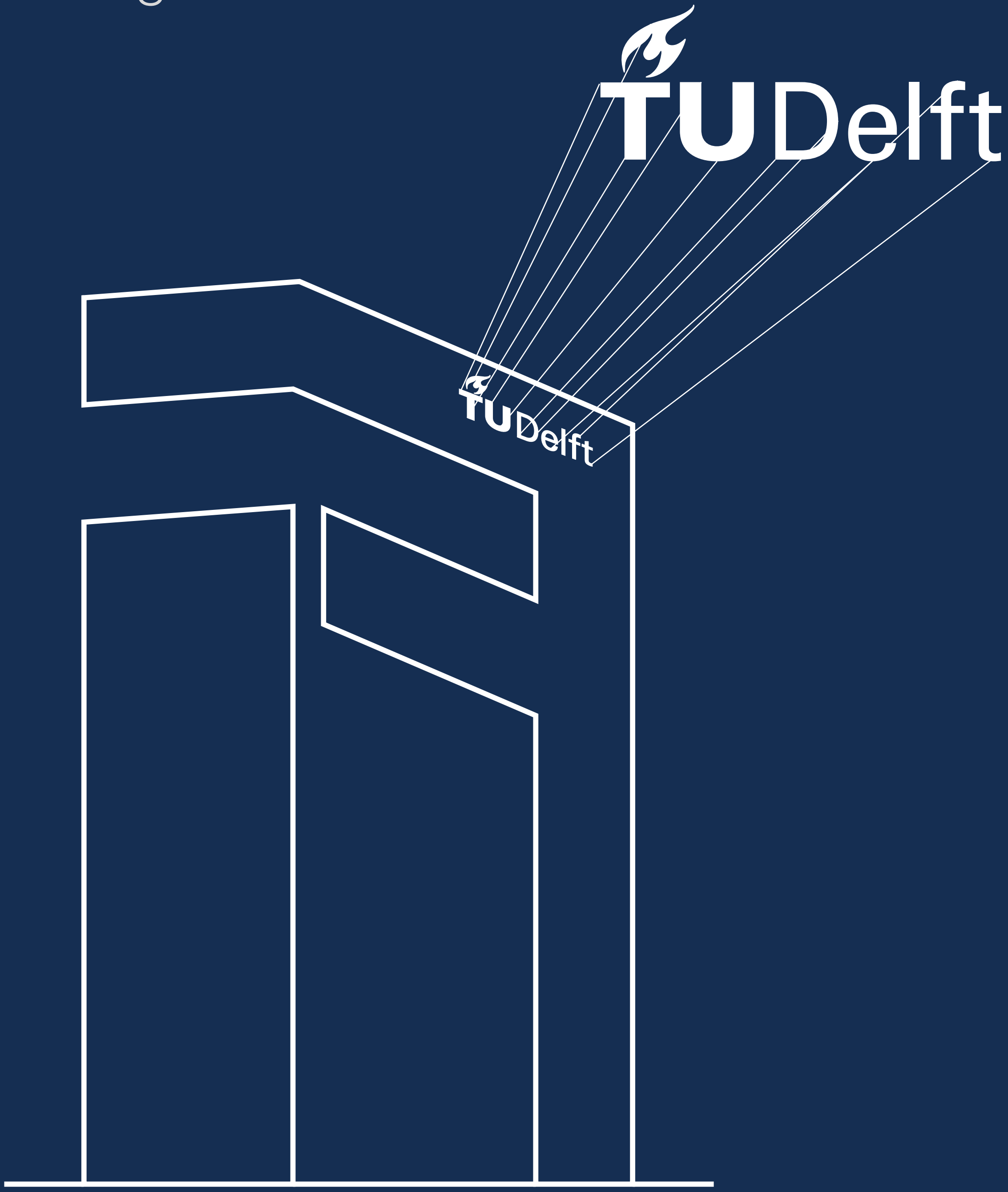
Image Search Engine for Digital History

A Deep Learning approach

Mathijs van Geerenstein, Philippe van Mastrigt, and Laurens Vergroesen



Engineering Historical Memory – Bachelor Thesis 2021

Image Search Engine for Digital History

A deep learning approach

by

M.R. van Geerenstein, P.G. van Mastrigt, and L. Vergroesen

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended on Wednesday June 30, 2021 at 11:00 AM.

Student numbers:	4598660, 4893298, 4448871	
Project duration:	April 19, 2021 – July 2, 2021	
Thesis committee:	Prof. dr. ir. A.H.M. Smets	TU Delft, Chair of the Jury
	Dr. ir. J. Dauwels	TU Delft, Supervisor
	Dr. S. Cotofana	TU Delft
	Dr. ir. S. Vollebregt	TU Delft

TU Delft

Abstract

This research investigates and describes an image search engine for digital history using deep learning technologies. It is part of the Engineering Historical Memory research, contributing to a multilingual and transcultural approach to decode-encode the treasure of human experience and transmit it to the next generation of world citizens. The engine provides a new way to search in online (historical) digital libraries using content-based image retrieval and makes linguistic metadata redundant. State-of-the-art deep learning methodologies in computer vision have been investigated and tested. These methodologies include both template-based matching and feature-based matching. A VGG-16 Convolutional Neural Network based approach, called D2-Net, is concluded to provide the best basis. D2-Net is then further analyzed, improved, and optimized to run on a large dataset of more than 12k image combinations related to history, heritage, and art. The final implementation shows promising results with a precision of 0.96 and a recall of 0.44 on a challenging testing dataset. Future improvements include speed improvement and model training.

Preface

This thesis is written to conclude our learnings of the Bachelor Electrical Engineering at the Delft University of Technology. The research was performed in the period April up until June of 2021. We are available to share our experience and answer any questions related to this project.

We would like to thank Ioan Lager for his assistance and feedback in the process of the project. We have really enjoyed this challenging project. Particularly its multidisciplinary aspect made it unique and rewarding. The artificial intelligence side, currently piloted in the bachelor curriculum, opens doors for new endeavours in the prospect of our careers. We look forward to the future endeavours by Delft University in this field of research.

Additionally, we would like to sincerely thank Justin Dauwels and Andrea Nanetti for their support in this project. It is an honour to have contributed to Engineering Historical Memory (EHM) and to obtain a better understanding of heritage science and computer vision. The weekly meetings with you have provided us with new energy and food for thought. In this fast-paced world, we are convinced that EHM provides the future levers for transmitting human experience and aggregation of knowledge in our digital age.

In the process of this research, we have made use of the High Performance Cluster of Delft University of Technology. We would like to thank Frans Broos for his support, enabling us to make use of fast and reliable hardware.

Lastly, this thesis has been created parallel to the standard approach thesis [1]. We have really enjoyed working together. Both the study and leisure activities have been wonderful.

M.R. van Geerenstein, P.G. van Mastrigt, and L. Vergroesen
Delft, June 2021

Contents

1 Introduction

This thesis, in a wider perspective, is part of an ongoing research project *Engineering Historical Memory* [2] aiming to develop and experiment aggregation apps for the (re)organisation and delivery of global historical knowledge in the digital age. Nanetti explains in his paper [3] the need for a multilingual and trans-cultural approach to decoding-encoding human experience and transmitting this to the next generations of humanity. Advances in information technologies such as artificial intelligence and machine learning algorithms can assist in (i) decoding knowledge and wisdom embedded in cultural artefacts and social rituals, (ii) encoding data in machine-readable systems, (iii) aggregating information according to the user's needs in real time, and (iv) simulating the effects of erasing, neglecting, preserving, and sharing human experiences.

To contribute to this need, an image search engine for digital history is developed. This thesis describes current state-of-the-art information technologies and includes the search engine design process. Section 1.1 covers the analysis and framing of the problem. Additionally, it includes the analysis and discussions necessary for scoping, bounding, and creating the Program of Requirements (PoR) described in Chapter 2. In Chapter 3, the design process is discussed, specifying all decisions made on the basis of the PoR. This is followed by Chapter 4 in which the implementation of the image search engine is laid out and discussed. Chapters 5 and 6 will include the prototype results, project wide conclusions, discussions, and recommendations that emerged from this research. Additional and intermediate analyses, relevant in the process of this project, are described in Appendix A. Code snippets and the GitHub can be found in Appendix B.

1.1. Problem definition

Defining the problem is crucial and requires the assessment of the context and scope. The predetermined project conditions are the ten weeks of available time, the group size consisting of six electrical engineering students, and the supervision by J. Dauwels and A. Nanetti. Additionally, some existing suggestions and code were provided at the start by the supervisors: a proposal document, a short description on template matching, and links to Python libraries.

To obtain a better understanding of the underlying factors and find potential levers for decision making, an inductive logic tree has been created. Based on this, the decision was made to investigate existing historical applications, object detection concepts, image extraction and matching methodologies, libraries, tools, and datasets.

1.1.1. Problem analysis

Looking into existing historical applications, relatively little was found on applying image search and retrieval in a historical context. This indicates research potential for developing an image search engine for digital history. The results found include the search of reproductions of art through visual attributes for historians [4] [5], detecting lost heritage in historical video material [6], word-image classification in historical document collections [7] [8] [9], indexing expert image collections specifically on heritage image datasets [10], the MARS (Multimedia Analysis and Retrieval System) used on images of ancient African artifacts from the Fowler Museum of Cultural History at UCLA [11], and lastly the search for artistic connections across cultures using image retrieval [12]. The specific application of using image retrieval for improving historical research is most similar to [4] and [12]. Particularly [12] is interesting, as it (i) finds pairs of semantically related artworks that span different cultures, media, and millennia, (ii) builds on and improves current approaches in image retrieval, and (iii) has been implemented online. However the algorithm distinguishes object media: objects may differ in material. Additionally, it uses filters based on human interpretation to structure content, which is not of interest.

Zooming in on object detection, extraction, and matching, several important ideas, methodologies, and concepts have been found. One of these is Content-Based Image Retrieval (CBIR): using color, shape and texture of one image to find similarities in other images [13]. Experiments were performed on huge image databases and major performances were obtained after involving neural networks [14]. Besides color, shape, and texture, images with variations in viewing angle should also be taken into account [15]. However, the solution proposed does not provide leverage because it requires labelling of data and object specification. In [16] a method is explained on how to retrieve objects from a large corpus, and resolves this through improving the visual vocabulary and incorporating spatial information into the ranking. This is an interesting approach to improve speed in large amounts of data. However, using a visual vocabulary, a collection of visual words which together can give information about the meaning of the image (or parts of it) [17], is out of scope. The 'image meaning' should not be involved in the engine, because it inherently puts human interpretation into the machine. This creates, although arguable, problems that this thesis considers out of scope. Some concerns are elaborated in the PoR in Section 2, and a more general ethical perspective on AI and search engines is elaborated in the documents on Ethics and Technology [18] [19]. Methodologies to perform object detection include both standard (handcrafted) approaches and deep learning approaches [14] [20] [21] [22].

Regarding image matching, two categories exist: template-based matching and feature-based matching. Template matching is a high-level machine vision technique that identifies parts of an image that match a predefined template pixel by pixel. Advanced template matching algorithms find occurrences of the template regardless of their orientation and local brightness. Feature based matching is used when both source and template images contain more correspondence with respect to features and control points [23]. Image features such as edges and interest points provide rich information on the image content. These features are unique for each image and hence, help in identifying between images. The features of an image are not affected by change in size and orientation and therefore suitable if images are transformed in some fashion. Additionally, this approach is more efficient to use if the image has a large resolution: moving a template image across a large source image, one pixel at a time, repeating it at different scales is computationally expensive [23].

Most of the code and libraries used for implementation are Python based. Libraries investigated related to Computer Vision and Deep Learning include OpenCV [24], PyTorch [25], PyDegensac [26] [27], and Scikit-image [28]. Existing methodologies investigated include Quality-Aware Template Matching [29], Autoencoders [30] [31], Convolutional Neural Networks [32] [33], Siamese Networks [34], and D2net [35].

1.1.2. Problem scoping and bounding

The necessity of this project lies in the search for visual content in the increasing amount of digital data. In a historical context Nanetti describes the need and a general approach [3]. The additionality of this project is the ability for historians and heritage stakeholders to find information and starting points for research through exploring visual content. Special attention is paid to avoiding text and 'image meaning' interpretation. Additionally, the risks of no such research are, hypothetically speaking, loss of information and access across the globe.

The first objective is developing an image search engine for digital history, capable of retrieving images from various databases (e.g. Wikipedia, Europeana) similar to the user input image. A second objective is to write theses that elaborate on the approaches and results of creating such an image search engine. Additionally, business and ethics considerations will be included in separate documents and presentations. The first objective is completed if the engine (i) extracts and matches images, (ii) can retrieve images from image databases, (iii) accounts for machine interpretation, and (iv) is not based on (textual) metadata. The second objective is completed according to the manual [36].

The constraints of this project are the 10 weeks of available time for both objectives and the use of only python related libraries or tools. Moreover, no object recognition and error feedback are implemented.

The parameters of the image search engine are the underlying thresholds and pre-trained models used by the algorithm. These can be used to improve or change the decision making of the algorithm on image similarity. Additional parameters are keypoints and matches detected by the algorithm. These (can) differ per methodology used. The parameters affect the key performance indicators: time, precision, recall, and (balanced) accuracy.

1.1.3. Problem statement

Based on the analysis, scoping, and bounding, the project has been split into investigating standard and deep learning methodologies. The following problem statement has been formulated for this report:

To develop an algorithm in ten weeks that compares image queries with other images purely based on image content in multiple formats using a deep learning approach.

2 Program of Requirements

The Program of Requirements (PoR) defines the functionality of the image search engine. It consists of key performance indicators (KPIs) and the conditions applying to the development and implementation of the deep learning image search engine for digital history. A distinction is made between mandatory requirements, trade-off requirements, functional requirements, and non-functional requirements. The full Program of Requirements is depicted in Figure 2.1.

Mandatory requirements
Functional requirements
1. The system must find and return images that match to a user input image
2. Image matching is based solely on image content
Non-functional requirements
3. The software must be written in the same language as the existing codebase of Engineering Historical Memory
4. The software must be able to be inserted in the existing codebase of Engineering Historical Memory
5. Out of all returned images, at least 80% must be true positives (precision > 0.8)
6. Out of all images that are supposed to be matches, at least 25% must be found (recall > 0.25)
7. Balanced accuracy must be at least 70%
8. The software implementation must make use of libraries and functions free for academic use
9. The system must accept the common image codecs jpg and png
10. The system must support image files up to 10MB in size
11. The software must allow for parallel computing
12. The full implementation must be completed within 10 weeks by a group of 6 students
13. The software must be able to be tested on hardware accessible to the group

Trade-off requirements
Functional requirements
14. The search time should be as low as possible
Non-functional requirements
15. Precision should be as high as possible
16. Recall should be as high as possible
17. Balanced accuracy should be as high as possible
18. The codebase should be structured clearly and documented in such a way that others can continue on our work
19. The search engine should not be biased (should not include user feedback to improve the performance)
20. The system should show how and why matches were found
21. The supported number of image formats should be as high as possible

Figure 2.1: General Program of Requirements

Its core functional requirements, shown as items 1 and 2, follow directly from the proposal document and supervisor discussions. The qualities and attributes follow from discussion and existing literature.

Items 3, 4 and 8 account for further research by other scholars. Items 5, 6, and 7 are important for performance measuring purposes. Items 8, 9, 10, and 11 limit the implementation, and items 12 and 13 specify the product-project relation. The trade-off requirements specify what is desired. Item 14 follows from item 1 and 2 in a end-user perspective. Items 15, 16, and 17 specify desired attributes for performance. Item 18 supports, again, further research and development. Items 19 and 20 specify the desire for transparency and understanding of underlying engine decisions. This is especially important when considering ethical concerns [18] [19]. Item 21 supports items 1, 2, and 9. Requirements specifically for a Deep Learning approach are shown in Figure 2.2. These items have been decided upon to limit the scope of the project and are based on literature study and supervisor discussions.

Deep learning
Functional requirements
22. The algorithm must make use of deep learning
Non-functional requirements
23. The algorithm must be trainable

Figure 2.2: Deep learning Program of Requirements

The requirements specified are focused on the performance, creation, efficiency, and product handling. Considerations about safety, environment and cost are not included in the scope of this thesis due to available time and resources. This directly implies that such considerations are open to research. Ethical considerations are discussed in separate documents [18] [19] and includes concerns about artificial intelligence and search engines. In the following chapters, the requirements are referred to using the (x) notation.

3 Design process

The requirements specified in the previous chapter can be satisfied in multiple ways. In the following sections the design process is elaborated, relating the PoR to the process of creating an image search engine. Firstly, the potential of Content-Based Image Retrieval and two image matching categories is explained. Secondly, the used pipeline for deep learning and measuring performance is described. Thirdly, the results and conclusions of testing various methodologies are discussed. Lastly, testing results are discussed and points of interest are concluded for the implementation and design of the prototype in Chapter 4.

3.1. Introduction to Content-Based Image Retrieval

Content-Based Image Retrieval (CBIR) is key to satisfying requirement 2, because it makes meta-data in image retrieval redundant [13]. The main difficulty of CBIR is relating low-level features to a higher semantic understanding, also known as the semantic gap [37]. The application of CBIR involves obtaining 'features' such as texture, color, shapes, and composition from images. These features are compared through extraction and matching. A typical approach is to use feature-based methodologies as described in [13]. An alternative method is a template-based approach [38]; a digital image processing technique for finding small parts of an image that match to a template image. Current state-of-the-art methodologies are based on deep learning [39].

3.2. Image matching

Based on the problem analysis in Section 1.1.1, the PoR, and supervisor discussions, both template-based and feature-based matching are investigated. Both are implemented either through standard approaches [1] or deep learning approaches.

3.2.1. Template-based matching

Template-based matching is an approach to CBIR in which a template (query) image is traced in other image(s) one pixel at a time [38] (see Figure 3.1). The general pipeline for template-based matching is (i) pre-processing, (ii) 2D convolution of the template image and the source image, and (iii) determine results based on the correlation score. Use cases for template-based matching include quality control in manufacturing [40], image-to-GPS verification [29], and image retrieval [41]. For that reason it has the potential to satisfy the PoR. Conventional template matching methods are computationally expensive due to the sampling of a large numbers of points. Additionally, standard approaches in template-based matching do not perform sufficiently if the template and source image are different in scale, luminescence, contrast, and viewpoint [42]. This, in combination with requirement 22 and 23, points towards researching faster and more novel approaches such as Quality-Aware Template Matching [29].

3.2.2. Feature-based matching

An alternative to template-based matching is feature-based matching (see Section 1.1.1 and Figure 3.2). The general pipeline of feature-based matching is (i) pre-processing (i.e. image decoding, conversion, and scaling), (ii) converting decoded images into feature vectors (extraction), and (iii) feature vector comparison between images. Feature-based matching applications include fingerprint recognition [43], sketch-to-photo matching [44], and medical image classification [45]. Next to that, feature-based approaches are used in state-of-the-art deep learning CBIR [46].

The content of such feature vectors describe either local or global features. Local features represent image patches and are obtained through detection and description of keypoints. Global features do not describe the image using keypoints, but rather as a whole. Global feature matching is generally less computationally expensive at the cost of matching performance [46]. In order to comply with performance requirements 5, 6 and 7, as well as the search time requirement 14, both global feature and local feature approaches are evaluated.

Global feature vectors are compared by directly comparing the values of their entries, which is often done with the euclidean or cosine distance. Local feature vectors contain a descriptor for each detector. Descriptors of different images are compared using a distance function, similar to how global feature vectors are compared. After matching individual descriptors, descriptor matches that are inliers are detected using RANSAC [26] [27] [47].

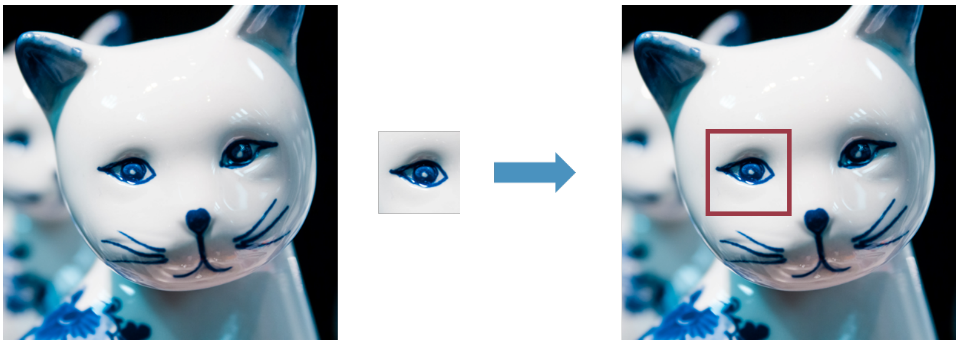


Figure 3.1: Template-based approach example

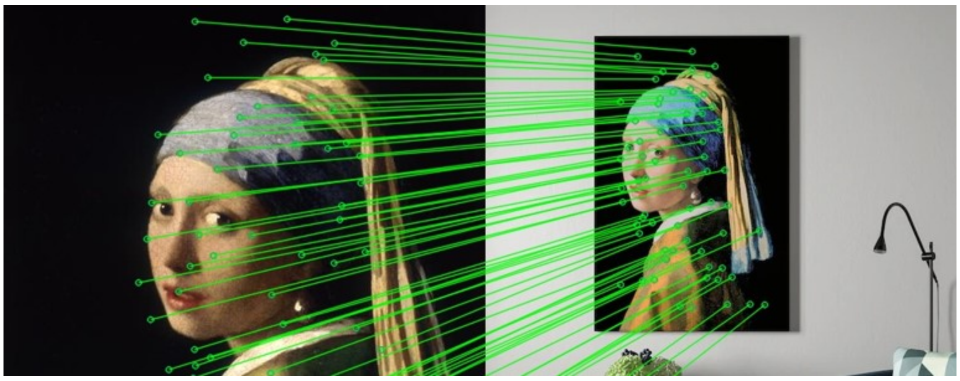


Figure 3.2: Feature-based approach example

3.2.3. Model training and architecture

In compliance with requirement 23, all models considered must be trainable. Training is done to 'teach' the model a desired input-output relation by use of a training dataset. The parameters of the trained model can then be stored, whereafter the model can be used to predict outputs of inputs not seen in the training phase. Models can be trained with different levels of supervision: supervised, unsupervised and some variations that fall in between both. Supervised approaches need labelled data in order to verify and adjust their predictions. Supervised learning generally outperforms other forms of learning [46]. A challenge of supervised learning is the limited quantity of labelled datasets. Regardless of the supervision type, the process of training an algorithm is computationally expensive as algorithms usually need tens of thousands of images in order to learn the correct relation. Training data should be selected carefully so that there is no bias introduced in the model (19).

Neural networks are build up out of layers of neurons. Neurons are the nodes in the network through which the data flows and were computations are performed. The purpose of a layer dictates how its neurons connect to neurons of the previous layer. A convolutional layer extracts features by convolving the previous layer with a filter/kernel. A pooling layer reduces the dimension of the previous layer by replacing the values of a set of neurons by their average or maximum. An activation layer allows for learning complex relations by introducing non-linearity. Fully connected and dropout layers are used to learn non-linear combinations of high-level features and to reduce over-fitting, respectively. Model architectures differ in layer types, number of layers, and number of neurons per layer. Because hand-crafting a neural network architecture is out of the scope of this project (Section 1.1.2), only pre-existing model architectures are included. VGG-16 is an example of such a model architecture. Figure 3.3 and 3.4 show the layers of that model in detail.

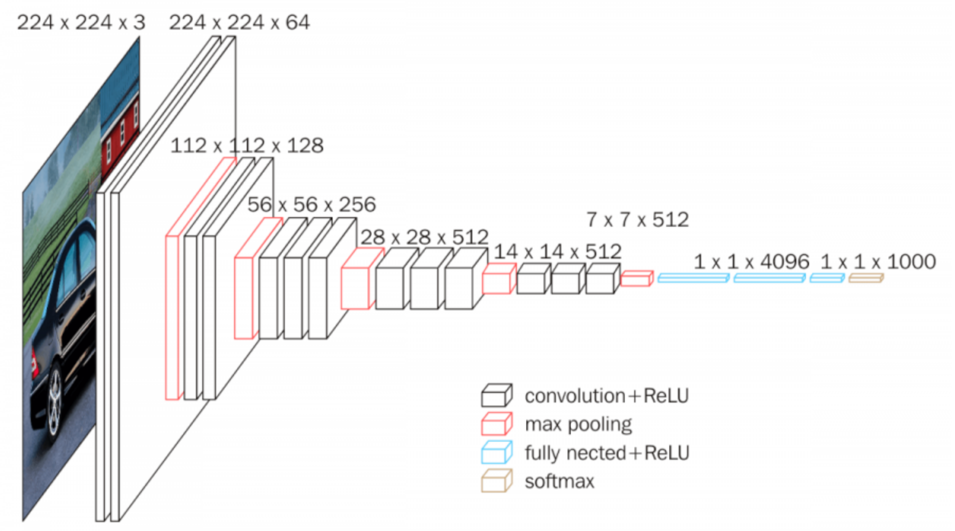


Figure 3.3: Visual representation of the VGG-16 model architecture.

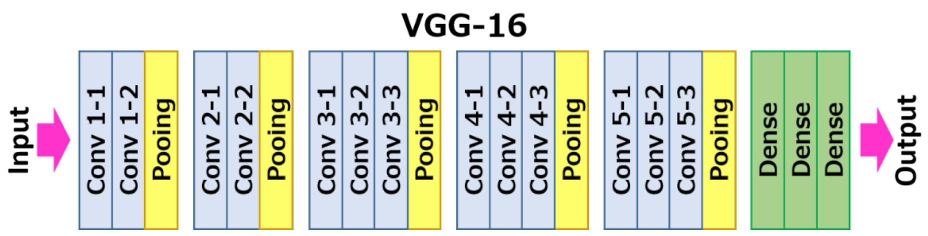


Figure 3.4: The layers of VGG-16 shown chronologically.

By training a neural network, all connections between neurons in the network are assigned the weights that resulted in the best performance during training. Since training doesn't change the architecture of the network (i.e. number of layers or number of neurons), the version of a model trained on a specific dataset is simply described by a set of weights that resulted from training on that dataset. Most well-known neural network models such as VGG-16, VGG-19, Xception, Inception-v3, and ResNet-101, are available as pre-trained models. They each provide a set of their respective weights after training on ImageNet; a large-scale hierarchical image database [48]. ImageNet is a infamous dataset with over 14 million images, organised according to WordNet [49]. Due to the time constraint of this project (12), most evaluated methods in this report make use of pre-trained models.

3.2.4. Pre-processing

Image pre-processing is done to improve performance. It can improve the quality of retrieved images (5, 6, 7) or reduce computation time in extracting and matching (14). Depending on the architecture of the algorithm, the model may expect a fixed number of variables (pixels) at the input. All images must then be resised to that resolution. Image resolution may also be reduced in order to prevent running into hardware limitations for very large images (10, 13).

Next to resizing, images can also be processed to improve the quality of the features found by the algorithm. In general, this comes down to suppressing unwanted distortions/noise and enhancing important image features. Ways to achieve this include but are not limited to: brightness correction, gray scale transformation, gamma correction, histogram equalisation and image segmentation. Most methods of image enhancement however hurt the performance of deep learning algorithms [50], especially when applied next to an algorithms' own pre-processing pipeline.

3.2.5. Performance indicators

To asses the performance of different image matching algorithms, they must be compared based on performance indicators. Mandatory requirements for the system include minimum values for precision (5), recall (6) and balanced accuracy (7). These metrics are commonly used in performance assessment of information retrieval systems [46]. In any system that predicts a binary (true/false) label, the performance can be evaluated by comparing the predicted labels to the ground truth. Every prediction can then be either one of four categories: true positive (TP), false positive (FP), true negative (TN) or false negative (FN). Precision, recall and balanced accuracy can then be found as shown below:

$$Precision = \frac{TP}{TP + FP} \tag{3.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.2}$$

$$Balanced\ accuracy = \frac{1}{2}[\frac{TP}{TP + FN} + \frac{TN}{FP + TN}] \tag{3.3}$$

where TP, FP, TN and FN are the number of matches that fall into their respective categories. Precision is then the fraction of relevant images out of all retrieved images, and recall the fraction of retrieved relevant images out of all relevant images. To yield a high score for either metric is simple. By varying the matching threshold, a high (≈1) precision score can be achieved at the cost of recall performance, and vice versa. Finally, balanced accuracy is a metric for accuracy that accounts for imbalance in classes by averaging the true positive rate and true negative rate.

For the performance of a search engine, an optimum balance between precision and recall should be choosen, depending on the specific use case. Mandatory requirements for both precision (5) and recall (6) are defined in the PoR to be at least 0.80 and 0.25 respectively. It is clear that precision is favored over recall. As varying the decision threshold influences both metrics simultaneously, a precision-recall curve can be used to visually show the relation. Details of precision-recall curves are elaborated on in Chapter 5.

To test and compare the performance of different algorithms, a test dataset can be used. This set should resemble the images present in the databases connected to Engineering Historical Memory [2]. Additionally, it should be similar in structure: a generic search should have a relatively small number of matches compared to the total number of images in the dataset. A needle-haystack type search can then be performed: running the algorithm for a set of different query images (needles) to retrieve matching images from a large dataset (haystack). The results for a specific matching threshold can then be shown in a confusion matrix, and in a precision-recall curve for different thresholds. For the intermediate testing of different methods, a small test set of 26 individual images is used. From these images, a total of 162 image combinations are made to be evaluated by the different methods. Examples of images out of the test dataset used in this chapter can be found in Appendix A.3.

3.3. Evaluated methods

Many possible algorithms [46] exist that can satisfy the PoR. One important limitation of many suggested methods however, is that they do not share their source code publicly. Since it is not realistic to write an own implementation of these methods in the time span of this project (12), only methods that have code readily available are considered. The potential licensing of those code bases, libraries or tools must allow for use in this project to satisfy requirement 8. Next to that, all methods must be written in Python, to satisfy requirement 3. To maximise the chance of a resulting system that is compliant with the requirements, the choice is made to diverge, and test a handful of different methods. After a literature study and suggestions from the supervisor, 5 methods were chosen to investigate further. After intermediate testing, one of the 5 methods was chosen to implement into a final product.

For each of the 5 methods, a basic prototype implementation was made in python. Consequently, all prototypes were tested using the same dataset. This dataset is a relatively small set of images that are structured in a needle-haystack manner (Appendix A.3). All predictions are compared with the ground truth, whereafter the precision, recall and balanced accuracy are calculated for all methods. Next to these metrics, it is also noted how computationally expensive each method is. The performance on the tests is visualised in five respective confusion matrices (Figure A.4), where true negatives, false positives, false negatives and true positives are shown from left to right and top to bottom, respectively. The remainder of this section briefly discusses each method and how it performed on the test dataset. A more technical explanation of each method can be found in Appendix A.2. After the performance evaluation, a conclusion is drawn as to which method is chosen for implementation in the final product.

3.3.1. Quality Aware Template Matching

Quality Aware Template Matching (QATM) [29] uses template-based matching to find similar images. It uses the standard, conventional technique, but also assesses the quality of a match to determine the best possible one. QATM uses a qualitative function (A.1) that determines how many times a match occurs. After that, a likelihood function (A.2) is applied. This is a type of soft-ranking that compares the current patch with all other patches. This comparison is done with the VGG-16 pre-trained CNN [32]. A more detailed explanation on QATM can be found in Appendix A.2.1.

After testing, it can be concluded that QATM does not comply with two out of the three performance requirements, namely precision (5) and balanced accuracy (7). It is noted that template-based matching conceptually does not align well with the use case of CBIR, as the query image is mostly not a template that has to be found in a larger image frame.

3.3.2. Autoencoders

An autoencoder [30] [31] uses feature-based matching. The total system is a combination of an encoder and a decoder. An image is encoded to a latent representation, after which it is decoded to reconstruct the original image. This system facilitates unsupervised learning, as the optimal latent representation can be learned by simply comparing the original and reconstructed image, and applying a loss function on the difference. In training, both the encoder and decoder are used, but after training, merely the encoder part of the system remains. The encoder extracts features from images by computing the latent representations. Images are then matched by computing the euclidean distance between their respective latent vectors; this is an example of global feature matching. A more detailed explanation on autoencoders can be found in Appendix A.2.2.

After testing, it is clear that this implementation of an autoencoder does not satisfy any of the performance requirements 5, 6 or 7. The limitation of this implementation lies in the type of features extracted: these are very low level. As a result, the autoencoder matches images based only on low level features like large shapes or background colors, which is not sufficient.

3.3.3. Convolutional neural network

Convolutional Neural Networks (CNN) are used as building blocks in various computer vision tasks. For image matching, a CNN is used as a global feature extractor, similar to the encoder of an autoencoder. Images are then matched by comparing the distance between their global feature vectors. Different CNN models were tested: VGG16 [32], VGG19 [32] and Xception [33]. All were pre-trained on the ImageNet [48] dataset. More details on this method can be found in Appendix A.2.3.

After testing, decent results are seen for the first time. The CNN, which is trained using supervised learning, outperforms the implementation of an autoencoder. The results show two out of the three performance requirements being met. Only requirement (5) of precision is not satisfied, due to the high number of false positives.

3.3.4. Siamese network

A Siamese neural network [34] (also known as twin neural network) is a combination of two identical CNNs in parallel, used to compute the similarity between two inputs. Siamese networks are trained by supplying positive (matching) and negative (non-matching) image pairs. This is a form of supervised learning. The algorithm learns to increase the distance between feature vectors of negative pairs, and decrease the distance between vectors of positive pairs. After training it then uses the same architecture to compute a similarity score between two inputs, after which matching is done based on a threshold for that similarity score. More details on siamese networks can be found in Appendix A.2.4.

The siamese network shows poor results from testing. It does not satisfy any of the performance requirements. The mediocre performance of the siamese network might be due to improper training on a training set that did not resemble the testing set.

3.3.5. D2-Net

The aim of D2-Net is to obtain a sparse set of features that are robust under challenging conditions and efficient to match and to store [35]. D2-Net uses local features (keypoints) for matching. These features are extracted using a pre-trained CNN. Unique to D2-Net, is that it does not use a detect-*then*-describe for its keypoints, but a simultaneous detect-*and*-describe method.

D2-Net shows very promising testing results. It yields the best score of 1.00 for precision, and respectable scores for recall and balanced accuracy as well. D2-Net is the first, and only, method that satisfies all three performance requirements.

3.3.6. Conclusion

The results for precision, recall and balanced accuracy for all methods are shown in Table 3.1. Additionally, the speed of each algorithm is scored in both extraction time and matching time (both times are for a single match). It should be noted that this comparison of speed is only accurate in orders of magnitude, as the tests could not be performed on identical hardware. D2-Net excels in precision, recall and balanced accuracy. It does show limited performance for speed as the only method that uses keypoints for matching, compared to the other methods which make use of global feature vectors for matching. Looking at the mandatory requirements, it can be concluded that D2-Net performs best. Performance gains such as an increase in speed, a decrease in computational cost and an optimisation of the matching threshold can still be made however. For these reasons, the subjects of pre-processing, tuning the pre-trained CNN, and tuning the matching parameters for D2-Net are elaborated on in more technical detail in Chapter 4. Next to that, the approach for testing the algorithm on a large dataset is discussed.

KPIs	QATM	Autoencoder	CNN	Siamese network	D2-Net
Precision	0.21	0.22	0.63	0.25	1.00
Recall	0.56	0.11	0.56	0.17	0.61
Balanced accuracy	0.57	0.53	0.76	0.55	0.81
Extraction time (s)	n/a	0.09	0.34	0.05	3.10
Matching time (s)	0.83	0.02	0.08	0.01	22.1

Table 3.1: Test results of evaluated methods. Time duration in seconds is calculated per image. Note that the extraction time is per image and the matching time is per image combination

4 Prototype

This chapter discusses the creation, improvement, and implementation of the search engine prototype. Chapter 3 identified shortcomings of existing deep learning methodologies. Based on the intermediary testing, D2-Net is the most promising. However, room for improvement exists in satisfying the PoR. Additionally, this chapter will discuss the preparation for testing on a larger dataset.

4.1. Pre-processing

The first step in pre-processing is scaling images to a user-defined size. This step influences the order of magnitude of keypoints, scores, and descriptors. In the context of this project, the maximum edge size (i.e. how many pixels in width or height) is set to 1200px. The sum of the edges is set to a maximum of 2600px. Criteria for setting these values are the requirements 10, 13, and 14. Although no thorough analysis is performed on the effects of these parameters, lower size values led to lower extraction time and fewer matches. This holds vice-versa.

The existing color pre-processing of D2-Net is either a ‘Caffe’ or ‘Torch’ implementation. This is used as preparation for the extraction process. The torch approach is a full color normalization: a mean of 0 and a standard deviation of 1. The caffe approach zero-centers the input images by mean pixel subtraction. This is based on the pixel values from the original training data by the authors of VGG [32]. In Figure 4.1 different approaches are shown: (i) no pre-processing, (ii) Torch pre-processing, and (iii) Caffe pre-processing. Figure 4.2 shows the RGB pixel value distribution of these approaches.

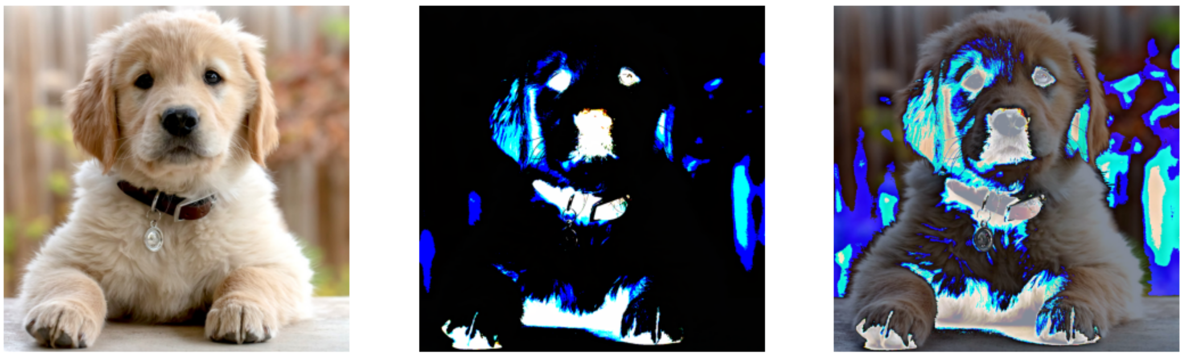


Figure 4.1: Color pre-processing visual example

Apart from the D2-Net pre-processing approach mentioned above, other research in image enhancement suggests that image pre-processing in Convolutional Neural Networks reduces performance [50]. This includes fine-tuning the pre-trained CNN model using contrast limited adaptive histogram equalization (CLAHE), successive means quantization transform (SMQT), Wavelet transform, and Laplace operator.

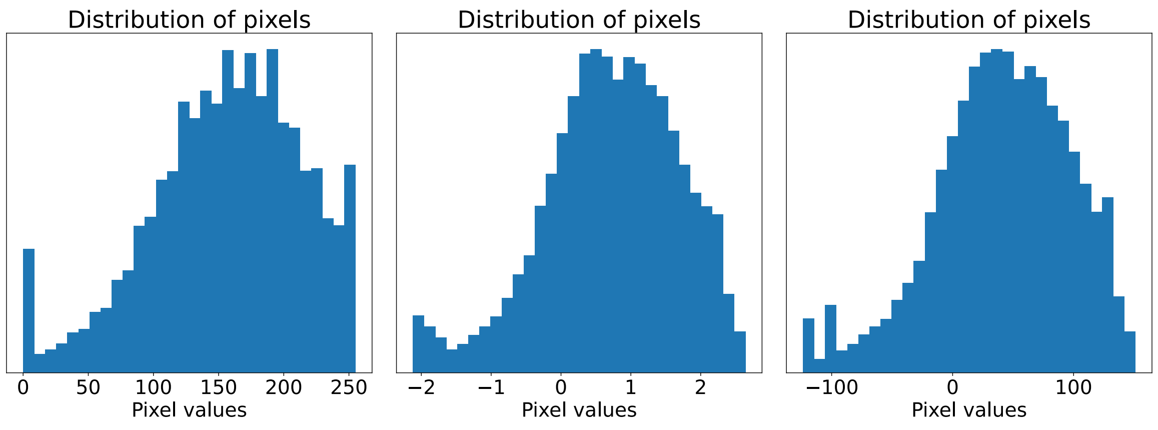


Figure 4.2: Distribution of pixel values of Figure 4.1

Based on the above, the existing implementation of D2-Net pre-processing is deemed sufficient to evaluate the performance of the prototype on a large dataset. Some minor changes in code have been made such as replacing deprecated code from Skimage. This is replaced by a OpenCV implementation. Section B.1 includes the code used for pre-processing.

4.2. Extraction

The extraction process of the prototype is fully based on D2-Net. D2-Net uses a single Convolutional Neural Network (CNN) to extract dense features serving as both descriptors and detectors (describe-and-detect approach) [35]. The first step in the extraction process is to insert an image I into a CNN $\mathcal{F}$ to obtain a 3D tensor F, shown in Equation 4.1. The letters h, w, and n are the height, width, and number of channels of the feature maps respectively.

$$F = \mathcal{F}(I),\ F \in \mathbb{R}^{h \times w \times n} \tag{4.1}$$

4.2.1. Feature description

From F a dense set of descriptor vectors $\mathbf{d}$ is formed. This is shown in Equation 4.2, with $i = 1, ..., h$ and $j = 1, ..., w$. These descriptors are normalized using the L2-norm (euclidean distance) of the descriptor vectors and can be compared to descriptors of other images.

$$\mathbf{d}_{ij} = F_{ij},\ \mathbf{d} \in \mathbb{R}^n \tag{4.2}$$

$$\hat{\mathbf{d}}_{ij} = \frac{\mathbf{d}_{ij}}{\|\mathbf{d}_{ij}\|_2} \tag{4.3}$$

4.2.2. Feature detection

Feature detection is conducted by obtaining a collection of 2D responses D from the 3D tensor F. This is illustrated in Equation 4.4, where $k = 1, \ldots, n$. These n different 2D response maps are post-processed to obtain output keypoints [35].

$$D^k = F_{::k},\ D^k \in \mathbb{R}^{h \times w} \tag{4.4}$$

This post-processing is split into *hard feature detection* and *soft feature detection*. Hard feature detection uses the multiple detection maps $D^k (k = 1, \ldots, n)$. For a point (i, j) to be detected, the following is required:

$$(i,j) \text{ is a detection} \iff D^k_{ij} \text{ is a local maximum in } D^k.$$

$$\text{with } k = \arg\max_t D^t_{ij}$$

In other words, the algorithm selects the most striking channel D and verifies if a local-maximum exists at position (i,j) on that channel. However, to become 'learned' at performing this task, some sort of feedback is necessary. This is done through a process called back-propagation. The hard feature detection is not amenable for back-propagation. For that reason soft feature detection is implemented as is described in the paper of D2-Net [35]. First, a soft local-maximum score α^k_{ij} is defined, shown in Equation 4.5.

$$\alpha^k_{ij} = \frac{\exp\left(D^k_{ij}\right)}{\sum_{(i',j')\in\mathcal{N}(i,j)} \exp\left(D^k_{i'j'}\right)}, \tag{4.5}$$

where $\mathcal{N}(i,j)$ is defined as the set of 9 neighbours of the pixel (i,j) (including itself). Secondly, a soft channel selection parameter β^k_{ij} is defined that computes a ratio-to-maximum per descriptor. In simpler words, it calculates how large the descriptor in channel k at pixel (i,j) is compared to the maximum value descriptor in channel t at pixel (i,j). This is expressed in Equation 4.6.

$$\beta^k_{ij} = \frac{D^k_{ij}}{\max_t D^t_{ij}} \tag{4.6}$$

These figures are combined as γ_{ij} by maximizing the product of both across all feature maps k. This is shown in Equation 4.7. Lastly, the soft detection score s_{ij} is calculated by performing an image-level normalization (Equation 4.8).

$$\gamma_{ij} = \max_k \left(\alpha^k_{ij}\beta^k_{ij}\right) \tag{4.7}$$

$$s_{ij} = \frac{\gamma_{ij}}{\sum_{(i',j')} \gamma_{i'j'}} \tag{4.8}$$

Important in Sections 4.2.1 and 4.2.2 is the size of the image. It determines, or rather limits, the amount of keypoints and descriptors. The size of images is related to resolution as well (i.e. a picture can be scaled and therefore its resolution changes). However, the performance of the CNN is not invariant to scale changes; extraction and matching is affected [35]. The D2-Net implementation accounts for this by multi-scale detection. An image pyramid I^ρ is constructed with three different resolutions $\rho = 0.5$, 1, and 2. Subsequently, feature maps F^ρ are extracted, and larger image structures in lower resolution feature maps are propagated to higher resolution feature maps using Equation 4.9.

$$\tilde{F}^\rho = F^\rho + \sum_{\gamma<\rho} F^\gamma \tag{4.9}$$

To enable this summation in different resolutions, feature maps F^γ are resized to resolution F^ρ using bilinear interpolation. Additionally, in order to avoid re-detection of features, actions are performed in the following order: (i) start at the coarsest scale, (ii) mark and upsample the detected positions (using nearest neighbour) to the resolutions of the next scales, and (iii) ignore detections falling into marked regions. A perhaps more intuitive explanation on multi-scale detection in general can be found in Section A.3.

The extraction of keypoints and descriptors is at the core of the image search engine. Using a CNN, image features become accessible to process and extract. An illustration of feature description and detection is shown in Figure 4.3. When extracted, keypoints, scores (how much activation per keypoint), and descriptors are saved to a numpy file. Part of the code used for extracting features is shown in Section B.2.

The implemented feature extraction network $\mathcal{F}$ is pre-trained. It is based on the VGG16 architecture, trained on ImageNet, and truncated after the `conv4_3` layer. VGG16 is a neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford [32]. More details on the fine-tuning of the CNN can be found in the D2-Net paper [35].

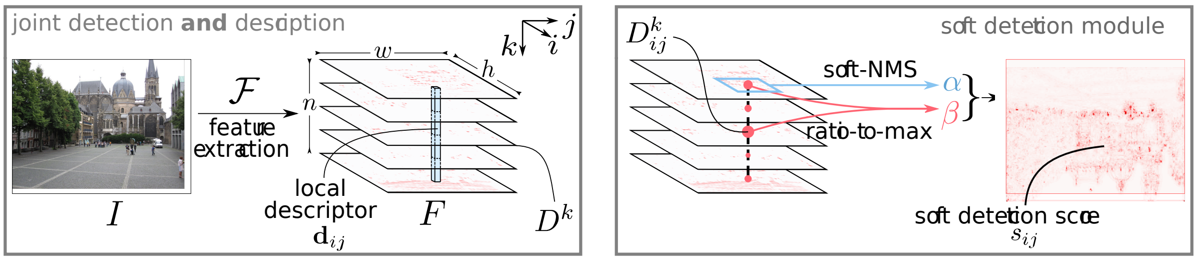


Figure 4.3: Detection and Description visual from [35]

4.3. Matching

The matching implementation consists of two parts: (i) matching of keypoints and descriptors and (ii) validation of matches using random sample consensus (RANSAC).

The matching is based on a brute-force approach and implemented using OpenCV [24]. For each descriptor in the first set, the closest descriptor in the second set is found by trying each one. Cross-checking is enabled: a match is valid if both descriptors from the sets are closest (euclidean distance) to each other. The advantage of brute-force matching is that it finds the best possible image feature matches. However, a major drawback is the time necessary to find matches. A trade-off is necessary between requirements 14, 15, 16, and 17. Precision and accuracy are considered more important than the speed of the algorithm; defined in the PoR.

The random sample consensus (RANSAC) is used for geometric verification. When two images have many supposed matches, a geometric verification is used to filter wrong feature matches that occur due to a viewpoint difference. More fundamentally speaking, RANSAC is an iterative method capable of identifying inliers (i.e. proper matches) and mitigating the influence of outliers (i.e. incorrect matches) [51]. Additionally, it allows the orientation of images to be determined. This is not further investigated, but suits future potential for algorithm transparency by indicating the interpreted orientation (contributing to requirement 18 and 20). A visual example of RANSAC is given in Figure 4.4. The algorithm has been implemented using pydegensac [26] [27] [52]. This particular implementation scores high and marginally better than the OpenCV implementation [47]. The Python code implementation is shown in Appendix B.3.

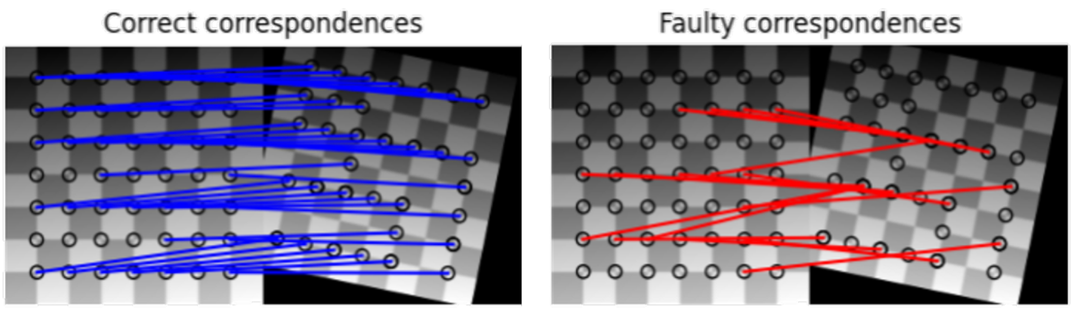


Figure 4.4: Example of RANSAC in use (inliers in blue) and identifying the incorrect matches (outliers in red) [28]

4.4. Training

In order to train a neural network model a loss function $\mathcal{L}$ is needed. A loss function is an objective function that searches for a solution resulting in the best score. It translates all aspects from the model into a number that reflects the improvements. In [35] the loss function used is called a triplet margin ranking loss [53]. In the triple loss function in Equation 4.10 the s_c are the soft detection scores from Equation 4.8 at the points *A* and *B* of the two images being matched and $\mathcal{C}$ is the set of all matched features of the two images. This loss function creates the sum of the weighted average *m* that can then be used to minimize the loss and increase the accuracy of the matching between the descriptors.

$$\mathcal{L}(I_1, I_2) = \sum_{c\in\mathcal{C}} \frac{s_c^{(1)} s_c^{(2)}}{\sum_{q\in\mathcal{C}} s_q^{(1)} s_q^{(2)}} m(p(c), n(c)) \tag{4.10}$$

4.5. Testing

Testing preparation is necessary to evaluate the search engine and verify whether the search engine requirements have been reached. The requirements specify precision, recall, and balanced accuracy as KPIs. The output parameters include the duration in seconds, the amount of matches, and an inlier count per image pair. Only the inliers are used to determine precision, recall, and balanced accuracy. The approach is to perform analysis on how many inliers results the best performance. To prevent overfitting the data, the dataset is split into training (70%) and testing (30%) [54]. A simple `for-loop` is performed on the training subset to obtain the ideal threshold, which is based on the accuracy values. Next, the metrics accuracy, balanced accuracy, precision and recall are calculated using the testing subset, and a precision-recall curve is plotted. The code can be found in Appendix B.4 and is executed in a Google Colab environment. Additionally, the time duration of running the code on the High Performance Cluster has been recorded. In Chapter 5 the results are clarified and elaborated. The image extraction and matching process requires a dataset and sufficient computing power. These requirements are elaborated in the Subsections 4.5.1 and 4.5.2 respectively.

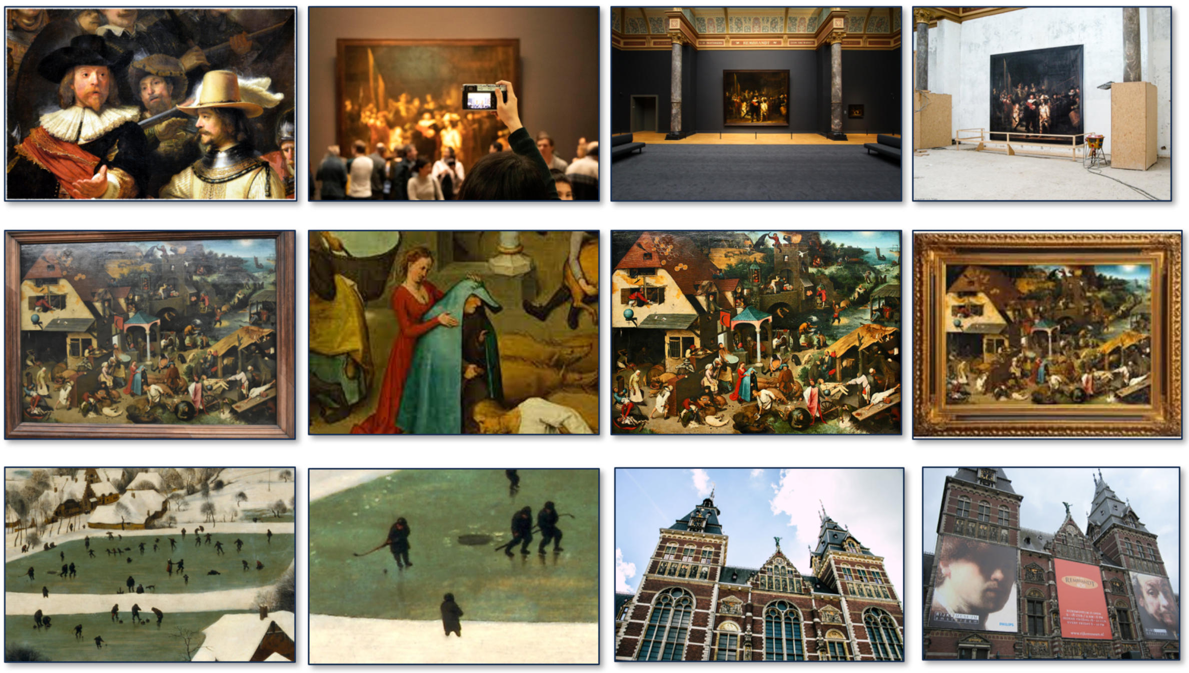


Figure 4.5: A subset of images from the dataset

4.5.1. Dataset

The final dataset contains 159 images and is used to assess a total of 12.6k image combinations (i.e. 159 choose 2). A comma-separated values (CSV) file is used to guide the code and to write back results. It contains the relative path of images and the ground-truth value on whether it is a match. The images, related to history, heritage, and art, have been scraped from the web and can be found in the GitHub repository. The code used to generate the initial CSV file can be found in Appendix B.4.4. The set has a lot of ground-truth negatives, intentionally representing the real case scenario of images not matching. A preview of the dataset is visible in Figure 4.5.

4.5.2. High Performance Cluster

Computing power is necessary to perform feature extraction and matching. The necessary amount of power depends on the size of the image, the amount of images, the Python implementation, and the demanded speed of the algorithm. Particularly for the Python implementation, some libraries support GPU calculations through CUDA and cuDNN. This makes calculations a lot more efficient in hardware utilisation. D2-Net recommends at least 12GB of VRAM to handle the multi-scale implementation [35]. Additionally, the brute-force matching methodology is computationally intensive. Although intermediate and small analysis have been performed using Google Colab and DeepNote, the implementation required stronger hardware. For that reason, the High Performance Cluster (HPC) of Delft University of Technology has been utilised. To increase the hardware efficiency, multiprocessing has been used for brute-force matching as OpenCV GPU support was unavailable on the HPC. The extraction implementation has made use of CUDA and available VRAM. The large dataset required 24 hours of calculations, mostly due to the brute-force approach.

5
Results

This chapter elaborates on the results of the prototype described in Chapter 4. In the following sections, the results are illustrated and clarified for interpretation.

5.1. Performance results

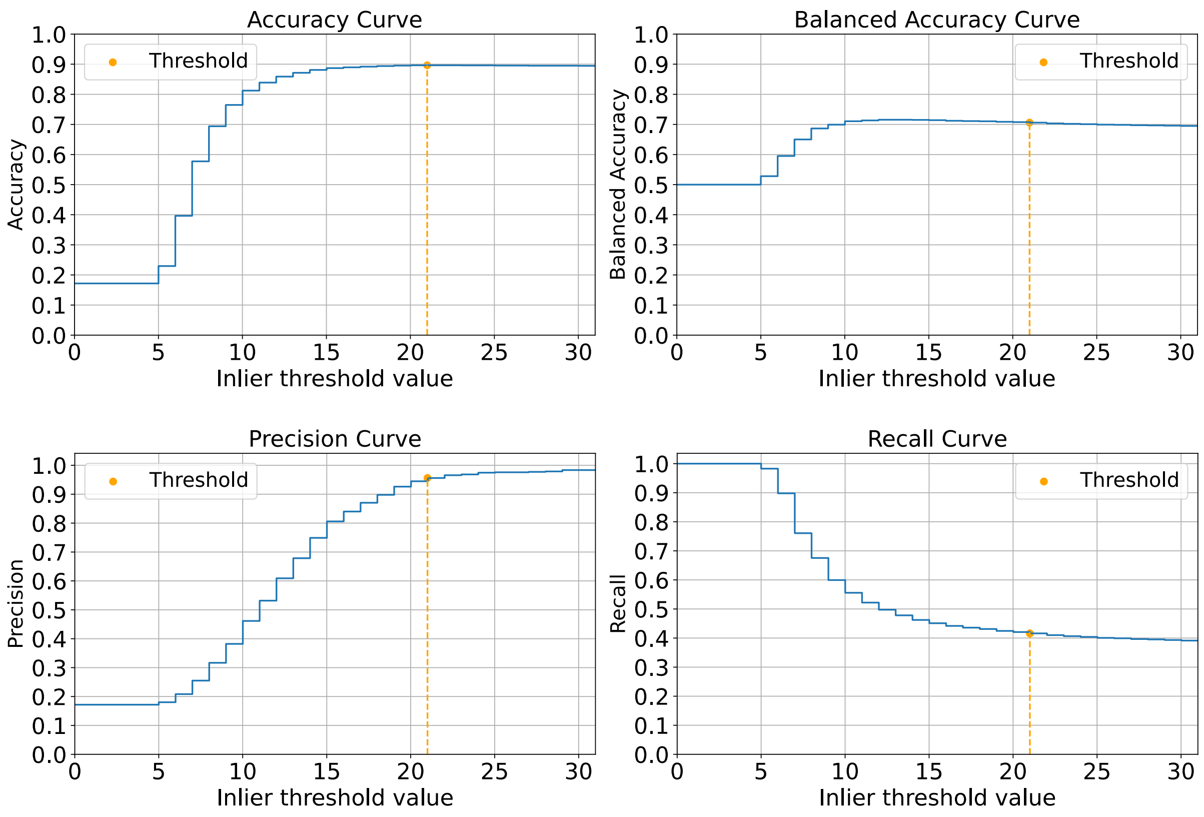


Figure 5.1

The four graphs in Figure 5.1 represent the metric scores of the training data subset. Section 3.2.5 goes into detail on the significance of the performance indicators. Table 5.1 shows the Key Performance Indicator values determined with a threshold value of 21 inliers.

KPIs	Training Set	Testing Set
Accuracy	0.90	0.91
Balanced Accuracy	0.71	0.72
Precision	0.96	0.96
Recall	0.42	0.44

Table 5.1: Key performance indicator values on both subsets of the dataset

Based on the evaluation of the training subset, a precision-recall curve is created and shown in Figure 5.2. The determined threshold of 21 inliers is shown with an orange dot.

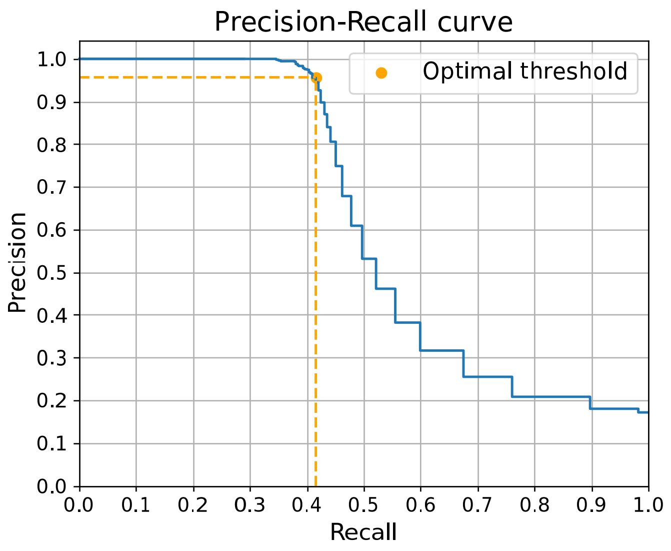


Figure 5.2: The precision recall-curve of the testing data subset

Next to the above metrics, the time duration of each process has also been recorded. These are displayed in Table 5.2.

Process	Duration (1 Core)	Duration (4 Cores)
Extraction	2.0	2.0
Brute Force Matching	34.0	5.66
RANSAC	0.21	0.03

Table 5.2: Average time durations in seconds: extraction is per image and both brute force matching and RANSAC are per image combination.

6 Conclusion

The requirements from Chapter 2 have been fulfilled and are discussed below. In Chapter 1, the following problem statement was defined:

To develop an algorithm in ten weeks that compares image queries with other images purely based on image content in multiple formats using a deep learning approach.

After initial tests on multiple different methodologies in Chapter 3, D2-Net was decided as the basis for the deep learning search engine prototype (22). The search engine has been improved and optimized and is elaborately explained in Chapter 4. The search engine performance has been evaluated on the High Performance Cluster using a large dataset. The results are visible Table 6.1 and satisfy the requirements stated in Chapter 2.

KPIs	Requirements	D2-Net
Precision	0.80	0.96
Recall	0.25	0.45
Balanced Accuracy	0.70	0.72
Extraction time (s)	ToR	2.0
Matching time (s)	ToR	5.69

Table 6.1: Requirements and obtained results. Note that extraction time is per image, and the matching time is the addition of brute-force matching time and RANSAC time per image combination.

Requirements (1) and (2) are achieved. Images are taken as an input and matches that correspond to that image are found. The software was written completely in Python and is ready for integration with the EHM website (3 and 4). From the comparison in Table 6.1, it is clear that the three performance requirements (5), (6) and (7) exceed the targets that were set. Only open-source Python libraries were used to create the algorithm, complying with (8). As requested in (9), the system can accept common image file extensions and use them without hindrance. The size of the files can be up to 10MB (10) but also allows for larger files by resizing images. To increase its speed in matching, multiprocessing was implemented (11) on the HPC (13). The whole project was completed in the span of the 10 weeks (12).

With the mandatory requirements achieved, a closer look is taken at the trade-off requirements to asses how well the software performs. In Chapter 5 the values precision (15), recall (16) and balanced accuracy (17) have been maximized in such a way that at least the minimum requirements stated for each parameter were met. Precision was deemed more important than recall, and the values can be seen in Table 6.1. The accuracy of the software weighs more than its speed. This results in a longer search time (14), but was reduced by multiprocessing. The whole project, including the code and datasets used for testing have been kept and organized (18). The link to the GitHub can be found in Appendix B. Furthermore, the program works without any type of user feedback and is a deterministic open-loop system (19). Multiple formats, e.g. jpg or png, are supported (21). The output provides the number of inliers, and optionally an image, showing if and how a match is found (20).

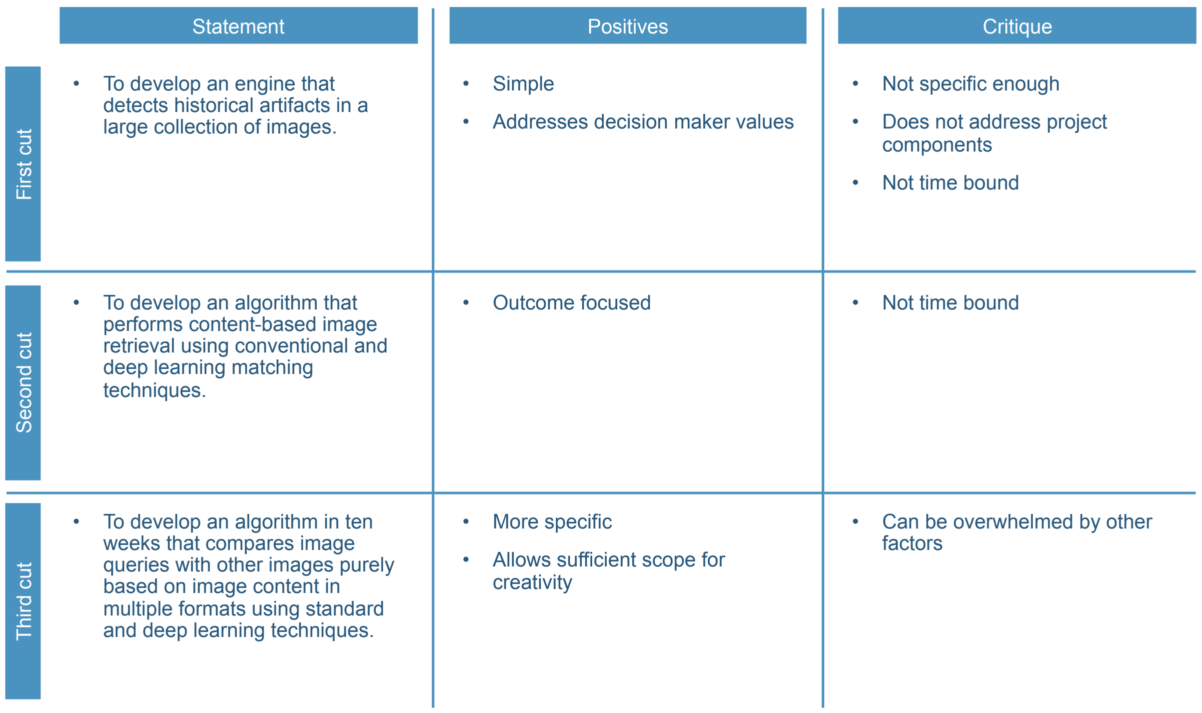
6.1. Discussion

Although the performance meets the specified requirements, room for improvement exists. Implementations that seem promising include an approach for deep learning image retrieval, discussed in [39], and a novel training method described by [55]. Its implementation and code is available on GitHub [56]. Additionally, as discussed in Chapter 4, the model is trainable because D2-Net uses deep learning (23). Training the model increases accuracy in specific certain cases. Since the primary topic of searches for EHM are about historical subjects and heritage science, the algorithm can be fine-tuned to perform better on these subjects by training. Another opportunity lies in improving the matching. Different types of feature matchers should be tested to try and increase the precision, recall, and balanced accuracy scores. Especially matchers that make use of CUDA should be investigated. They can greatly improve the speed of the algorithm, which is a very important point to improve upon if the prototype is to be used on a large scale. Lastly, the bag-of-visual words approach shows potential for improving content-based image retrieval. Currently, it is predominantly used for image classification, but has potential to give a better balance between precision, recall, accuracy, and time. However, its limitations should be investigated.

A Appendix

A.1. Project approach and teamwork

This section elaborates on Chapters 1 and 2. To obtain a better understanding of the problem at hand, weekly discussions with J. Dauwels and A. Nanetti have been held. Particularly understanding the bigger picture of Engineering Historical Memory and the multidisciplinary nature adds an interesting perspective to this project. In Figure A.1 the problem statement development is shown. In Figure A.2 the inductive logic tree is shown. This has been used to manage and make decisions in which direction lie to the problem statement. The group held a daily starter every morning, either physically at TU Delft and online using Microsoft Teams. A supervisor meeting was held every week in which the progress was discussed. Additionally, a GANTT chart was sent every week on Monday.



	Statement	Positives	Critique
First cut	• To develop an engine that detects historical artifacts in a large collection of images.	• Simple • Addresses decision maker values	• Not specific enough • Does not address project components • Not time bound
Second cut	• To develop an algorithm that performs content-based image retrieval using conventional and deep learning matching techniques.	• Outcome focused	• Not time bound
Third cut	• To develop an algorithm in ten weeks that compares image queries with other images purely based on image content in multiple formats using standard and deep learning techniques.	• More specific • Allows sufficient scope for creativity	• Can be overwhelmed by other factors

Figure A.1: Problem statement progression

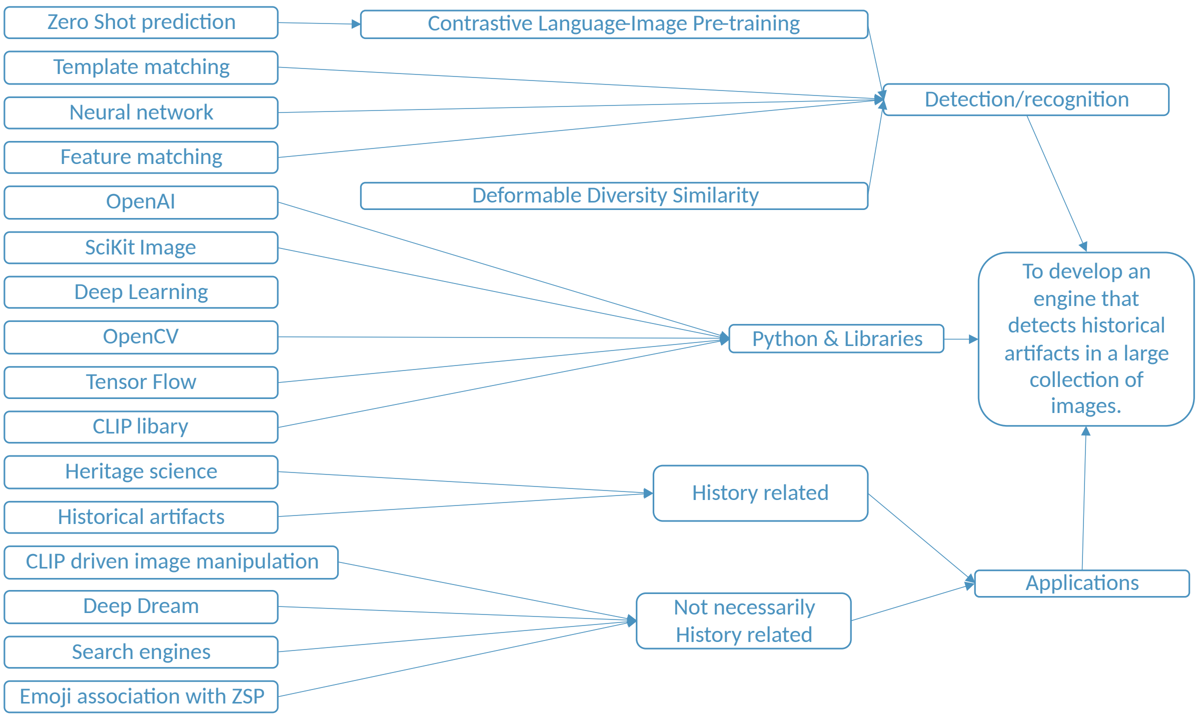


Figure A.2: Non-exhaustic inductive logic tree used for studying this field of research.

A.2. Methodologies

A.2.1. Quality-Aware Template Matching

As stated in Chapter 3, Quality-Aware Template Matching (QATM) uses the standard template matching but adds an extra quality factor to the different types of matches it finds. To create a quality score, five different matching cases are considered with a patch *t* from the template image **T** and a patch *s* from the search image **S**.

1. 1-to-1
2. 1-to-N
3. M-to-1
4. M-to-N
5. no match

From these five possibilities only the first one is referred to as a high quality match because it means two objects are matched and it only occurs once. For 1-to-N and M-to-1 a pattern in either the template or search image is found such as a wall, sky or floor. The last possibility is immediately excluded and M-to-N matches indicate many homogeneous/patterned patches are found. By excluding all but the 1-to-1 matching case the reliability of the obtained match is increased.

A quantitative assessment of the matching cases is used to find the region R of **S** that maximizes the matching quality. R is the fixed size window that corresponds to the size of the template. The function can be seen in Equation A.1.

$$R^* = arg_R max\{\sum max\{Quality(r,t)|t \in \mathbf{T}\}\} \tag{A.1}$$

In Equation A.1, the Quality(r,t) is the function that assesses the the matching quality between *s* (equal to r) and *t*. For this function, the similarity between patches must be determined. This is done through the likelihood function in Equation A.2.

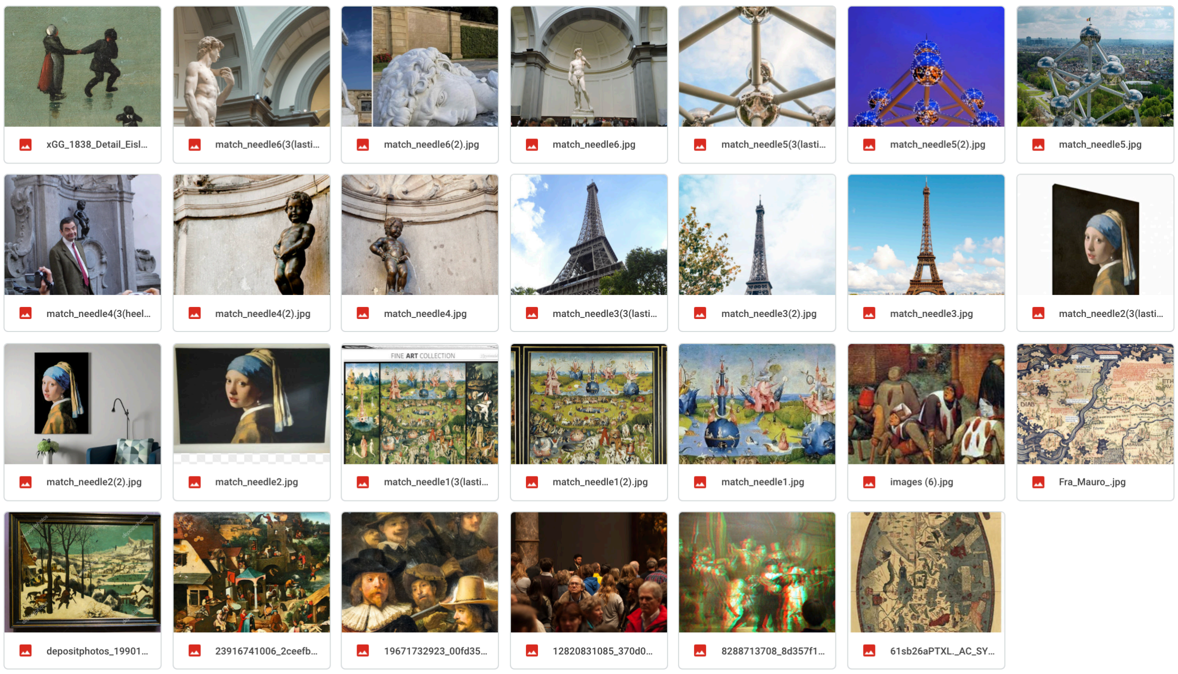


Figure A.3: Data set for intermediate testing of different methodologies

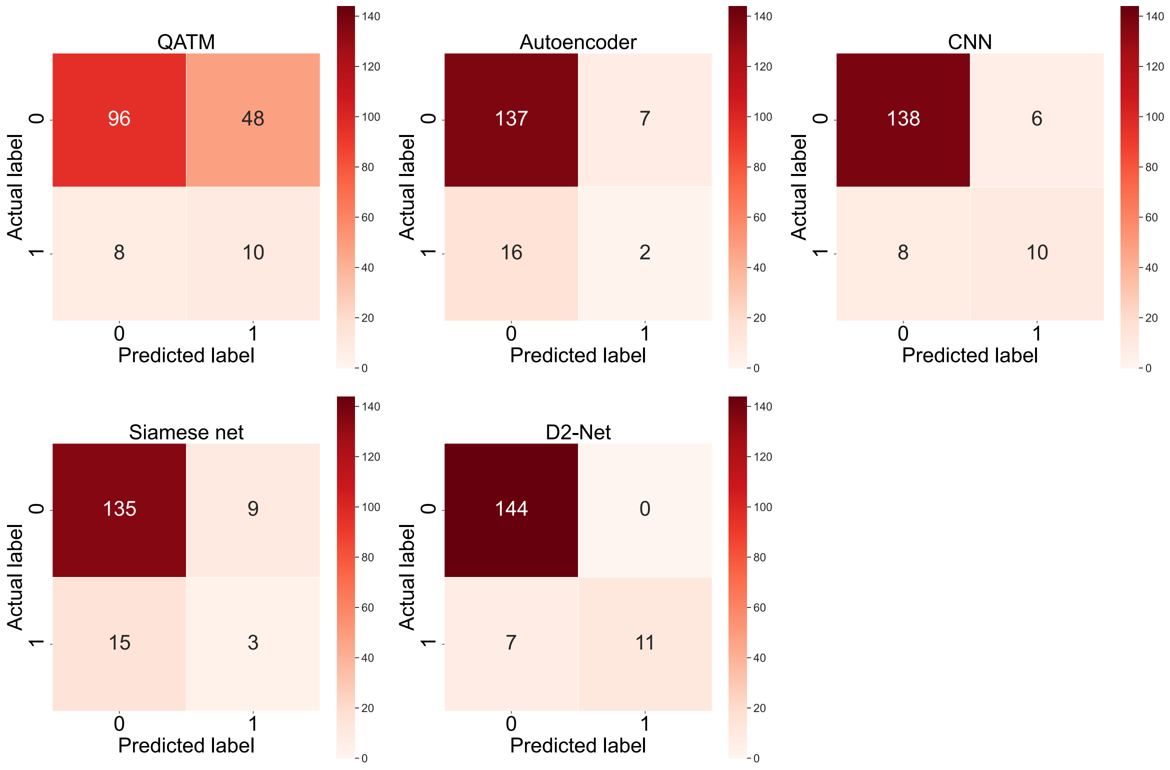


Figure A.4: Confusion matrices for the 5 evaluated methods.

$$L(t|s) = \frac{exp\{\alpha \cdot \rho(f_t, f_s)\}}{\sum_{t' \in \mathbf{T}} exp\{\alpha \cdot \rho(f_t, f_s)\}} \tag{A.2}$$

In Equation A.2, f_s and f_t are the feature representations of the patches s and t, ρ is a similarity measure between two patches that makes use of a pre-trained neural network. The variable α is a factor that adds extra quality discernibility to the function. The value for α is taken directly from [29] where they empirically determined what the optimal values could be.
The quality measure can the be defined as the product of the likelihood that s is matched in **T** and t is matched in **S** as shown in Equation A.3.

$$\textbf{QATM}(s,t) = L(t|s) \cdot L(s|t) \tag{A.3}$$

With Equation A.3, the scores of each matching case are calculated.

Matching Case	L(s|t)	L(t|s)	QATM(s,t)
1-to-1	1	1	1
1-to-N	1	1/N	1/N
M-to-1	1/M	1	1/M
M-to-N	1/M	1/N	1/MN
No match	1/||**S**||	1/||**T**||	≈0

Table A.1: Ideal QATM scores [29]

The matching quality of an region in **S** can then be found with Equation A.4.

$$q(s) = max\{QATM(s,t)|t \in \mathbf{T}\} \tag{A.4}$$

The best matched region R^* can then be found with Equation A.5 which maximizes the overall matching quality.

$$R^* = arg_R max\{\textstyle\sum_{r \in R} q(r)\} \tag{A.5}$$

A.2.2. Autoencoders

An autoencoder is a type of an unsupervised neural network, meaning no class labels or labeled data is needed to train it. The autoencoder encodes an input image into a latent representation, after which it is decoded back into an image similar to the input image. Dependent on the level of detail needed to reconstruct the input, the the latent space can be small in size. In Figure A.5, a sample of the MNIST [57] data set is encoded and reconstructed. For such a data set, which contains only 10 classes (all Arabic numerals), inputs can be reconstructed with very little loss from a small latent representation.

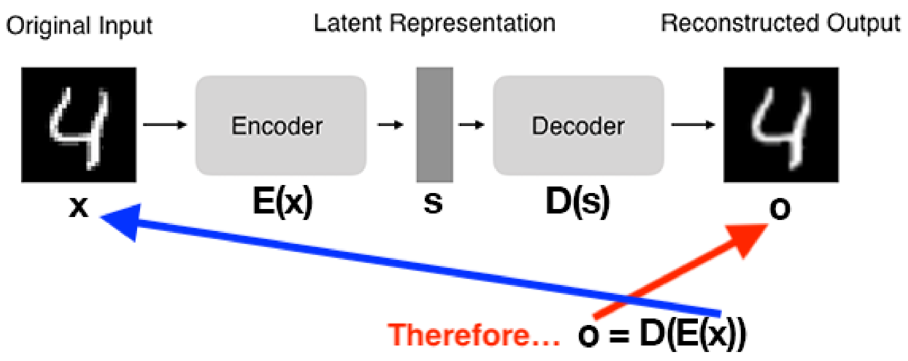


Figure A.5: Schematic example of the structure of an autoencoder.

$$MSE = \frac{\sum_{i=1}^{n}(y_i^o - y_i^x)^2}{n} \tag{A.6}$$

An autoencoder is trained by use of a loss function that compares the reconstructed image with the input image, and tunes the network accordingly to minimize the error (in this case the mean square error, shown by Equation A.6). This pipeline of encoding and decoding does not provide any use for the task of CBIR. However, the true value of an autoencoder lies in the latent representation. The meaning of its entries might be arbitrary, but they do describe features that are key in representing the input image. For CBIR, one can strip the system of the decoder, and use the encoder as a global feature extractor. The distance (Equation A.7) between these feature vectors can then be used to match their corresponding images.

A.2.3. Convolutional Neural Network

A Convolutional Neural Network (CNN) uses multiple convolutional layers made up of filters that utilizes neurons to process the input image into a feature map. The distance between feature maps is then used to determine whether or not two images match. A feature map is the result of a filter that has been applied to the input image. So at every layer of the neural network the output is a feature map which contains the detected features of the image.

The matching itself is done by calculating the euclidean distance in between the features of the image used to search and the query image from a database. The euclidean distance is calculated with Equation A.7, where n is the dimension size of the features. To determine whether a match is found a threshold for the maximum allowed distance can be set.

$$D(a,b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + (a_3 - b_3)^2 + \dots + (a_n - b_n)^2} \tag{A.7}$$

A.2.4. Siamese network

The siamese network consists of two CNN working together to maximize the euclidean distance for negative and minimize it for positive matches. This is done through a loss layer. The loss layer function can be seen in Equation A.8 [34].

$$\mathcal{L} = \frac{1}{2} l D^2 + \frac{1}{2}(1 - l)\{max(0, m - D)\}^2 \tag{A.8}$$

In Equation A.8, the *l* is a label that selects whether the input matches or not (*l* can be either 0 or 1), *m*>0 is the margin for dissimilar pairs and *D* is the euclidean distance between the image features from Equation A.7. When the siamese network was tested, a pre-trained model was used and no labels were attached to the files to simulate a real situation where random images have to be matched without any prior knowledge about the images.

A.3. Multi-scale principle

Multi-scale detection is used in computer vision to detect features better and more efficiently. An intuitive way to think about it, is to look how humans detects and looks for objects. For example, if a person looks for a tree, he or she implicitly looks for objects in the 10^1 order of magnitude. In an image a person can infer relative sizing by looking at other objects such as people, buildings, and the sky. However, a computer cannot easily or automatically infer this. To overcome this obstacle, a computer analyses an image in multiple resolutions. A commonly used method is the pyramid representation [58]. A visual representation of an image pyramid is shown in Figure A.6. In this project a pyramid with 3 levels is used with resolutions [0.5, 1, 2].

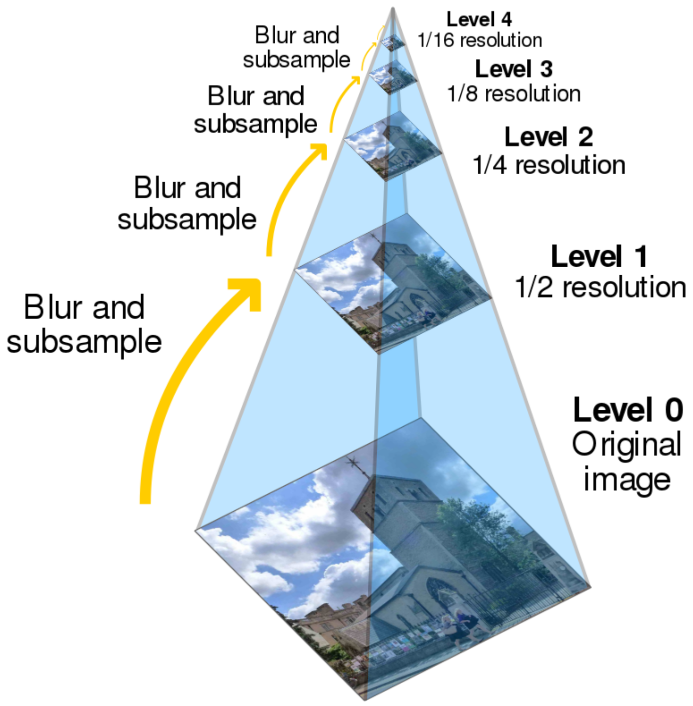


Figure A.6: A visual representation of an image pyramid with 5 levels

B
Code

This appendix chapter includes snippets of code used for the search engine. A significant part of its implementation is based on the D2-Net [35] code. Other code is primarily based on OpenCV [24] and Pydegensac [26] [27] [47]. A GitHub repository has been created that includes the dataset and Python code for both the standard and deep learning approach [1]. It can be found here.

B.1. Pre-processing

B.1.1. Read and resize images

```
# Read image
image = cv2.imread(path)

if image is not None:
    # format image array if necessary
    if len(image.shape) == 2:
        image = image[:, :, np.newaxis]
        image = np.repeat(image, 3, -1)

    # resize images
    resized_image = image
    if max(resized_image.shape) > MAX_EDGE:
        fraction = MAX_EDGE / max(resized_image.shape)
        width = int(resized_image.shape[0] * fraction)
        height = int(resized_image.shape[1] * fraction)
        dim = (width, height)
        resized_image = cv2.resize(resized_image, dim).astype('float')

    if sum(resized_image.shape[: 2]) > MAX_SUM_EDGES:
        fraction = MAX_SUM_EDGES / sum(resized_image.shape[: 2])
        width = int(resized_image.shape[0] * fraction)
        height = int(resized_image.shape[1] * fraction)
        dim = (width, height)
        resized_image = cv2.resize(resized_image, dim).astype('float')
```

B.1.2. Color

```python
def preprocess_image(image, preprocessing=None):
    image = image.astype(np.float32)
    image = np.transpose(image, [2, 0, 1])
    if preprocessing is None:
        pass
    elif preprocessing == 'caffe':
        # RGB -> BGR
        image = image[:: -1, :, :]
        # Zero-center by mean pixel
        mean = np.array([103.939, 116.779, 123.68])
        image = image - mean.reshape([3, 1, 1])
    elif preprocessing == 'torch':
        image /= 255.0
        mean = np.array([0.485, 0.456, 0.406])
        std = np.array([0.229, 0.224, 0.225])
       image = (image - mean.reshape([3, 1, 1])) / std.reshape([3, 1, 1])
    else:
        raise ValueError('Unknown preprocessing parameter.')
    return image
```

B.2. Extraction

```python
        with torch.no_grad():
                if MULTISCALE:
                    keypoints, scores, descriptors = process_multiscale(
                        torch.tensor(
                            input_image[np.newaxis, :, :, :].astype(np.float32),
                            device=DEVICE
                        ),
                        model
                    )
                else:
                    keypoints, scores, descriptors = process_multiscale(
                        torch.tensor(
                            input_image[np.newaxis, :, :, :].astype(np.float32),
                            device=DEVICE
                        ),
                        model,
                        scales=[1]
                    )

            # Input image coordinates
            keypoints[:, 0] *= fact_i
            keypoints[:, 1] *= fact_j
            # i, j -> u, v
            keypoints = keypoints[:, [1, 0, 2]]

            if OUTPUT_TYPE == 'npz':
                with open(path + OUTPUT_EXTENSION, 'wb') as output_file:
                    np.savez(
                        output_file,
                        keypoints=keypoints,
                        scores=scores,
                        descriptors=descriptors
                    )
            elif OUTPUT_TYPE == 'mat':
                with open(path + OUTPUT_EXTENSION, 'wb') as output_file:
                    scipy.io.savemat(
                        output_file,
                        {
                            'keypoints': keypoints,
                            'scores': scores,
                            'descriptors': descriptors
                        }
                    )
            else:
                raise ValueError('Unknown output type.')
```

B.3. Matching

```python
# Loading the numpy data files that include keypoints, scores, and
  ↪  descriptors
feat1 = np.load(data_url1)
feat2 = np.load(data_url2)

# OpenCV Brute Force implementation
bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)
# Match descriptors
matches = bf.match(feat1['descriptors'], feat2['descriptors'])
# Sort on distance
matches = sorted(matches, key=lambda x: x.distance)

# Indexing of matches
match1 = [m.queryIdx for m in matches]
match2 = [m.trainIdx for m in matches]

# Get subset of keypoints that are matches
keypoints_left = feat1['keypoints'][match1, : 2]
keypoints_right = feat2['keypoints'][match2, : 2]

# RANSAC implementation
H, inliers = pydegensac.findHomography(keypoints_left, keypoints_right,
  ↪  10.0, 0.99, 10000)
```

B.4. Results

B.4.1. Initialisation

```python
# Import libraries
import pandas as pd
import numpy as np
from tqdm import tqdm
import sklearn.metrics as skm
import matplotlib.pyplot as plt
import seaborn as sns


# Read CSV file
data = pd.read_csv(csv_path, sep=',', quotechar='"', encoding='utf8',
  ↪ header='infer')

# Load image entries
img1 = data['img1']
img2 = data['img2']

# Load Inlier values
x = data['inliers']

# Calculate and load ground-truths
y = pd.Series(0, index=np.arange(len(img1)))

for i in np.arange(len(img1)):
    if img1[i][:16] == img2[i][:16]:  # if first 16 characters in string
      ↪ are equal
        y[i] = 1

# For-Loop to calculate results for a given threshold value
def get_prediction_inliers(threshold, inliers):
  y_pred = pd.Series(0, index=np.arange(len(inliers)))
  for entry in range(len(inliers)):
      if inliers[entry] >= threshold:
        y_pred[entry] = 1
      else:
        y_pred[entry] = 0
  return y_pred

# Setup data-split in training and testing
from sklearn.model_selection import train_test_split

# 0.7 training and 0.3 testing
x_train, x_test, y_train, y_test = train_test_split(x, y,
  ↪ test_size=0.3,random_state=4)
x_train = x_train.to_numpy()
x_test = x_test.to_numpy()
y_train = y_train.to_numpy()
y_test = y_test.to_numpy()
```

B.4.2. Calculation

```
# Calculate accuracy, bal. accuracy, precision and recall for threshold values 0-100 inliers
iterator = range(100)
accuracy = [skm.accuracy_score(y_train, get_prediction_inliers(tr, x_train)) for tr in tqdm(iterator, position=0, leave=True) ]
b_accuracy = [skm.balanced_accuracy_score(y_train, get_prediction_inliers(tr, x_train)) for tr in tqdm(iterator, position=0, leave=True)]
precision = [skm.precision_score(y_train, get_prediction_inliers(tr, x_train)) for tr in tqdm(iterator, position=0, leave=True) ]
recall = [skm.recall_score(y_train, get_prediction_inliers(tr, x_train)) for tr in tqdm(iterator, position=0, leave=True) ]

from sklearn import metrics
metric = accuracy
# Calculate performance on training subset
print(”Accuracy:”,metrics.accuracy_score(y_train, get_prediction_inliers(np.argmax(metric), x_train)))
print(”Precision:”,metrics.precision_score(y_train, get_prediction_inliers(np.argmax(metric), x_train)))
print(”Recall:”,metrics.recall_score(y_train, get_prediction_inliers(np.argmax(metric), x_train)))
print(”Balanced accuracy:”, metrics.balanced_accuracy_score(y_train, get_prediction_inliers(np.argmax(metric), x_train)))

# Calculate performance on testing subset
print(”Accuracy:”,metrics.accuracy_score(y_test, get_prediction_inliers(np.argmax(metric), x_test)))
print(”Precision:”,metrics.precision_score(y_test, get_prediction_inliers(np.argmax(metric), x_test)))
print(”Recall:”,metrics.recall_score(y_test, get_prediction_inliers(np.argmax(metric), x_test)))
print(”Balanced accuracy:”, metrics.balanced_accuracy_score(y_test, get_prediction_inliers(np.argmax(metric), x_test)))
```

B.4.3. Plotting

```python
# Plot the above scores
data = [accuracy, b_accuracy, precision, recall]
plots = ['Accuracy', 'Balanced Accuracy', 'Precision', 'Recall']
fig, axs= plt.subplots(1, 4)
fig.set_size_inches(30, 30/5-1, forward=True)
for i in range(len(plots)):
  axs[i].step(iterator, data[i], where='post')
  axs[i].set_title(plots[i] + ” Curve”, fontsize=18)
  axs[i].set_ylabel(plots[i], fontsize=18)
  axs[i].set_xlabel('Inlier threshold value', fontsize=18)
  axs[i].scatter(np.argmax(accuracy), data[i][np.argmax(accuracy)],
    ↪ marker='o', color='orange', label='Threshold')
  axs[i].set_xlim((0,31))
  axs[i].axvline(np.argmax(accuracy), ymin=0,
    ↪ ymax=data[i][np.argmax(accuracy)], color='orange', ls='--')
  axs[i].set_xticks(np.arange(0, 31, step=5))
  axs[i].set_yticks(np.arange(0, 1+0.1, step=0.1))
  axs[i].tick_params(axis='both', which='major', labelsize=12)
  axs[i].legend()
  axs[i].grid()
plt.tight_layout()

plt.savefig('dataset_performance.svg')

# Precision-recall curve on the testing subset
fig, ax = plt.subplots()
plt.xlim((0,1))
plt_recall = np.insert(recall,-1,0, axis=0)
plt_precision = np.insert(precision,-1,1, axis=0)
plt.step(plt_recall, plt_precision, where='post')
plt.scatter(recall[np.argmax(metric)], precision[np.argmax(metric)],
  ↪ marker='o', color='orange', label='Optimal threshold')
plt.axhline(precision[np.argmax(metric)], xmin=0,
  ↪ xmax=recall[np.argmax(metric)], color='orange', ls='--')
plt.axvline(recall[np.argmax(metric)], ymin=0,
  ↪ ymax=precision[np.argmax(metric)], color='orange', ls='--')
plt.xticks(np.arange(0, 1+0.1, step=0.1))
plt.yticks(np.arange(0, 1+0.1, step=0.1))
plt.tick_params(axis='both', which='major', labelsize=12)
plt.title('Precision-Recall curve')
plt.xlabel('Recall')
plt.ylabel('Precision')

plt.grid()
plt.legend()

# show the plot
plt.rcParams[”figure.figsize”] = (6, 5)
plt.tight_layout()
plt.savefig('precision_recall_curve.svg')
plt.show()
```

B.4.4. Dataset

```python
import os
import pandas as pd
import glob
from os import listdir
from os.path import isfile, join
arr = os.listdir()
writer = pd.ExcelWriter('folderlist.xlsx', engine='xlsxwriter')
jpgs = glob.glob("*/*.jpg")
frame = pd.DataFrame({
    'img1': [],
    'img2': [],
    'match': [],
            })
for i in range(len(jpgs)):
    for j in range(i+1, len(jpgs)):
        if jpgs[i] != jpgs[j]:
            dirname1 = os.path.dirname(jpgs[i])
            dirname2 = os.path.dirname(jpgs[j])
            bool = (dirname1==dirname2)
            new_row = {'img1': str(jpgs[i]), 'img2': str(jpgs[j]),
             ↪  'match': bool}
            frame = frame.append(new_row, ignore_index=True)



# Convert the dataframe to an XlsxWriter Excel object.
frame.to_excel(writer, sheet_name='Sheet1', index=False)

# Close the Pandas Excel writer and output the Excel file.
writer.save()
```

Bibliography

[1] M. Deutman, P. Groet, and O. van Hooff, *Image search engine for digital history: Standard approach*, 2021.

[2] A. Nanetti, *Engineering Historical Memory*, https://engineeringhistoricalmemory.com, Accessed on: 21/05/2021, 2021.

[3] A. Nanetti, "Defining heritage science: A consilience pathway to treasuring the complexity of inheritable human experiences through historical method, ai, and ml," *Complexity*, vol. 2021, p. 4 703 820, 2021, ISSN: 1076-2787. DOI: 10.1155/2021/4703820.

[4] B. Seguin, "The replica project: Building a visual search engine for art historians," *XRDS*, vol. 24, no. 3, pp. 24–29, 2018. DOI: 10.1145/3186653.

[5] B. Seguin, C. Striolo, I. diLenardo, and F. Kaplan, "Visual link retrieval in a database of paintings," *Springer International Publishing 2016*, vol. 9913, no. 1, pp. 753–767, 2016. DOI: 10.1007/978-3-319-46604-0_52.

[6] 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.

[7] L. Schomaker, *A large-scale field test on word-image classification in large historical document collections using a traditional and two deep-learning methods*, 2019. arXiv: 1904.08421 [cs.CV].

[8] T. van der Zant, L. Schomaker, S. Zinger, and H. van Schie, "Where are the search engines for handwritten documents?" *Interdisciplinary Science Reviews*, vol. 34, no. 2-3, pp. 224–235, 2009. DOI: 10.1179/174327909X441126.

[9] T. M. Rath, R. Manmatha, and V. Lavrenko, "A search engine for historical manuscript images," in *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '04, Sheffield, United Kingdom: Association for Computing Machinery, 2004, pp. 369–376, ISBN: 1581138814. DOI: 10.1145/1008992.1009056.

[10] D. Michaud, T. Urruty, P. Carré, and F. Lecellier, "Adaptive features selection for expert datasets: A cultural heritage application," *Signal Processing: Image Communication*, vol. 67, pp. 161–170, 2018, ISSN: 0923-5965. DOI: 10.1016/j.image.2018.06.011.

[11] R. Veltkamp and M. Tanase, "Content-based image retrieval systems: A survey," *Technical report, Utrecht University*, Nov. 2000.

[12] M. Hamilton, S. Fu, M. Lu, J. Bui, D. Bopp, Z. Chen, F. Tran, M. Wang, M. Rogers, L. Zhang, C. Hoder, and W. T. Freeman, *Mosaic: Finding artistic connections across culture with conditional image retrieval*, 2021. arXiv: 2007.07177 [cs.LG].

[13] R. Inbaraj and G. Ravi, "A survey on recent trends in content based image retrieval system," *Journal of Critical Reviews*, vol. 7, no. 11, pp. 961–965, 2020.

[14] M. Yasmin, S. Mohsin, and M. Sharif, "Intelligent image retrieval techniques: A survey," *Journal of Applied Research and Technology*, vol. 12, no. 1, pp. 87–103, 2014, ISSN: 1665-6423. DOI: 10.1016/S1665-6423(14)71609-8.

[15] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 9, pp. 1627–1645, 2009.

[16] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Object retrieval with large vocabularies and fast spatial matching," in *2007 IEEE conference on computer vision and pattern recognition*, IEEE, 2007, pp. 1–8.

[17] J. Yang, Y.-G. Jiang, A. G. Hauptmann, and C.-W. Ngo, "Evaluating bag-of-visual-words representations in scene classification," ser. MIR '07, Augsburg, Bavaria, Germany: Association for Computing Machinery, 2007, pp. 197–206, ISBN: 9781595937780. DOI: 10.1145/1290082.1290111.

[18] M. van Geerenstein, P. van Mastrigt, and L. Vergroesen, *An inquiry into the ethics and technology of artificial intelligence*, 2021.

[19] M. Deutman, P. Groet, and O. van Hooff, *An inquiry into the ethics and technology of search engines*, 2021.

[20] Q. Jia, J. Cai, Z. Cao, Y. Wu, X. Zhao, and J. Yu, “Deep learning for object detection and grasping: A survey,” in *2018 IEEE International Conference on Information and Automation (ICIA)*, 2018, pp. 427–432. DOI: 10.1109/ICInfA.2018.8812318.

[21] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, “Object detection with deep learning: A review,” *IEEE transactions on neural networks and learning systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[22] J. Ma, X. Jiang, A. Fan, J. Jiang, and J. Yan, “Image matching from handcrafted to deep features: A survey,” *International Journal of Computer Vision*, vol. 129, no. 1, pp. 23–79, 2021, ISSN: 1573-1405. DOI: 10.1007/s11263-020-01359-2.

[23] R. N. Luces, *Template-based versus feature-based template matching*. [Online]. Available: https://medium.datadriveninvestor.com/template-based-versus-feature-based-template-matching-e6e77b2a3b3a.

[24] *The opencv reference manual*, 3.4, OpenCV, Jun. 2021.

[25] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[26] O. Chum, J. Matas, and J. Kittler, “Locally optimized ransac,” in *Pattern Recognition*, 2003.

[27] O. Chum, T. Werner, and J. Matas, “Two-view geometry estimation unaffected by a dominant plane,” in *CVPR*, 2005.

[28] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors, “Scikit-image: Image processing in Python,” *PeerJ*, vol. 2, e453, Jun. 2014, ISSN: 2167-8359. DOI: 10.7717/peerj.453.

[29] J. Cheng, Y. Wu, W. Abd-Almageed, and P. Natarajan, “QATM: quality-aware template matching for deep learning,” *CoRR*, vol. abs/1903.07254, 2019. arXiv: 1903.07254. [Online]. Available: http://arxiv.org/abs/1903.07254.

[30] A. Krizhevsky and G. E. Hinton, “Using very deep autoencoders for content-based image retrieval.,” in *ESANN*, Citeseer, vol. 1, 2011, p. 2.

[31] 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)*, IEEE, 2019, pp. 1–5.

[32] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2015. arXiv: 1409.1556 [cs.CV].

[33] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

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

[35] 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. DOI: 10.1109/CVPR.2019.00828.

[36] I. E. Lager, K. Bertels, V. Scholten, E. Bol, C. Richie, and S. Izadkhast, *EE3L11 Bachelor Graduation Project*, 2020-2021. Delft University of Technology, 2020.

[37] H. H. Wang, D. Mohamad, and N. A. Ismail, *Approaches, challenges and future direction of image retrieval*, 2010. arXiv: 1006.4568 [cs.IR].

[38] R. Brunelli, *Template Matching Techniques in Computer Vision: Theory and Practice*. Mar. 2009, ISBN: 978-0-470-51706-2. DOI: 10.1002/9780470744055.

[39] A. Gordo, J. Almazan, J. Revaud, and D. Larlus, *End-to-end learning of deep visual representations for image retrieval*, 2017. arXiv: 1610.07940 [cs.CV].

[40] M. S. Aksoy, O. Torkul, and I. H. Cedimoglu, “An industrial visual inspection system that uses inductive learning,” *Journal of Intelligent Manufacturing*, vol. 15, no. 4, pp. 569–574, 2004, ISSN: 1572-8145. DOI: 10.1023/B:JIMS.0000034120.86709.8c.

[41] A. Del Bimbo and P. Pala, “Visual image retrieval by elastic matching of user sketches,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 2, pp. 121–132, 1997. DOI: 10.1109/34.574790.

[42] I. Talmi, R. Mechrez, and L. Zelnik-Manor, *Template matching with deformable diversity similarity*, 2017. arXiv: 1612.02190 [cs.CV].

[43] K. Ito, A. Morita, T. Aoki, H. Nakajima, K. Kobayashi, and T. Higuchi, “A fingerprint recognition algorithm combining phase-based image matching and feature-based matching,” in *International Conference on Biometrics*, Springer, 2006, pp. 316–325.

[44] B. Klare and A. K. Jain, “Sketch-to-photo matching: A feature-based approach,” in *Biometric technology for human identification VII*, International Society for Optics and Photonics, vol. 7667, 2010, p. 766 702.

[45] Q. Li, W. Cai, X. Wang, Y. Zhou, D. D. Feng, and M. Chen, “Medical image classification with convolutional neural network,” in *2014 13th international conference on control automation robotics & vision (ICARCV)*, IEEE, 2014, pp. 844–848.

[46] S. R. Dubey, “A decade survey of content based image retrieval using deep learning,” *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 1–1, 2021, ISSN: 1558-2205. DOI: 10.1109/tcsvt.2021.3080920.

[47] Y. Jin, D. Mishkin, A. Mishchuk, J. Matas, P. Fua, K. M. Yi, and E. Trulls, “Image matching across wide baselines: From paper to practice,” *International Journal of Computer Vision*, vol. 129, no. 2, pp. 517–547, Oct. 2020, ISSN: 1573-1405. DOI: 10.1007/s11263-020-01385-0.

[48] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

[49] G. A. Miller, “Wordnet: A lexical database for english,” *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[50] X. Chen, *Image enhancement effect on the performance of convolutional neural networks*, 2019.

[51] M. A. Fischler and R. C. Bolles, “Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography,” *Commun. ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981, ISSN: 0001-0782. DOI: 10.1145/358669.358692.

[52] D. Mishkin, J. Matas, and M. Perdoch, “Mods: Fast and robust method for two-view matching,” *Computer Vision and Image Understanding*, 2015, ISSN: 1077-3142. DOI: 10.1016/j.cviu.2015.08.005.

[53] D. P. Vassileios Balntas Edgar Riba and K. Mikolajczyk, “Learning local feature descriptors with triplets and shallow convolutional neural networks,” in *Proceedings of the British Machine Vision Conference (BMVC)*, E. R. H. Richard C. Wilson and W. A. P. Smith, Eds., BMVA Press, Sep. 2016, pp. 119.1–119.11, ISBN: 1-901725-59-6. DOI: 10.5244/C.30.119.

[54] J. Watt, R. Borhani, and A. K. Katsaggelos, *Machine Learning Refined: Foundations, Algorithms, and Applications*, 2nd ed. Cambridge University Press, 2020. DOI: 10.1017/9781108690935.

[55] J. Revaud, J. Almazán, R. S. Rezende, and C. R. d. Souza, “Learning with average precision: Training image retrieval with a listwise loss,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 5107–5116.

[56] J. Revaud, R. de Rezende, C. de Souza, D. Larlus, and J. Almazan, *Deep image retrieval*. [Online]. Available: https://github.com/naver/deep-image-retrieval.

[57] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[58] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden, “Pyramid methods in image processing,” *RCA engineer*, vol. 29, no. 6, pp. 33–41, 1984.