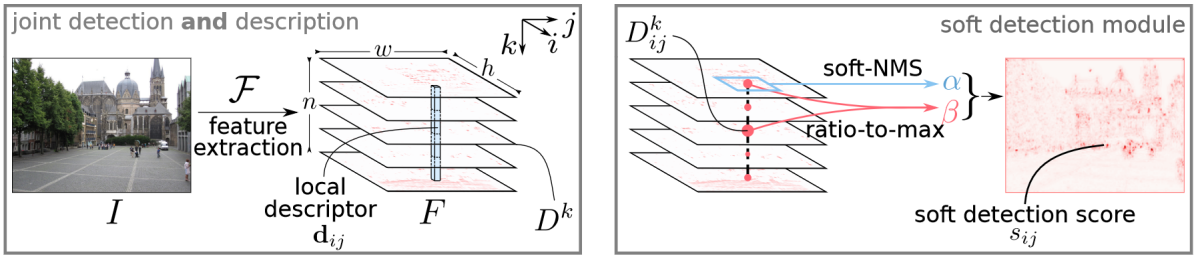


Figure 4.3: The pipeline of the detect-and-describe (D2) network. Adapted from [23].

After the basic implementation was made, the method was tested. Unfortunately, the linear comparison could not be performed because the output of the D2-Net method is a feature descriptor vector of size $N \times 512$, where N is the number of detected keypoints. Matching multiple descriptor vectors of this size with a linear comparison, will take a lot of time. For this reason it was assumed that the precision would not be much different from the results of the previous BGP group. The execution time could be measured without the linear comparison, but that would not be consistent because the precision and recall are measured at different hardware. As a result, the extraction time was also used from the previous BGP group. This means that the precision will fulfill the requirements, but the execution time will not meet the requirements.

LoFTR

Figure 4.4 shows the pipeline of LoFTR. It can be seen that LoFTR consists of four components. The first component of LoFTR is a local feature CNN that extracts the coarse-level feature maps $\tilde{F}^A$ and $\tilde{F}^B$ from an input image pair I^A and I^B. From this image pair the fine-level feature maps $\hat{F}^A$ and $\hat{F}^B$ are also extracted. The second component of LoFTR is the coarse-level feature transform. This component flattens the coarse feature maps to 1-D vectors and these vectors are added with the positional encoding. The positional encoding is derived from DEtection TRansformer (DETR) [33]. The sum of the vectors and the positional encoding is then processed by the LoFTR module. In this module the self- and cross-attention layers are interleaved by N_c times. The results of the LoFTR module are the transformed feature maps $\tilde{F}^A_{tr}$ and $\tilde{F}^B_{tr}$. The last two components are the matching module and the coarse-to-fine module. These two modules are not considered, because the matching of the feature vectors is not part of the feature extraction module.

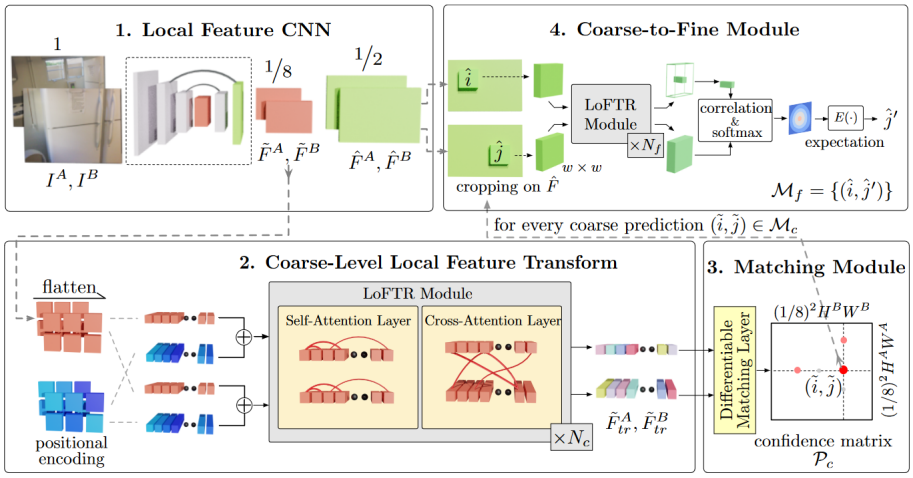


Figure 4.4: The pipeline of LoFTR. Adapted from [24].

The original idea was to make a basic implementation with the help of the codebase at GitHub from LoFTR [34], but this was not possible. The problem was that LoFTR performs both feature extraction and feature matching and that the feature extraction part could not be used as an independent module. In the paper of LoFTR it was stated that LoFTR outperforms D2-Net with five times more matches

between an image pair. LoFTR also outperforms D2-Net based on another performance metric called Area under the ROC Curve (AUC) [35]. The ROC curve plots the True Positive Rate (TPR) versus the False Positive Rate (FPR) at different classification thresholds. The AUC is a value between 0 and 1 (or 0 and 100 %) and the higher the AUC is, the better the model is at predicting no matches as 0 and matches as 1. The AUC at a threshold of 10 pixels is 53.6 % for D2-Net and 84.6 % for LoFTR [24].

SOLAR

Figure 4.5 shows the pipeline of the global descriptor SOLAR. It can be seen that a number of Second-Order Attention (SOA) blocks are inserted at different levels of a CNN backbone. This will result in a re-weighted feature map which is then followed by Generalised-Mean (GeM) pooling [36], whitening [37] and l_2 descriptor normalisation [38]. The SOA block works as follows: first three projections of feature map **f** named *query* **q**, *key* **k** and *value* **v** are generated through 1×1 convolutions. Then these tensors are flattened to obtain the shape $d \times hw$, where h, w and d are height, width and feature dimensionality, respectively. The next step is to compute the attention map **z**:

$$\mathbf{z} = \text{softmax}(\alpha \cdot q^T k) \tag{4.5}$$

where α is a scaling factor and **z** has a shape of $hw \times hw$. The last part of the SOA block is to compute the second order feature map f^{so}:

$$f^{so} = \mathbf{f} + \psi(\mathbf{z} \times \mathbf{v}) \tag{4.6}$$

where **f** is the first-order feature map and ψ is another 1×1 convolution to make sure that it has the same size as **f** $(D \times H \times W)$.

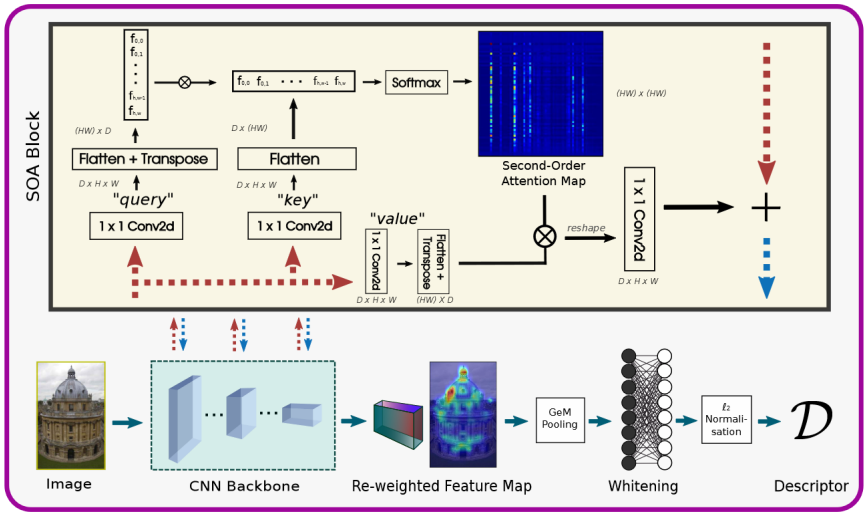


Figure 4.5: The pipeline of SOLAR. Adapted from [19].

Using the code of SOLAR from GitHub [39] [40], a feature extractor was made. The available code provided both a global and a local feature extractor, but only the global variant was considered. The reason behind this is that it was known from the previous two evaluated methods that a local feature extraction will take too long. After implementing the global feature extractor, the system was tested. The implementation was tested using the pre-trained Residual network (Resnet) 101 model [41], which was also available on the GitHub of SOLAR. The tests with this CNN showed that SOLAR is the most promising method. In comparison to D2-Net, the average precision is a bit lower but the execution time is much lower.

Conclusion

After evaluating the three methods above, it can be concluded that SOLAR is the most promising method to use for extraction of the features. It was observed that SOLAR outperforms the other two

(a) The original query image.

(b) The cropped query image.

Figure 4.6: The query images.

methods on time. It has to be noted that the execution time of LoFTR was not mentioned in the paper, but the assumption was made that a global feature extractor is always faster than a local feature extractor. The comparison between SOLAR and D2-Net can be seen in Table 4.3. The average precision and recall of SOLAR are calculated at the number of ground truth positives. The number of GTP's for each query image are shown in Table 4.2. The mAP and the mean Recall of SOLAR in Table 4.3 are obtained by taking the mean of the three different AP's and Recalls from Table 4.2. The table shows that the extraction time for SOLAR is 1.8 times faster than D2-Net. The mean average precision of D2-Net is not much higher than SOLAR, so these results show that SOLAR is indeed the best method to work with. The only disadvantage with using SOLAR is that it uses a pre-trained Resnet-101 model as backbone CNN, because this model is pre-trained on landmarks. The purpose of this project is to not only look at landmarks but also look at daily objects such as clothes or specifically the type of brand of clothes. In general, a lot of backbone CNN's are pre-trained on landmarks and can not be used for a general case. For this project the pre-trained Resnet-101 CNN will still be used, because there was not found a better alternative. This is also due to the time constraint of 8 weeks.

The limitation of the type of dataset the network was trained on came forward in early tests. We ran our first prototype on the images from Appendix A.2 using Figure 4.6a as query image. Surprisingly, in the top-5 results there was only one image in which the query object was slightly present. On the other images only a building was present. This result is explained if you look closer at the query image in Figure 4.6a. In the top-left and top-right of the image that same building is visible. Since the network has been trained on buildings and landmarks, it extracted most of its features from those small parts of the image. After cropping the query image (Figure 4.6b), such that the building parts were no longer visible, the returned results (Figure 4.8) were like what was initially expected.

Table 4.2: Average Precision and Recall at the number of GTP's for each query image at $1280 \times 720p$ for SOLAR.

Query image	1	2	3
GTP	79	4	27
AP@GTP	0.95	0.69	0.93
Recall@GTP	0.95	0.75	0.93

4.3. Rescaling

During testing of the three methods mentioned above, the same resolution as the query video and image was used ($1280 \times 720p$). However, changing the resolution before extracting the features can reduce the extraction time. On the other hand, changing the resolution will also change the precision of

Figure 4.7: The 5 best matching images of run with query image 4.6a (best match top-left, worst best match bottom-right).

Figure 4.8: The 5 best matching images of run with query image 4.6b (best match top-left, worst best match bottom-right).

Table 4.3: Test results of evaluated methods. Extraction time is measured in seconds per image.

Methods	D2-Net	SOLAR
mean Average Precision	0.96[1]	0.85
mean Recall	0.44[1]	0.88
Extraction Time (s)	2.0[1]	1.1

[1] The mAP and the recall are used from the previous BGP group [6].

the system. To determine what the ideal trade-off between the resolution and the precision is, multiple tests were done using the basic implementation of SOLAR. The purpose of the first test was to see what the effect of the resolution is on the extraction time. For this test multiple common resolutions [42] for the query image and the video frames were tested. These resolutions are shown in Table 4.4. Figure 4.9 shows the time versus resolution plot for 50 randomly generated images. Randomly generated data was chosen, because the content of the images do not affect the extraction time, only the resolution does affect the extraction time. It can be seen that the time increased as the resolution also increases.

Table 4.4: Common resolutions for 16:9 ratio. Adapted from [42].

Width (px)	256	320	426	640	848	960	1024	1280
Height (px)	144	180	240	360	480	540	576	720

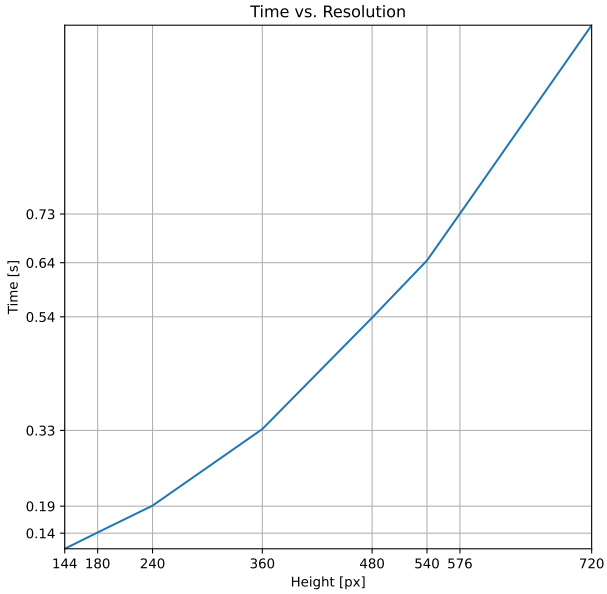
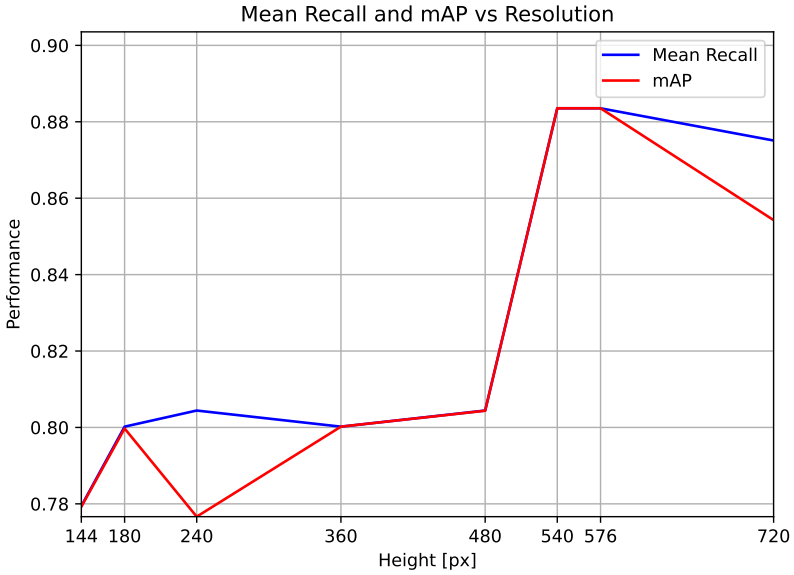


Figure 4.9: Time vs. resolution plot using SOLAR.

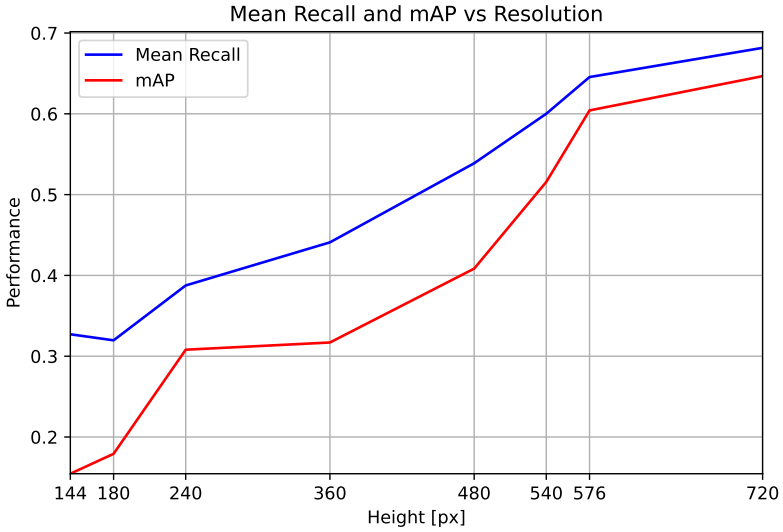
Now the effect of rescaling the resolution on the extraction time is known, the effect of rescaling the resolution on the average precision and recall will be evaluated. Figure 4.10 shows the mean Recall and mAP versus Resolution plot. The mean Recall and mAP are obtained by taking the mean of the average precision and recall for each resolution from Table 4.4 and for each query image from Table 4.2. It can be seen that the performance increases for higher resolutions. It has to be noted that these tests are performed on a relative 'easy' dataset and for that reason the performance is still pretty high for small resolutions.

Eventually, by comparing Figure 4.9 and Figure 4.10 a resolution of $1024 \times 576p$ was chosen. This resolution was chosen because the performance is still high enough at that point. For this specific case

the performance at a resolution of 960×540 is equal to the performance at a resolution of $1024 \times 576p$, but it was assumed that the performance did not decrease between these resolutions because it is an 'easy' dataset. In short, a resolution of $1024 \times 576p$ was chosen because this resolution has a higher probability of giving a better performance than a resolution lower. This means that the extraction time per image will be 0.73 seconds, which can be seen in Figure 4.9.



(a) 'Easy' dataset.



(b) 'Hard' dataset.

Figure 4.10: Mean recall and mAP vs. resolution using SOLAR.

5

Prototype implementation and validation

In this chapter the prototype implementation and validation of the whole system and of the feature extraction module will be explained. Firstly, the dataset will be discussed. This dataset is used to validate the prototype implementation. Secondly, as described in the previous chapter, SOLAR was chosen as the best method and will be described in detail below. The performance of SOLAR on the dataset will also be shown. Lastly, the prototype of the complete system will be described and the results will be shown. The Python code for the implementation of the whole system and the feature extractor can be found on GitHub [43].

5.1. Dataset

To validate the implementations of SOLAR and the complete system, a dataset was made. This dataset is an 'easy' dataset and is obtained by selecting 10 videos from an existing dataset provided by Dr. Andrea Nanetti from the Engineering Historical Memory project. The dataset from Nanetti consists of 52 videos of different lengths. The videos are about three historic travellers: Marco Polo, Ibn Battuta and Zheng He. For each video there are multiple query images available in another dataset. These query images are directly extracted from the videos, so the query image literally appears in the video. That is why the selection of 10 videos from this dataset is called 'easy'. The selected videos can be seen in Table 5.1. Eventually, a diverse dataset consisting of 10 videos of approximately three minutes with approximately 10 query images for each video can be used to validate the prototype implementation of the whole system and the SOLAR module. It has to be noted that a more difficult dataset is also been worked on. This dataset will contain multiple self-made videos of approximately one minute and will have one query image for each video. The object in the query image will appear in a different condition in the video. This means that the object in the video is hardly visible due to obstruction by another object for example. This dataset will contain random objects like the TU Delft logo or a coffee cup. Testing of the SOLAR module and the whole system on this 'hard' (and more realistic) dataset will also be done.

Table 5.1: Selection of query videos and query images for the 'easy' dataset.

Query video	1	2	3	4	5	6	7	8	9	10
Time (min:sec)	04:21	03:08	03:17	03:29	05:23	02:11	03:24	04:34	01:53	03:07
Traveller	Battuta	Battuta	Battuta	Battuta	Polo	Polo	Polo	He	He	He
Query images	10	10	10	5	10	10	10	10	7	10

Labeling

Labeling is needed to check whether the results of the system are correct or incorrect. Labeling means evaluating the (key)frames of a query video based on the target object in the query image. If the target object is visible in the frame then this frame will be labeled as a 1 and if the object is not visible then the frame will be labeled as a 0. This process will be done partly by hand and partly by the computer.

A script was created in Python which assigns the labels to the frames. The input of the script is the set of handpicked frames in which the object is visible and the output is the set of labeled frames. The script can be found on GitHub [43].

5.2. SOLAR Module

Implementation

To implement the SOLAR module in the whole Image-Based Video Search Engine, the inputs and outputs of the feature extraction module should be aligned with the other modules. For this reason the SOLAR module was implemented based on the following restrictions:

- The input should be two arrays. One array that contains the data of the query image(s) and one array that contains the data of the keyframes.
- The output should be two arrays. One array that contains the extracted features of the query image(s) and one array that contains the extracted features of the keyframes.

Validation

The implementation of SOLAR can be validated using the performance metrics explained in Section 4.1. The performance will be measured at six different videos from the selected dataset described in Section 5.1. Videos 1, 2, 5, 8, 9 and 10 will be used from Table 5.1. For half of the videos multiple query images will be used to measure the performance. For *He2*, *Polo1* and *Battuta2* one query image will be used. The query images used for the validation can be found in Appendix A.1. Just like the evaluation of the methods in Section 4.2, SOLAR will be validated independent of the other modules. That means that 2 frames per second will be extracted from the query videos instead of the keyframes. The features will then be matched with a linear comparison instead of a Nearest Neighbour Search. The results for SOLAR based on the AP and recall can be seen in Table 5.2.

The most results of AP@GTP and Recall@GTP are above 0.8 which was set as the target in module requirements 7 and 8 respectively, but the results of *Battuta1_4* and *He1_2* do not fulfill these requirements by a large margin. The reason of these small AP's and Recalls is that a lot of frames of the query videos were rated as Ground Truth Positive (GTP) by hand, while the the object was hardly visible in the frame. For a global feature extractor, such as SOLAR, finding an instance of an object in a video frame where the instance is relatively small in comparison to the size of the frame, the model will extract less features from the instance of the target object. That is why the performance is lower for these two cases.

The final results of SOLAR are shown in Table 5.3 and Table 5.4. These tables show the results of the Key Performance Indicators as described in Chapter 2. The mAP is calculated by using Equation 4.1 and the mean Recall is calculated by taking the mean of the Recall column of Table 5.2. Lastly, the extraction time per image is 0.73 seconds as explained in Section 4.3.

Table 5.2: Results of SOLAR at a resolution of $1024 \times 576p$. See Appendix A.1 for the used images.

Name	Extracted frames	GTP	AP@GTP	Recall@GTP
Battuta1_1	524	79	0.97	0.97
Battuta1_2	524	4	0.75	0.75
Battuta1_3	524	27	0.93	0.93
Battuta1_4	524	110	0.33	0.41
Battuta2_1	379	12	1	1
He1_1	550	11	1	1
He1_2	550	56	0.32	0.43
He1_3	550	102	0.74	0.82
He2_1	228	4	1	1
He3_1	377	124	0.98	0.98
He3_2	377	32	1	1
He3_3	377	11	1	1
Polo1_1	649	98	0.73	0.78

Table 5.3: Key Performance Indicators of SOLAR at a resolution of $1024 \times 576p$ of the 'easy' dataset.

Key Performance Indicators	
mAP	0.83
mean Recall	0.85
Extraction time [s]	0.73

Table 5.4: Key Performance Indicators of SOLAR at a resolution of $1024 \times 576p$ of the 'hard' dataset.

Key Performance Indicators	
mAP	0.60
mean Recall	0.65
Extraction time [s]	0.73

5.3. Image-Based Video Search Engine

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 develop/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.
- The input to the complete system consists of: a single query 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 5.1.

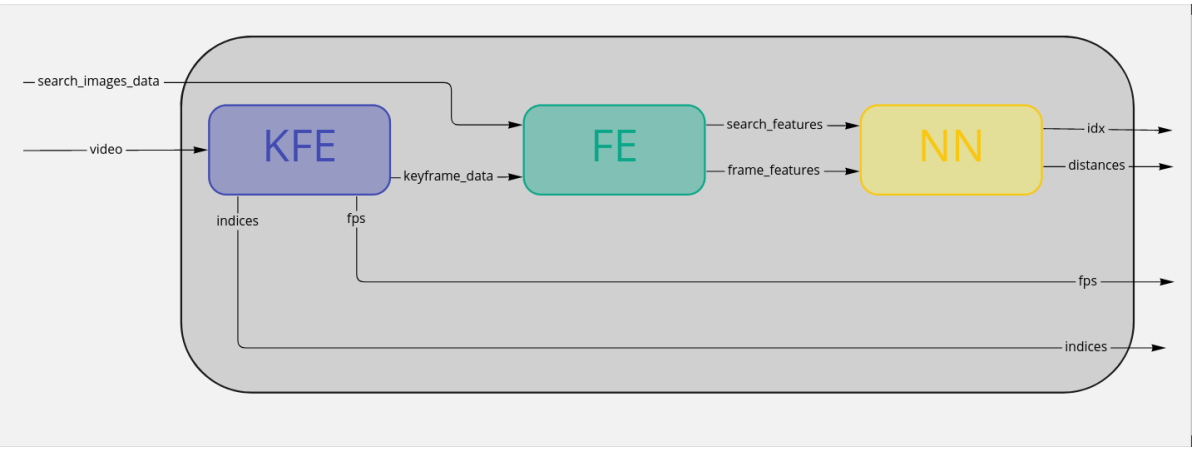


Figure 5.1: The in- and outputs of all the modules.

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 Python GUI will not be considered, so only the time measurement of the Python code 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 4.1.

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 according to the DCNNS module [2]), 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.

Validation

Timings
In Table 5.5 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 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.

Table 5.5: 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 in Section 4.3).

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**

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 5.3. The mAP calculations for the complete system can be seen in Table 5.6 and in Table 5.7. These tables show the results of both the 'easy' and the 'hard' dataset. Furthermore, the tables show the difference between using a filter and not using a 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 5.6: 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 5.7: 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. Feature Extraction module

In this paper different feature extraction methods have been discussed. This was done to tackle the problem defined in Chapter 1:

Develop an algorithm that can extract the features of the keyframes and of the query image(s).

The first step in tackling this problem was to enumerate the requirements of the system in Chapter 2. Next, an overview of the different methods to extract features was given in Chapter 3. After evaluating these methods in Chapter 4, SOLAR showed the best performance and was therefore used in the implementation (satisfying module requirement 3). Lastly, Chapter 5 showed the results of SOLAR on an 'easy' dataset.

When looking at the results of SOLAR it can be concluded that most of the requirements defined in the Programme of Requirements are met. With a mAP of 83 % and a mean recall of 85 % on the 'easy' dataset, module requirements 7 and 8 are both fulfilled respectively. Only an extraction time of 0.73 seconds per image was not enough to fulfill module requirement 6.

Discussion

In Section 4.2 it was already mentioned that the chosen backbone convolutional neural network plays a very important role in determining the performance of the system. Therefore, we recommend further research to focus, among other things, on the type of backbone that is used, so that it matches the use-case(s). If needed to further improve the performance, the network can be trained instead of using a pre-trained network.

For 'easy' datasets the system performs reasonably well. However, those 'easy' datasets do not reflect the real world. To improve the performance on datasets that better reflect the real world, we suggest that further research also dives deeper in using (fast) local feature extractors to better recognize different objects in query images that contain a lot more information. In combination with object detection this will most likely improve the accuracy of the system by a lot. However, care must be taken that the increase in processing time, that these modifications will definitely bring, does not outweigh the increase of accuracy.

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 5.3. Requirements 4,

5, 7 and 8 have been fulfilled and the results can be found on the GitHub [43]. 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 Chapter 2. Requirement 10 and by extension requirement 18 have been met as described in Section 5.3. Requirement 11 and by extension requirement 14 have been met which is shown in Chapter 5.

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 [43]. The structure of the code on GitHub also shows that Requirement 15 has been fulfilled. Requirements 16 and 17 are based on the video and image reading libraries used, which support a variety of formats. These are OpenCV [44] and Pillow (PIL) [45] respectively and the supported formats can be found in their respective documentations. Requirement 19 has been tested. As a result, the system works for videos of durations up to 1 hour and 20 minutes.

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].

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.

A

Images and Videos

A.1. ‘Easy’ Dataset Images

Figure A.1 contains the images that were used for testing the system. The images are extracted from videos provided by Dr. Andrea Nanetti. Table. A.1 contains the links to the corresponding videos.

Video Name	Link
Battuta 1	link
Battuta 2	link
He 1	link
He 2	link
He 3	link
Polo 1	link

Table A.1: Links to the videos used for determining the performance of the system.

(a) Batutta1_1

(b) Battuta1_2

(c) Battuta1_3

(d) Battuta1_4

(e) Batutta2_1

(f) He1_1

(g) He1_2

(h) He1_3

(i) He2_1

(j) He3_1

(k) He3_2

(l) He3_3

(m) Polo1_1

Figure A.1: ‘Easy’ Dataset query images.

A.2. Mekelpark Video

On the next page the extracted frames can be found from the video we used for our first prototype test. The frames were uniformly extracted at 2 fps using the following script that is available on our Github (link).

Bibliography

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

[2] L. Hoogland and M. Korevaar, “Image-based video search engine: Data compression and nearest neighbour search.”

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

[4] I. E. Lager, V. Scholten, E. Bol, C. Richie, and S. I. with special thanks to Koen Bertels, “Bachelor graduation project manual,” 2022.

[5] M. Deutman, P. Groet, and O. van Hooff, “Image search engine for digital history, a standard approach,” 2021. [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,” 2021. [Online]. Available: http://resolver.tudelft.nl/uuid:f1a2902b-14be-416c-ae1a-ce4f179a0425

[7] Y. Yao, “[unpublished manuscript],” Available at https://github.com/YYao-42/ImgSearch.

[8] W. Chen, Y. Liu, W. Wang, E. M. Bakker, T. Georgiou, P. Fieguth, L. Liu, and M. Lew, “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. [Online]. Available: https://www.mdpi.com/2220-9964/9/5/297

[10] A. Araujo, M. Makar, V. Chandrasekhar, D. Chen, S. Tsai, H. Chen, R. Angst, and B. Girod, “Efficient video search using image queries,” in *2014 IEEE International Conference on Image Processing (ICIP)*, 2014, pp. 3082–3086.

[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, pp. 1470–1477 vol.2.

[12] 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., vol. 5018, International Society for Optics and Photonics. SPIE, 2003, pp. 123 – 130. [Online]. Available: https://doi.org/10.1117/12.476329

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

[14] J. Ma, X. Jiang, A. Fan, J. Jian, and J. Yan, “Image matching from handcrafted to deep features: A survey,” *International Journal of Computer Vision*, 2021.

[15] S. R. Dubey, “A decade survey of content based image retrieval using deep learning,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 32, no. 5, pp. 2687–2704, 2022.

[16] W. Chen, Y. Liu, W. Wang, E. Bakker, T. Georgiou, P. Fieguth, L. Liu, and M. S. Lew, “Deep learning for instance retrieval: A survey,” 2021. [Online]. Available: https://arxiv.org/abs/2101.11282

[17] F. Tan, J. Yuan, and V. Ordonez, “Instance-level image retrieval using reranking transformers,” in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 12 085–12 095.

[18] M. Yang, D. He, M. Fan, B. Shi, X. Xue, F. Li, E. Ding, and J. Huang, "DOLG: single-stage image retrieval with deep orthogonal fusion of local and global features," *CoRR*, vol. abs/2108.02927, 2021. [Online]. Available: https://arxiv.org/abs/2108.02927

[19] T. Ng, V. Balntas, Y. Tian, and K. Mikolajczyk, "Solar: Second-order loss and attention for image retrieval," in *Computer Vision – ECCV 2020*. Cham: Springer International Publishing, 2020, pp. 253–270.

[20] C. Luo, X. Li, L. Wang, J. He, D. Li, and J. Zhou, "How does the data set affect cnn-based image classification performance?" in *2018 5th International Conference on Systems and Informatics (ICSAI)*, 2018, pp. 361–366.

[21] W. Lin, K. Hasenstab, G. M. Cunha, and A. Schwartzman, "Comparison of handcrafted features and convolutional neural networks for liver mr image adequacy assessment," *Scientific Reports*, 2020.

[22] M. Aly, P. Welinder, M. Munich, and P. Perona, "Automatic discovery of image families: Global vs. local features," in *2009 16th IEEE International Conference on Image Processing (ICIP)*, 2009, pp. 777–780.

[23] M. Dusmanu, I. Rocco, T. Pajdla, M. Pollefeys, J. Sivic, A. Torii, and T. Sattler, "D2-net: A trainable cnn for joint description and detection of local features," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 8084–8093.

[24] J. Sun, Z. Shen, Y. Wang, H. Bao, and X. Zhou, "Loftr: Detector-free local feature matching with transformers," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 8918–8927.

[25] P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich, "Superglue: Learning feature matching with graph neural networks," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 4937–4946.

[26] S. Yohanandan, "map (mean average precision) might confuse you!" Available at https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2 (2020/06/09).

[27] R. J. Tan, "Breaking down mean average precision (map)," Available at https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52 (2019/03/24).

[28] R. Padilla, S. L. Netto, and E. A. B. da Silva, "A survey on performance metrics for object-detection algorithms," in *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, 2020, pp. 237–242.

[29] J. Cheng, Y. Wu, W. AbdAlmageed, and P. Natarajan, "Qatm: Quality-aware template matching for deep learning," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 11 545–11 554.

[30] I. A. Siradjuddin, W. A. Wardana, and M. K. Sophan, "Feature extraction using self-supervised convolutional autoencoder for content based image retrieval," in *2019 3rd International Conference on Informatics and Computational Sciences (ICICoS)*, 2019, pp. 1–5.

[31] I. Melekhov, J. Kannala, and E. Rahtu, "Siamese network features for image matching," in *2016 23rd International Conference on Pattern Recognition (ICPR)*, 2016, pp. 378–383.

[32] M. Dusmanu and Y. Cabon, "D2-net: A trainable cnn for joint detection and description of local features," Available at https://github.com/mihaidusmanu/d2-net (2021/07/26).

[33] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds. Cham: Springer International Publishing, 2020, pp. 213–229.

[34] J. Sun, Z. Shen, Y. Wang, H. Bao, and X. Zhou, “Loftr: Detector-free local feature matching with transformers,” Available at https://github.com/zju3dv/LoFTR (2022/02/14).

[35] A. P. Bradley, “The use of the area under the roc curve in the evaluation of machine learning algorithms,” *Pattern Recognition*, vol. 30, no. 7, pp. 1145–1159, 1997. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320396001422

[36] F. Radenović, G. Tolias, and O. Chum, “Fine-tuning cnn image retrieval with no human annotation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 7, pp. 1655–1668, 2019.

[37] S. Zafeiriou and N. Laskaris, “On the improvement of support vector techniques for clustering by means of whitening transform,” *IEEE Signal Processing Letters*, vol. 15, pp. 198–201, 2008.

[38] Z. Dai, W. Chen, X. Huang, B. Li, L. Zhu, L. He, Y. Guan, and H. Zhang, “Cnn descriptor improvement based on l2-normalization and feature pooling for patch classification,” in *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2018, pp. 144–149.

[39] T. Ng, “Solar: Second-order loss and attention for image retrieval,” Available at https://github.com/tonyngjichun/SOLAR (2021/05/28).

[40] F. Radenović, G. Tolias, and O. Chum, “Fine-tuning cnn image retrieval with no human annotation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 7, pp. 1655–1668, 2019.

[41] Z. Wu, C. Shen, and A. van den Hengel, “Wider or deeper: Revisiting the resnet model for visual recognition,” *Pattern Recognition*, vol. 90, pp. 119–133, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320319300135

[42] Š. Pilipović, “Complete list of 16:9 resolutions,” Available at https://levvvel.com/169-resolutions/ (2022/01/08).

[43] R. Bos, L. Zheng, A. Hoogeveen, M. van Oort, L. Hoogland, and M. Korevaar, “Image-based video search engine,” Available at https://github.com/aron-hoogeveen/ibvse (2022/06/13).

[44] “Opencv: Image file reading and writing.” [Online]. Available: https://docs.opencv.org/4.x/d4/da8/group__imgcodecs.html%7D

[45] A. Clark, “Pillow: Image file formats.” [Online]. Available: https://pillow.readthedocs.io/en/stable/handbook/image-file-formats.html%7D