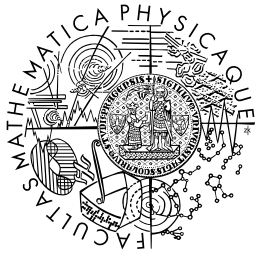


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Jana Bátoryová

Searching Image Collections Using Deep Representations of Local Regions

Department of Software Engineering

Supervisor of the master thesis: doc. RNDr. Jakub Lokoč, Ph.D.
Study programme: Computer Science
Study branch: Artificial Intelligence

Prague 2020

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

Firstly, I would like to thank my supervisor doc. RNDr. Jakub Lokoč, Ph.D. for ideas, numerous corrections, and spot-on help. My thanks also go to the SIRET Research group members for sharing their experience and knowledge in the research topic with me. Last but not least, I thank all the faculty members and especially fellow students. They created an amazing growth environment where I could learn. Personal thanks go to my close friends and family, who helped me persist in my studies. Finally, I wish to express my deepest gratitude to my partner, for the enormous support he gave me.

Title: Searching Image Collections Using Deep Representations of Local Regions

Author: Jana Bátoryová

Department: Department of Software Engineering

Supervisor: doc. RNDr. Jakub Lokoč, Ph.D., Department of Software Engineering

Abstract: In a known-item search task (KIS), the goal is to find a previously seen image in a multimedia collection. In this thesis, we discuss two different approaches based on the visual description of the image.

In the first one, the user creates a collage of images (using images from an external search engine), based on which we provide the most similar results from the dataset. Our experiments show that we can provide more relevant results when incorporating spatial information by splitting the images into several parts. We compared this approach to a baseline, which does not utilize this spatial information and an approach that alters a layer in a deep neural network.

We also present an alternative approach to the KIS task, search by faces. In this approach, we work with the faces extracted from the images. We investigate face representations for their ability to sort people based on their similarity. Based on these similarities, we build a structure that allows easy exploration of the set of faces.

We provide a demo, implementing all presented techniques.

Keywords: Known-item search Convolutional neural network Content-based image retrieval Multimedia exploration

Contents

Introduction

In the last decades, we witnessed a massive increase in the amount of digital information owned by individuals. Looking back 20–30 years, people used to record only a few hours of their lives, capturing precious moments. Now, according to available estimates, more than 500 hours of videos are uploaded every minute to YouTube[1] only. Furthermore, decreasing prices and the increasing availability of the recording electronics (especially smartphones) contribute to the amount of multimedia data created every day. It has also become a trend to share videos from day-to-day lives.

The significant increase in the volume of multimedia data opens several new challenges. One of the most vital is the need for effective search and retrieval of stored data. This problem is not only attractive to researchers, but the initiative also comes from the industry. Companies try to help their customers to organize a vast amount of multimedia data (Google Photos, Facebook, OneDrive, MEGA, and others). Those companies often rely on a broad spectrum of techniques used to store and organize the data internally. Unfortunately, users are usually only provided with the most straightforward techniques to filter the data.

Besides attempting to overcome the challenges of querying large volumes of stored data, we will focus on a scenario in which a user searches for one specific image in a dataset. This task of searching for a previously seen multimedia object is often referred to as a visual known-item search (KIS). In this thesis, we investigate several known-item search techniques where users provide a few example images as a collage query or browse through images of faces organized in a grid with respect to their similarity.

Known-item search has become a well-known research area [Lokoč et al., 2018]. According to recent findings [Rossetto et al., 2020], most of the known-item search engines incorporate both querying and interactive search functionality. In order to improve the level of developed KIS systems, researchers organize and participate in annual competitions. These efforts help to increase the interest in user-centered multimedia search. One, for example, is Video Browser Showdown[2], abbreviated as VBS. For a comparison, TRECVid [Awad et al., 2019] is also an annual competition with the main focus on the ranking of scenes based on a textual description.

In this thesis, we investigate two approaches to solve the known-item search task:

1. Searching by an image collage query.
2. Searching by browsing in a set of faces from the dataset.

In the first approach, we focus on searching known images by using only images from other sources and their position on a canvas. Users can create a collage of images that reminds them of the searched scene and then browse through the ranked result list looking for the match.

[1] Statista – Hours of video uploaded to YouTube every minute

[2] `https://videobrowsershowdown.org/`

The second approach is a traversal system over the dataset organized by visual similarity of human faces. This brings two partially contradicting goals. On the one hand, we want to show enough information to the user so they could navigate in the dataset. On the other, we do not want to overwhelm them by the sheer amount of faces in the dataset. We tackle this challenge by organizing the faces into a multilevel view supporting navigational queries. We implemented the traversal system and tested its abilities.

The goal of this thesis is to test and implement both approaches, as mentioned earlier. We aim to create a user-friendly interface for smooth user–system interaction.

For the techniques which search by collages, we provide evaluations on an annotated set of queries. We manually created a set of collages, as our queries, and use them to test the proposed system and fine-tune the hyperparameters. We evaluate the proposed methods based on the rank of the target image. Well-performing methods bring the searched items to the foreground, placing them at the top ranks.

We evaluate the second approach — search based on face similarities — with the help of a case study. Firstly, we investigate the space of the feature vectors by evaluating the responses from the survey. Then we present our system for dataset exploration. In the end, we provide a preliminary study, comparing the average time needed to find a target face using our system and using a simple linear search.

Thesis Structure

After the Introduction, we continue by reviewing Related Work (chapter 1) and Technical background (chapter 2). We then continue by defining the task in chapter 3.

Following the overview chapters, we propose our solutions in two chapters — search by collage in chapter 4 and search based on the face similarities in chapter 6. chapter 5 includes evaluations of the approaches mentioned in the chapter 4.

The end of the thesis is dedicated to the implementation of the aforementioned approaches. This includes the user's guide (chapter 7) – how to interact with the system, and the guide how to run the application on a custom dataset (chapter 8). We present the implementation overview in the Appendix A.

In summary, we designed and successfully tested a promising approach based on collage queries. The most promising approach splits images into multiple parts. We also developed a traversal system for previewing a dataset of faces.

1. Related Work

In this chapter, we review approaches of handling the Known-item Search (KIS) task, and the recent approaches for user-friendly traverse through the immense amount of the visual data.

In recent years, we have witnessed a significant advancement in frameworks focusing on the KIS task. An overview of the Video Browser Showdown evolution in the past few years is summarized in [Lokoč et al., 2018]. The scale of the frameworks' complexity for user interaction ranges from simple ones (e.g., sketching color blocks) with only a few descriptors to the ones that use recent advances in deep learning, such as concept labeling.

Our goal is to perform a known-item search task on a set of images. Developing a successful technique to approach the image KIS task can lead us to solve the KIS task over videos. We can perform the image search over the extracted frames from the videos. In our case, we used videos as a source for our dataset of images. More information on the dataset is presented in section 1.4.

At the end of this chapter, we provide an overview of the face recognition topic. We link to the recent overviews of the most successful systems based on deep neural networks.

1.1 Known-item Search Task

The known-item search task is present in any multimedia format. We may look for a particular article in the database of newspapers or for a specific photo in a family album. This thesis focuses on visual KIS task; specifically, we retrieve images. In this chapter, we review a few of the systems for the visual KIS task presented at the last VBS2019. We mainly follow the summary from Rossetto et al. [2020]. As the study presents, the most popular approach at the VBS2019 was Query by an Image and Concept Labeling.

Query by an Image in this case mostly refers to finding the most similar results from a database to a given image. The downside of this approach is the difficulty of obtaining an image that is sufficiently similar to the searched one.

Another widely used approach at VBS2019 was Concept Labeling. In this case, the user describes the image by words. Firstly, the items in the database are annotated by a variety of automatic tools. Then for each textual input query, the system checks the database for the presence of images with assigned labels matching the query. Concept Labeling often faces a limitation on the vocabulary size, and accuracy issues caused by a classification misses. Recent advancements in the textual annotation by neural networks helped to tackle this problem. Nowadays, automatic annotation systems are able to classify thousands of different entities. However, even in so many classes, we may not find a word to describe rarely used objects.

One of the other presented approaches is creating a color sketch. The user draws the sketches on the canvas, which resembles the searched scene. The database is then searched for images with corresponding color patches. As a significant advantage of this approach, we see its ability to incorporate spatial information by searching for a specific color in a specific region of the image.

Solving the KIS task in the VBS setting offers the option of using full video information and not only separate images. This allows the usage of additional techniques such as Temporal Queries or Multimodal Queries. Also, some of the frameworks presented included Optical Character Recognition (OCR).

We designed our techniques as a possible enhancement for a complex system in order to support multiple search strategies based on a user preference. At the same time, we present a standalone application to preview the described techniques.

1.1.1 Traversal Approaches

KIS task is the task of two sides — the algorithm and the user. It is essential not to forget about the user experience. Smooth user experience with quick navigation over a dataset can help to find the target image faster. As the review of the VBS by Rossetto et al. [2020] shows, the most common approach is to show a 2D grid of frames to the user. Several approaches also provide an easy way to play the original video.

The traversal systems often rely on effective visualization techniques for high-dimensional data. Many of the frameworks presented at VBS create a 2D grid of images, where the images are usually organized based on their high-dimensional representation, often produced by neural networks.

1.2 Overview of the Existing Frameworks

In the next section, we shortly investigate some of the existing frameworks, which competed in the Video Browser Showdown.

1.2.1 VIRET

A framework named VIRET [Lokoč et al., 2019a,b](see Figure 1.1) has been successfully participating in competitions for several years now. The framework has evolved throughout the years, and now it offers a wide variety of strategies for solving the KIS task. VIRET also implements its own strategy for a frame selection, which we also used for obtaining our images. As one of the most significant strategical advantages of the VIRET, we consider the option to use Multimodal and Temporal Queries. Some of the features included in the tool are: search by color regions, video playback, keyword search, and search by an example image. For the purposes of the Lifelog Search Challenge [Gurrin et al., 2020] the videos can also be searched by the meta-attributes (e.g., day of the week, time of the day). We take inspiration from VIRET, although we do not implement the same approaches as they are already present in the VIRET. We focus on testing new alternatives.

We highly recommend a more thorough description of the VIRET tool presented in Kovalčík et al. [2020].

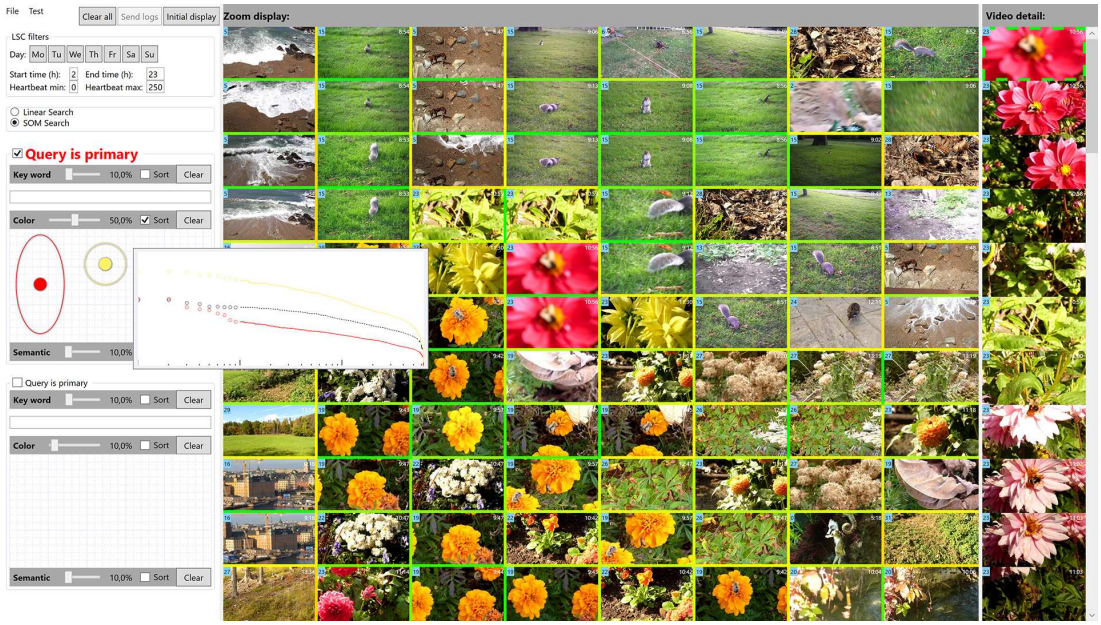


Figure 1.1: Example search in VIRET framework. Source: Kovalčík et al. [2020]

1.2.2 SOM-Hunter

A SOM-Hunter was for the first time introduced at the VBS2020. This tool is related to our approach because it also supports Self-organizing Maps to solve a Known-item search task.

The typical workflow (as described in Kratochvíl et al. [2020]) starts with a search based on the keywords. This reduces the dataset only to relevant images. During the entire browsing session, they preserve the relevance score for each image. To display the results, they use a mapping onto a trained Self-organizing Map (where samples are sampled with the weight of the relevance scores). The second type of view they support is a grid view, where the images are sorted by the relevance scores. The user can continue to explore the dataset further by browsing or reformulating the query. Since the showed displays are relatively small (8×8 images when using a SOM), the corresponding self-organizing map can be computed quickly. The user can also explore the temporal context of any of the images. A sample interaction with the system is displayed in Figure 1.2.

1.2.3 Vitrivr

Vitrivr [Rossetto et al., 2016b](see Figure 1.3) is a sophisticated framework for retrieving a specific video in a collection of videos. Vitrivr is separated into three modules, Vitrivr-ng, Cineast, and Cottontail-DB. Vitrivr-ng is the module responsible for creating queries that are later processed by the Cineast. Cottontail-DB supports fast retrieving methods of the features required by Cineast. The overview of the system is displayed in figure 1.3. Vitrivr supports different query types: query by a sketch, as well as an example image, semantic concept, keywords, audio, or motion. The feature extraction and the system behind retrieving the most similar results are parts of the second module — Cineast (Rossetto et al. [2016a]). Cineast uses multiple approaches incorporating deep features, e.g., scene text recognition and speech-to-text recognition. Vitrivr-ng (the part responsible

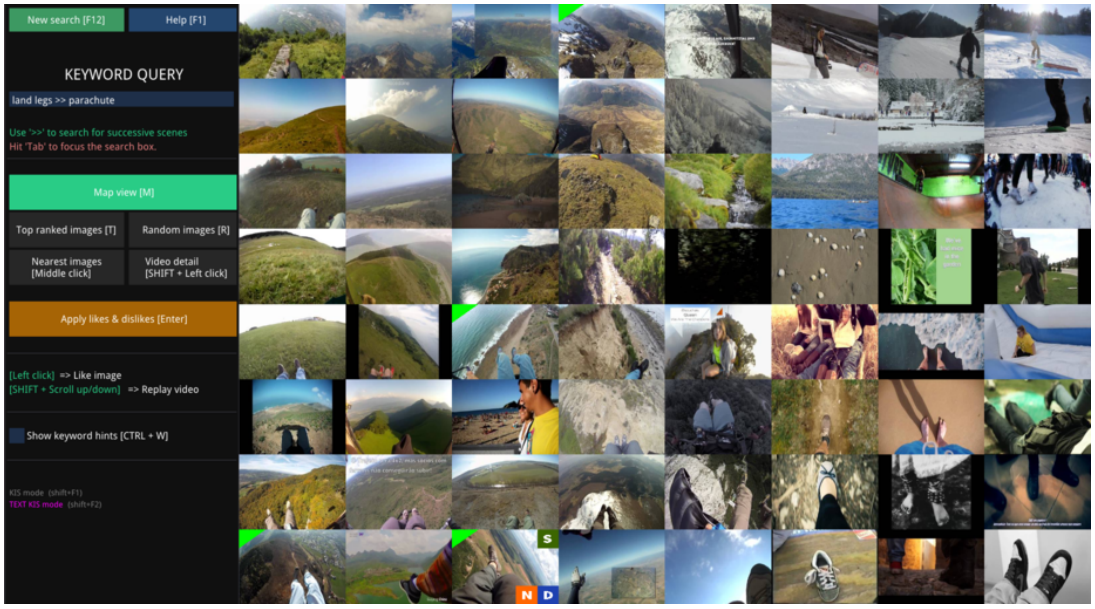


Figure 1.2: A sample search in SOM-Hunter. Source: Video Broser Showdown, Hall of Fame

for creating the queries) is a web-based software. The modules structure provides a clean separation between the query formulation and feature extractions.

1.3 Face Recognition

In the second part of the thesis, our goal is to organize faces to allow a faster search through the dataset. We leverage the face recognition technology to organize the faces into a hierarchical structure. One such approach is presented in [Girgensohn et al., 2004]. The authors' algorithm provides users with faces based on the similarity to already retrieved people. They examined the performance of the model by a series of simulations.

In recent years, many steps have been taken towards solving the face recognition task. As the Ranjan et al. [2018] overviews, three separate modules are typically needed for automatic verification and identification systems.

Firstly, a face detector is applied to localize faces in images. A robust detector should be able to detect faces with a varying pose, illumination, and scale. Also, the bounding boxes of the detected faces should be minimized, to contain a minimal amount of background.

Secondly, a landmark detector is usually incorporated. This detector localizes the important facial landmarks such as nose tip, mouth corners, etc. These points are then used to rotate and scale the face, which creates a normalized input for the next phase.

Lastly, a feature descriptor encodes information from the aligned face. Based on these encodings and various similarity measures, the model can signify if the two faces belong to the same person or perform other related tasks.

The face recognition problem can be further split into two categories: face verification and face identification. In the face verification task, the goal is to determine whether two subjected faces belong to the same person. In the iden-

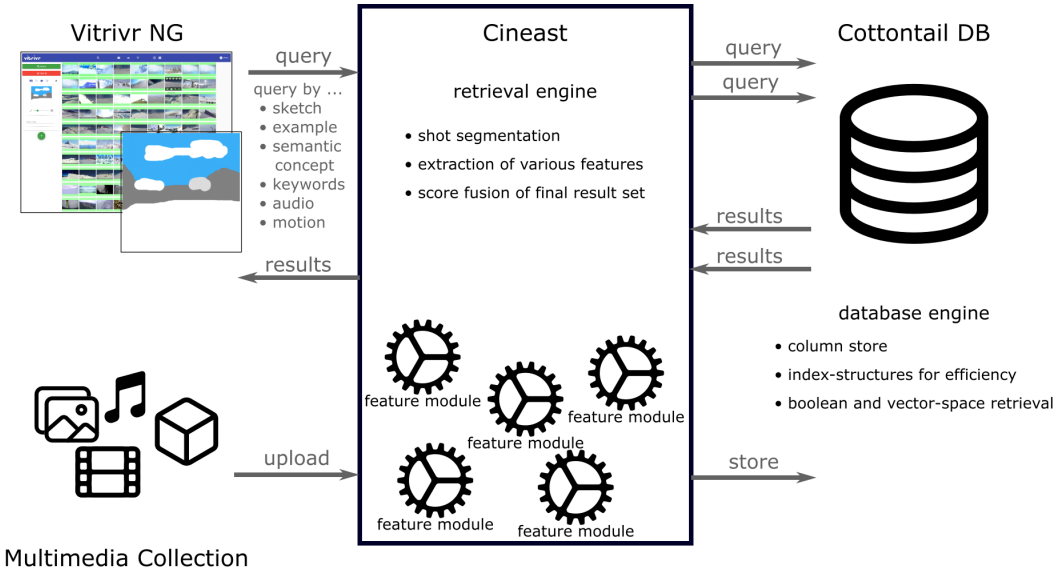


Figure 1.3: Overview of the separation in the Vitrivr framework. Source: `https://vitrivr.org/vitrivr.html`

tification scenario, a set of known subjects is inserted into the system. Then, a new subject (target) is presented. The goal of the model is to retrieve all faces from the dataset belonging to this person, or in a supervised manner, to assign the identity.

Since the early 1990s, many face identification/verification systems were proposed. For example, in [Kalocsai et al., 1998], authors conducted a study, where two faces were shown to the respondents, simulating the verification task. Then, the respondents were asked to decide if the faces belong to the same person. The faces displayed different emotions. The researchers suggest that the model's performance correlates with the human's ability to recognize people.

Twenty-five years later, with the advancements of the deep neural networks, new state-of-art methods achieve an almost perfect score on some of the benchmarks (e.g., [Huang et al., 2008]). We refer to the [Masi et al., 2018] to provide an overview of the recent models used for face identification and verification. The summary also provides a thorough description of the individual steps usually taken by the approaches presented. In our work, we use pre-trained models provided by the dlib library. We include more technical information in subsection 2.1.5.

1.4 Dataset

We use Vimeo Creative Commons Collections (V3C1)[1] dataset for experimenting and evaluations. This dataset is used for evaluations at VBS and also at TRECVID. The dataset is composed of 7475 Vimeo videos. We selected only the first 750 videos for proving concepts of our work. From these 750 videos, 111 764 images were extracted with resolution 320×180 by using an extraction tool from

[1]TRECVID 2019 Video Data

Lokoč et al. [2019a]. I extend my gratitude to Tomáš Souček and Gregor Kovačík for providing extracted images.

The videos capture a wide range of sceneries on many different occasions. We can see many different landscapes, from seas to mountain views, from desert to snow. A large proportion of the videos contain people. Videos capture people doing different activities, e.g., skateboarding in a park, Hindi wedding, or a news broadcast.

2. Theoretical and Technical Background

This chapter intends to review the fundamentals of the theoretical approaches used in this thesis. It includes the fundamental concepts of neural networks and a short description of the networks we use. The chapter also reviews projection methods of high-dimensional data to lower-dimensional space.

2.1 Deep Neural Networks

In recent years, we have witnessed the emergence of many record-breaking machine learning models. Many of those breakthroughs were possible, thanks to the advancement of Deep Neural Networks (DNN). Nowadays, these models replaced more traditional Machine Learning approaches in many tasks.

Deep Neural Network is a machine learning model, whose goal is to approximate a given function f. The set of its parameters is often referred to as θ. One of the everyday tasks performed by these networks is classification, where the goal of the network is to predict to which category sample X belongs to. Even though we will not perform a classification task in this thesis, we will use some of the available classification networks.

When we talk about neural networks, we usually refer to a feedforward network. These networks consist of layers that only pass information in one direction during the evaluation. We can imagine it as applying a function to the results from the previous layer. For example, let us create a small neural network. Denote first layer as f_1 and second layer as f_2. The output of the network will be $f_2\left(f_1\left(\cdot\right)\right)$.

Stacking more and more layers on top of each other leads us to the notation deep neural networks. This notation has no fixed threshold on which networks "deserve" to be called deep. The word "deep" separates the eras between the neural networks consisting of a couple of layers from neural networks consisting of tens or more layers.

The first and last layers are commonly referenced as an input and output layer, respectively. Layers between the input and output layers are usually denoted as hidden layers. Deep neural networks can have four or hundreds of layers, and there are multiple types of operations that the layer can perform. Network Architecture captures the "build order" of the network. It is important to note that there exist many networks with different architectures solving the same task.

There are many reasons for the advancement of neural networks in the past years. One of the crucial stones was not only theoretical innovations used for the networks but increasing computability limits. Deep Neural Networks learn to approximate function by training. This training is usually the heaviest part of the computation. Even though it is enough to train it once and use it forever, it usually lays some limitations on the size of the networks or the functions used.

Since this topic is broad, we recommend more thorough reading, such as Neural Networks and Deep learning online book [Nielsen, 2015].

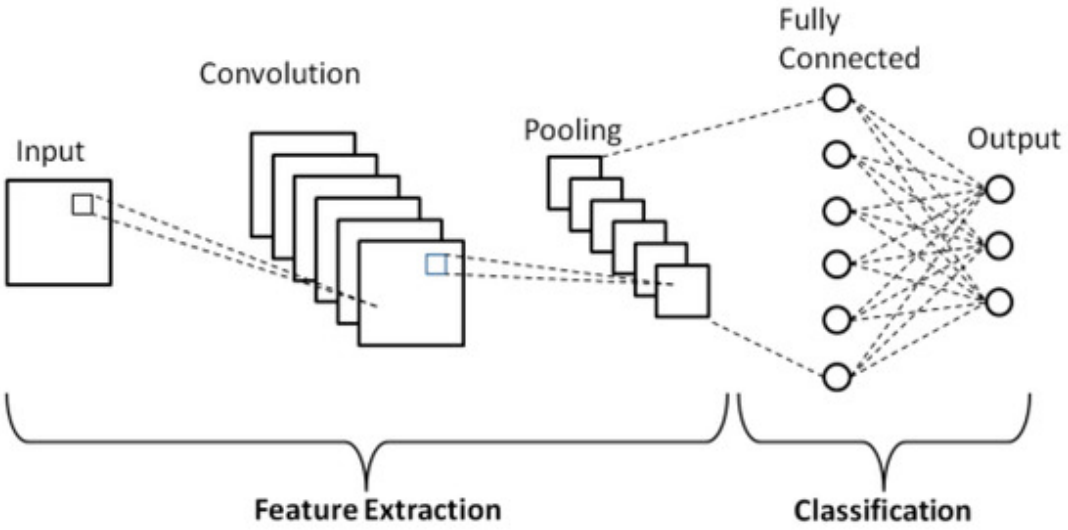


Figure 2.1: Schematic diagram of a convolutional neural network. Source: Phung, V.H.; Rhee, E.J. A High-Accuracy Model Average Ensemble of Convolutional Neural Networks for Classification of Cloud Image Patches on Small Datasets. Appl. Sci. 2019, 9, 4500.

2.1.1 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a class of Deep Neural Networks. Even though they emerged in the late 1980s [LeCun et al., 1989], it took another 20 years for further advancements in the research area. Convolutional Neural Networks are mainly used in image-related tasks, such as image classification ("What is on the image?"), object detection ("Where are the objects in the image and what are they?") or even content generation ("Create a new image"). Their abilities were also tested in many, not image-related tasks, e.g., music genre recognition.

Convolutional Neural Networks are specialized kind of networks which usually work with grid-like topology. For the 2D case, most typically image pixels represent the grid. The name convolution refers to using a mathematical linear function convolution in at least one of the layers. A simplified overview of the structure is displayed in figure 2.1. We refer to Goodfellow et al. [2016] for additional information about the convolutional layers.

2.1.2 Transfer Learning

Transfer Learning is a research problem in machine learning that focuses on storing knowledge gained while solving one problem and applying it to a different problem. We have seen many successful transfers of the network architecture and parameters learned to a new task. Transfer learning may help to reduce the cost of the training and often also to overcome an insufficient set of training examples for the new task.

We utilize some of the pre-trained Convolutional Neural Networks. The possibility of using a pre-trained neural network on a different task than they were trained on was explored as early as 2014 by Donahue et al., and many others were able to use this process to acquire better models. Networks we use are

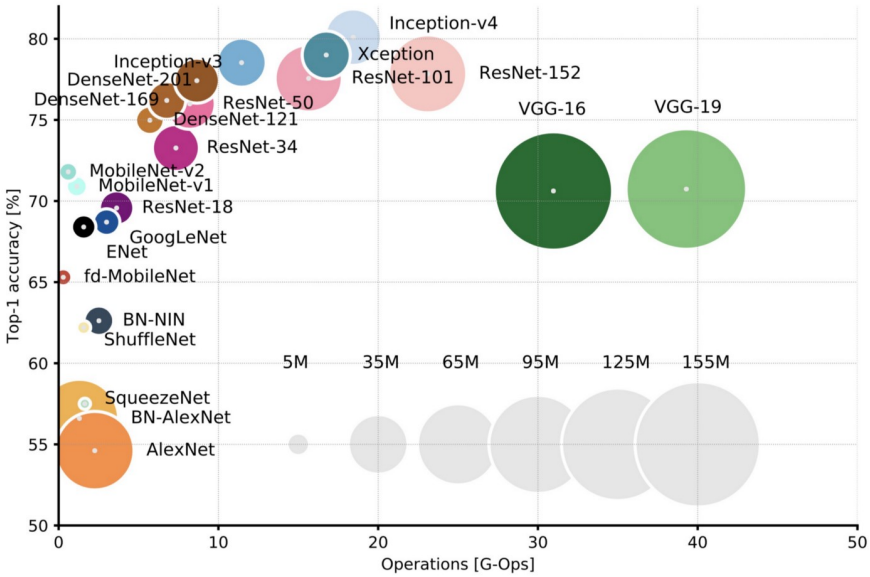


Figure 2.2: Top-1 one-crop accuracy versus amount of operations required for a single forward pass. The size of the blobs is proportional to the number of network parameters. Source: Canziani et al. [2016]

mostly pre-trained on ImageNet[1]. ImageNet serves as one of the benchmarks for comparing the performance of the different networks. Since the ImageNet Large Scale Visual Recognition Challenge [Russakovsky et al., 2015] is a classification task, we utilize the transfer learning to obtain deep features, by stripping the last classification layer. Layers at the end of the networks accumulate semantic information, i.e., they contain high-level features. Therefore, we work with the layers close to the output layer. These layers work with encoded information about the image in high dimensional vectors. Our task is to use obtained deep features for solving our known-item search task.

2.1.3 Pretrained Models

In our work, we use various pre-trained models. We obtain the models from Keras, dlib, or other open-source projects. In the following sections, we describe the frameworks and libraries we use.

2.1.4 Keras

Keras [Chollet et al., 2015] is a deep learning API written in Python, running on top of the machine learning platform TensorFlow [Abadi et al., 2015]. It was developed with a focus on experimentation in deep learning. We use it and its pre-trained models in this thesis. The models, which we use from Keras Applications, were trained on ImageNet. Keras API allows us to remove the last fully connected layer used for classification from the selected neural network.

In this thesis, we use pre-trained models to implement new approaches. We do not aim to train new models or further train the existing ones. We try to extract the information from available models to solve the known-item search task. Here

[1] http://www.image-net.org/

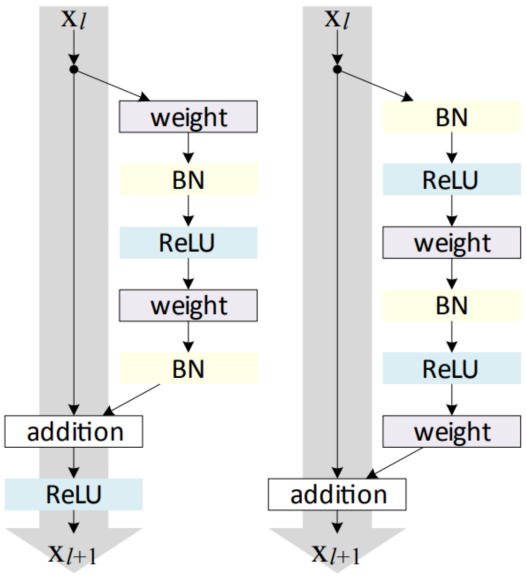


Figure 2.3: Left: ResNet residual unit as proposed in [He et al., 2016a]. Right: ResNetV2 residual unit by [He et al., 2016b]. Source: He et al. [2016b]

we describe two models which we experimented with the most. ResNet was a state of the art model as of 2015. Since then, it gained popularity in many tasks. The second network we focus on is MobileNet. MobileNet has excellent performance to complexity ratio. Therefore, it is an ideal network for our purpose, where the predictions need to be computed online and displayed to the user.

Resnet50V2

ResNet (abbv. for Residual Networks, He et al. [2016a]) is a classic neural network used in many computer vision tasks. ResNet was the winner of the ILSVRC 2015 [Russakovsky et al., 2015]. The authors aimed to solve the problem of degrading the training accuracy of the neural networks when more layers were added.

The authors argued that the reason behind the degradation is that approximating identity function by a layer in a neural network is difficult (otherwise, the added layers should learn to approximate identity, and the resulting error should be no greater than when using shallower counterpart). To solve this problem, instead of trying to approximate underlying mapping $H(x)$, they ask the neural network to approximate residual function $F(x) = H(x) - x$. They do so by replacing simple layers by residual building blocks (see figure 2.3).

In [He et al., 2016b], the authors improved the architecture of ResNet by changing the position of activation a batch normalization layers. By doing so, they made the easiest path for the information to propagate even simpler. For details see the figure 2.3. This improved architecture is commonly known as ResNetV2.

MobileNetV2

In April 2017, a group of researchers from Google published a study [Howard et al., 2017] introducing a new neural network architecture. This MobileNet was

optimized for mobile devices. They optimized the model to deliver high accuracy while keeping the model and mathematical operations as fast as possible. Not a year later, MobileNetV2 was introduced by Sandler et al. [2018]. It extended the predecessor by using Inverted Residuals with Linear Bottlenecks. For more details, please refer to the original research. In our work, we use only MobileNetV2 out of MobileNets.

2.1.5 Dlib

Dlib [King, 2009] is a modern C++ toolkit containing machine learning algorithms and tools for creating complex software in C++ to solve real-world problems. We use the Dlib library for face detection and also face feature extraction. For both, we use Python API provided by `face_recognition` [Geitgey, 2016].

Face Detection

The authors of dlib implemented two key approaches to face detection. The first one, the frontal face detector, is based on the histogram of gradients. This approach is very fast in learning to detect the faces and also fast on performing the detections. It is still used in a variety of online tasks. The original release notes are available in [King, 2017a].

The second approach present in the dlib is based on the convolutional neural network. This approach has a longer inference time. However, based on evaluations, it has higher accuracy in non-frontal face views. The overview of both approaches is provided, for example, by [Ponnusamy, 2018].

Face Encodings

Dlib also presents a model for face encoding extraction. This model was trained for the face verification task. Based on published results (Labeled Faces in the Wild Benchmark), the model achieves 99.38% accuracy on the verification task. The network architecture is based on the ResNet34 and trained on the mix of the available datasets of faces. The review of the model is available at [King, 2017b].

2.2 Principal Component Analysis

In projection methods for dimensionality reduction, the goal is to find a mapping of the input vectors from the original d-dimensional space to a new k-dimensional space (where $k < d$) with minimum information loss. PCA is one of such methods. It is an unsupervised method, which employs linear projection to decrease the number of dimensions while preserving the explanation for variance within the data. Since the derivation of the formulas for PCA requires several technicals steps and some advanced knowledge of Linear Algebra, we refer to the Alpaydin [2020] for a more detailed description.

2.3 Self-organizing Maps

The Self-organizing Map was introduced by Kohonen [1982]. The Self-organizing Map, abbreviated as SOM, is an example of a popular neural network based on unsupervised learning. It produces low-dimension (typically two-dimension), discretized representation of the input space. Additionally, the Self-organizing map structure forms a semantic map, where similar samples are mapped close together and dissimilar ones further apart.

The SOM consists of neurons M organized on a regular low-dimensional grid. Each neuron is represented by a d-dimensional weight vector, where d is equal to the dimension of the input vectors. We shall denote this mapping $\phi : M \to \mathbb{R}^d$. Each neuron is connected to the adjacent neurons by a neighborhood relation, which is induced by the distance between their representations in the low-dimensional space and a threshold. This relation creates the structure of the map.

During the training, the weight vectors are initialized with random values. Then, iteratively, we find the closest weight vector to a data point, called the Best Matching Unit (we denote such mapping $\mu : \mathbb{R}^d \to M$). We shift the weight vector of the BMU, and usually also weight vectors of the neighboring neurons, closer to the data point. The impact on the neighboring neurons is usually scaled by the distance from the BMU.

The SOM can be thought of as net spread across the data. After the training, the neighboring neurons on the grid get similar weight vectors. For a more detailed description, please refer to Kohonen [1982] and Kohonen and Honkela [2007]. Extensive research on the topic of image retrieval using SOM was also presented by Koskela et al. [2003].

Various measures have been developed to quantify a map's quality. In this thesis, we use the following two: quantization error and topographic error. For the motivation behind the errors and more information about them, we refer to Breard [2017] and the original research. Here we present their definition and brief description.

Quantization Error

A quantization error is formulated, for example, in Wandeto and Dresp-Langley [2019]. It is a measure of the average distance between the data points $x_i \in X$ and the map nodes to which they are mapped. The quantization error QE for a map M is calculated as follows:

$$QE(M) = \frac{1}{n} \sum_{i=0}^{n-1} ||\phi(\mu(x_i)) - x_i||$$

where n is the number of samples in the training data D.

Topographic Error

The topographic error was introduced by Kiviluoto [1996]. It accounts for a SOM's ability to preserve local topological features in low dimensional output space. A sample for which the best matching unit and the second-best matching

unit are not adjacent counts as an error. The topographic error is given by the total number of errors divided by the total number of samples.

$$TE(M) = \frac{1}{n} \sum_{i=0}^{n-1} t(x_i)$$

$$t(x) = \begin{cases} 0, & \text{if } \mu(x) \text{ and } \mu'(x) \text{ are neighbors} \\ 1, & \text{otherwise} \end{cases}$$

where $\mu'(x)$ returns the second-best-matching unit.

3. Content-based Image Retrieval

Image retrieval and image indexing have been an active research field since the 1970s. In 1978 (Tamura et al. [1978]), a group of researchers proposed a system for retrieving textures based on the example texture. Since then, a wide range of techniques for image retrieval were presented. Traditional approaches included manual annotation of the images by textual or numerical metadata. The user could then formulate a query against these annotations to retrieve relevant images. This approach is often referred to as Concept-based image retrieval or meta-data search.

There are several drawbacks to the textual or numerical annotations. First of all, extensive human annotations are often needed to provide rich data for the filtering. Also, including spatial information of the objects takes more resources than only writing down objects present in the image. Furthermore, the images often include too many details (i.e., type, color, or shape of the objects), which may be impossible to comprehend by manual annotations. The annotations may not even represent a stable truth. With a different annotator, the annotations may include different details/objects, which were perceived differently. When the user searches for an image, they have to know the exact terms the annotators used in order to be able to retrieve the images they want. As the last problem, we see with human annotations is the scalability. As the amount of information increases every second, there is no human capability to hand-process all the examples.

During the 1990s, content-based image retrieval (CBIR) emerged (trend study from Datta et al. [2008]). In the CBIR approach, the images are indexed by features directly derived from their visual content using automatic or semi-automatic image processing techniques. Such indexing lacks building blocks (for example, verbal description); on the other hand, it provides low-level feature information about the whole images or its regions. The attributes of images are complex functions of regions of the image or the whole image.

CBIR has received considerable research interest in the last decades. With the advancement in Deep Learning, a new pool of possible complex functions to describe the images emerged. In our approaches, we use pre-trained neural networks to extract features. Based on these features, we implemented and evaluated several approaches to the CBIR task.

Presented techniques focus on the Known-item search task. An alternative could be an Ad Hoc search, where the goal is to retrieve all relevant items to the query. Known-item search task instead works with retrieving a known item from the dataset.

In the following chapters, we continue with the specifics of the individual approaches we present. Here we formulate the task and the goal.

3.1 Task Formulation

The goal of the thesis is to propose a system that can search and retrieve images based on visual similarity.

First we define universe $\mathcal{U}$ as the set of the all possible images:

$$\mathcal{U} = \bigcup_{h,w \in \mathbb{N}^2} \{0, 1, 2, \ldots, 255\}^{h \times w \times 3}$$

This definition follows the standard description of an image as a $w \times h$ matrix of pixels, where three color channels describe each pixel. We work over a dataset D of such images, that we can define as $D \subset \mathcal{U}$.

The user is then asked to find a target image $t \in D$, that they have some knowledge of, for example, by (imperfectly) remembering the image or other detailed description. This task is called a known-item search.

Our system then serves as a search engine, aiming to provide the user with the most relevant results based on the user input. The system leverages the visual similarity between the faces in the dataset D (as in chapter 6), or the similarity between user-provided example pictures and the images from the dataset D (as in chapter 4).

3.2 Feature Space

To search over a dataset of images D, we need a metric of how similar two items are. This similarity could be computed between the images directly, although it is more common to compare their descriptors (i.e., corresponding values of the descriptor extraction function). A descriptor extraction function is a function $f_e : \mathcal{U} \to \mathcal{F}$, where $\mathcal{F}$ is a feature space. Given a descriptor extraction function f_e, we compute for each image in the dataset D a descriptor (or feature vector) $L_I = f_e(I)$, where $I \in \mathcal{U}, L_I \in \mathcal{F}$. In our work, we use various neural networks as our descriptor extraction functions. It is common for neural networks to produce the descriptors in the space of real numbers $\mathbb{R}^n$, and in our case, we work only with the descriptor spaces, which are isomorph to the $\mathbb{R}^n$.

3.3 Distance Measures

To compare how similar descriptors of two images are, we use a concept of distance. Similar descriptors have smaller distance and vice versa. We use definitions for a distance space and metric space from the book Deza and Deza [2009].

Definition 3.3.1. Distance space $(\mathcal{F}, \delta)$ is a set $\mathcal{F}$ equipped with a distance $\delta : \mathcal{F} \times \mathcal{F} \to \mathbb{R}_0^+$ satisfying $\forall x, y \in \mathcal{F} : \delta(x, y) \geq 0$ (nonnegativity), $\delta(x, y) = \delta(y, x)$ (symmetry) and $\delta(x, x) = 0$.

Definition 3.3.2. Metric space $(\mathcal{F}, \delta)$ is a distance space, where δ additionaly satisfies $\forall x, y, z \in \mathcal{F} : \delta(x, z) \leq \delta(x, y) + \delta(y, z)$ (triangle inequality) and $\delta(x, y) = 0 \Rightarrow x = y$

In the next chapters, we often compare two high-dimensional descriptors. These descriptors are often produced as an output of a neural network. Our feature space is $\mathbb{R}^n$, given n is the number of features. We present distance functions over the space $\mathbb{R}^n$ later used in the thesis. Euclidean and Manhattan distance on $\mathbb{R}^n$ form metric spaces. Note that, cosine distance, deduced from cosine similarity, does not follow triangle inequality and forms only a distance space.

3.3.1 Euclidean Distance

For given $p, q \in \mathbb{R}^n$ we compute the distance as:

$$d_e(p,q) = \sqrt{\sum_{i=0}^{n-1}(p_i - q_i)^2} \tag{3.1}$$

3.3.2 Manhattan Distance

For given $p, q \in \mathbb{R}^n$ we compute the distance as:

$$d_m(p,q) = \sum_{i=0}^{n-1}|p_i - q_i| \tag{3.2}$$

3.3.3 Cosine Similarity

For given $p, q \in \mathbb{R}^n$ we compute the similarity as:

$$\text{similarity}(p,q) = \cos(p,q) = \frac{pq}{\|p\|\|q\|} = \frac{\sum_{i=0}^{n-1} p_i q_i}{\sqrt{\sum_{i=0}^{n-1} p_i^2}\sqrt{\sum_{i=0}^{n-1} q_i^2}} \tag{3.3}$$

Since the cosine similarity is bounded by one, we use a transformation $d = 1 - s$ to obtain a distance.

$$d_{cos}(\mathbf{p},\mathbf{q}) = 1 - \text{similarity}(\mathbf{p},\mathbf{q}) \tag{3.4}$$

4. Search by Collage

You see a picture in your head. Your friend is standing on the beach, and there is a little sandcastle on the left. The sea in the background beautifully reflects the sun, which is setting.

We can imagine that at that particular moment we were shooting a video of the scenery. However, years later, with a vast set of videos in our collections, it may be impossible to re-watch every one of them to find that particular memory. We present a technique that can be used to search in a dataset based on such memories of the scenery.

In this chapter, we elaborate approaches for Known-item search based on a visual description of the memorized image. We characterize the image by the set of interesting objects in it, the way how they looked (by providing example images) and by their relative location in the image (i.e., top left corner). With that information, we look for a match in the database to the described image. We refer to our input as collage query, or simply just as a collage. Collage is created by taking images and placing them onto an empty canvas. The placement of the images also carries a piece of information. We show an example of such collage (query) in figure 4.1. On the left, we can see a cat in the center behind a window. On the right, we can see a possible visualization of such memory. On the grey canvas, we placed a window, which reminded us of the original one. At the center, we added a similarly colored cat.

While we were describing the image, we used words for it. However, in this chapter, we do not focus on the search based on the textual description of the objects. Compared to such approach, we can capture more diversity in the objects by visual information. For example, a single word for a human may represent a wide range of visually distinct people based on clothes, age, and other attributes. These attributes are easier to capture by providing an example image. We can still struggle to find a similar image; that is why allowing collages is important. It is easier to put several images together, each capturing some of the attributes, rather than finding an exact match.

This chapter presents three approaches incorporating pre-trained neural net-

(a) Target image

(b) One of the possible collage descriptions of the target image

Figure 4.1: Example of searched image (target) with possible visual description by a collage.

works. We start with the formalization of the task. Next, we provide a short description of user–program interaction. Following the user interaction, we provide an overview of the framework and describe individual stages. Each of the stages is then separately discussed.

After the discussion of the options for our system, we evaluate the proposed techniques. We present a set of annotated data — collages for the target images. Using these queries, we test different sets of hyperparameters and investigate their effect on the system's performance.

4.1 Formal Description

In this task, we explore the dataset D based on a collage query provided by a user. We shall formally define universe of query images as follows:

$$\mathcal{Q} = \{(I, x_0, x_1, y_0, y_1) \mid I \in \mathcal{U}, x_0 \in [0, 1), y_0 \in [0, 1), x_1 \in (x_0, 1], y_1 \in (y_0, 1]\}$$

where $\mathcal{U}$ is a universe of images defined in section 3.1. I represents an image in the collage; x_0, y_0 represent the relative position of the top left corner of the image in the canvas; x_1, y_1 represent the relative position of the bottom right corner of the image in the canvas. Single collage Q is then defined as $Q \subset \mathcal{Q}$

Ultimately, we would like to construct a function r^*, that would for each query $Q_i \subset \mathcal{Q}$ find corresponding target image $t_i \in D$. Formally, we can define this function as follows:

$$r^* : \mathcal{P}(\mathcal{Q}) \to D$$
$$r^*(Q_i) = t_i \;\; \forall (Q_i, t_i) \in X$$

where X is a set of target images and corresponding queries constructed by the user.

However, due to imperfect user queries, or by the fact that user can search two different target images using the same collage, this task may be even impossible. Therefore, we focus on reordering the dataset in the way that a linear search by the user in this order would provide the target image as quickly as possible. We define ranking r_D as function with respect to a given dataset D:

$$r_D : (D \times \mathcal{P}(\mathcal{Q})) \to \{0, 1, \ldots, |D| - 1\}$$
$$\text{s.t. } \{r_D(d, Q) \mid d \in D\} = \{0, 1, \ldots, |D| - 1\} \; \forall Q \subset \mathcal{Q}$$

For example, $r_D(t, Q) = n$ means that target image t is at position n after ordering the dataset D with respect to collage Q. This value correlates with the time spent on the linear search of the provided ordered results. This gives us the motivation to evaluate our algorithm based on this rank across the whole set X.

In our approaches, we construct such ranking based on the appropriate dissimilarity score $\delta' : (\mathcal{U} \times \mathcal{P}(\mathcal{Q})) \to \mathbb{R}$. We discuss various choices of the dissimilarity score function further in the chapter. Then we may define the ranking as:

$$r_D(d, Q) = |\{d' \in D : \delta'(d', Q) < \delta'(d, Q)\}|$$

However, such construction may break the requirements on the ordering. Therefore, we need to "break ties", we do so in arbitrary fashion:

$$r_D(d, Q) = |\{d' \in D : \delta'(d', Q) < \delta'(d, Q) \vee (\delta'(d', Q) = \delta'(d, Q) \wedge d' <_a d)\}|$$

where $<_a$ is any arbitrary total ordering of the dataset D. For theoretical purposes, we could use the lexicographical ordering. In the implementation, we make use of a priori ordering of the dataset D as ordering $<_a$.

4.2 User-program Interaction

We provide the user with a canvas where they can place, move, and resize the collage images. They can add images by providing an URL of the image, or by pasting the image from a clipboard. One possible way to obtain images for the collage is to use an image search engine (e.g., Google Images). The images can be usually copied to the clipboard by selecting copy in the right-click menu. A quicker approach, we preferred, is using selective screenshots. With the selective screenshot, we can even crop the image, focusing on the part we like, and paste it onto the canvas.

Based on the provided collage, the program searches for similar images in the dataset and interactively presents them back to the user. The user can then alter the query for a new search, or investigate the displayed results. Figure 4.1b shows an example of the query – the collage of two images (window and cat).

4.3 Framework Overview

Our approach consists of several individual steps. Figure 4.2 shows a visual overview of the steps we describe here. Our first step is to compute the feature vector for each image, given the descriptor extraction function. We call this step feature extraction. This way, we obtain records, i.e., a pair of image from the dataset and its corresponding feature vector. We denote this feature extraction function as $f_d : \mathcal{U} \to \mathbb{R}^k$.

We often split feature vector into several parts (each describing only a part of the image). The appropriate part is then selected based on the position of the query image in the canvas and compared with the descriptor of the collage image. We denote selection function $z : (\mathbb{R}^k, [0,1]^4) \to \mathbb{R}^n$, where $[0,1]^4$ refers to the position of the image in the canvas.

In the collage processing path, we use feature extraction function $f_q : \mathcal{U} \to \mathbb{R}^n$ to obtain feature vectors of the images from the query collage (one vector for each image). This part is done online; therefore, we need to be able to obtain the descriptors on the fly to perform the search. Since we only extract descriptors for a couple of images, it is possible to use the predictions from the neural networks even without access to GPU.

As the next step, our goal is to obtain the distance from descriptor of each query image to descriptor of each image in the dataset, i.e., between the results of f_q and z. Finally, for each image in the dataset we obtain a multiset of distances. We aggregate these distances in order to compute the final dissimilarity score. Based on the dissimilarity scores, we compute the final ranking, as described in section 4.1.

To summarize, we compute the dissimilarity score $\delta'(d, Q)$ between $d \in D$ and collage $Q \subset \mathcal{Q}$, as follows:

$$\delta'(d, Q) = A(\{\delta(f_q(q_u), z(f_d(d), q_m)) \mid q \in Q\})$$

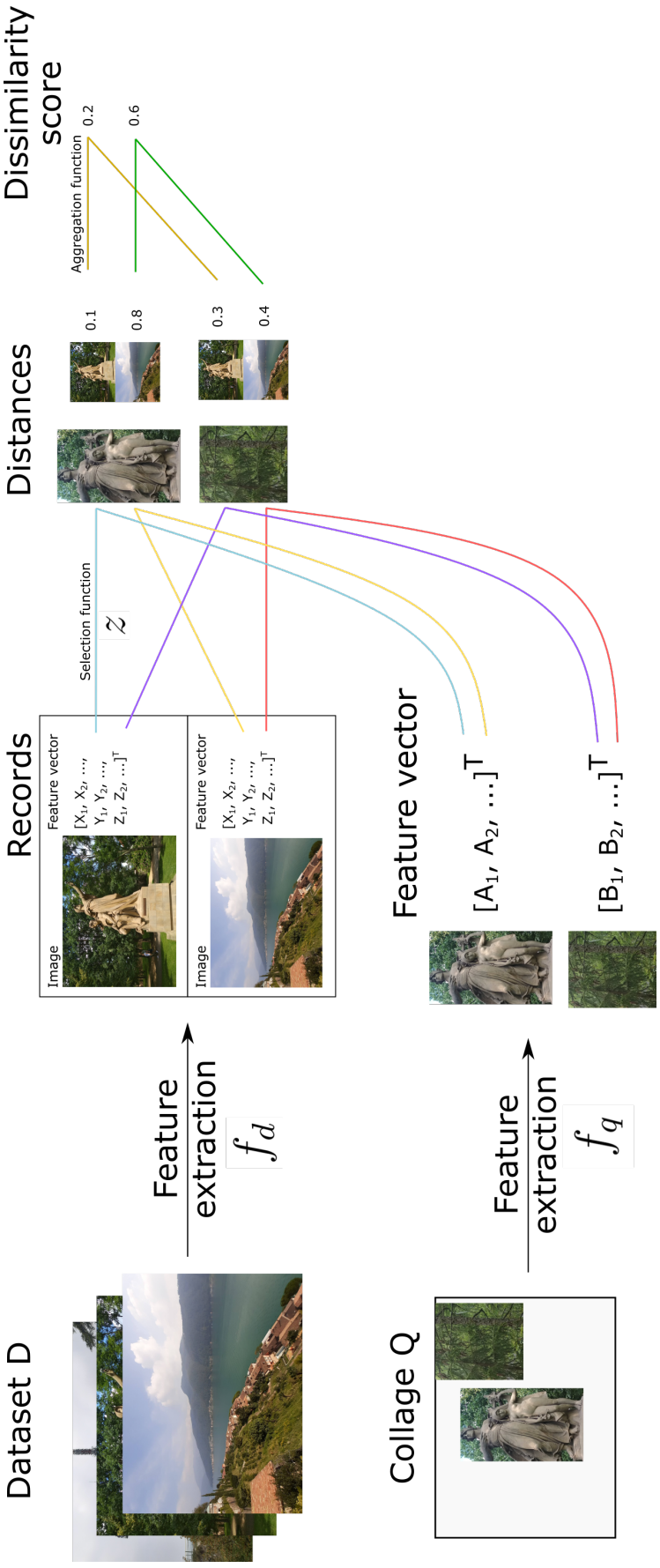


Figure 4.2: Overview of processing pipeline

```
Dataset:
    - image: 1 feature vector
Query:
    - query_image: 1 feature vector
    - compared to: each feature vector in the dataset
```

Figure 4.3: Overview of the Baseline – Image representation approach

where A denotes the aggregation function over a multiset. We use notation $q = (q_u, q_m)$, where q_u represents the collage image and q_m the metadata, i.e., position in the canvas. Also, in this particular equation, we use the $\{(\cdot)\}$ for a multiset.

While the described pipeline offers a reasonably straightforward approach to the task at hand, the individual stages' internal factors require a fair amount of tuning. We evaluate the hyperparameters with respect to the performance of the whole framework. Throughout the following sections, we progressively build the pipeline while describing the particular approaches we use. We start with the discussion on feature extraction and then progress towards the next stages of the system.

4.4 Features Extraction Strategies

In the following section, we present three feature extraction techniques. We begin this section with the description of the baseline model. The baseline model does not use spatial information about the images in the collage. This baseline approach is currently used, for example, by the VIRET tool [Kovalčík et al., 2020].

Following the baseline model, we present two approaches that utilize the spatial information from the collage. The first one splits the images in the dataset to fixed regions. The second one utilizes information from the neural network before the last pooling layer.

4.4.1 Baseline – Image Representation

The baseline ignores the spatial information of the images in the collage. We set this approach as our baseline since it is relatively simple and can solve our task. In this baseline approach, we compute the feature vector for an image in the dataset using a deep neural network as a descriptor extraction function and directly compare them with the descriptors of the collage's images.

In this approach we use a given CNN to obtain the descriptors of the collage's images and also dataset images. Therefore, $f_q = f_d$. Since, we directly compare the descriptors, the selection function $z(x, q_m) = x$.

Since the resolution of the images in the collage and images in the dataset may not match the input shape of the CNN, we rescale them to the required input size.

```
Database:
    - image:
        - multiple crops:
            - crop's position
            - feature vector
Query:
    - query_image: 1 feature vector
    - compared to: only to selected crops from each image
```

Figure 4.4: Overview of the regions' approach

4.4.2 Splitting the Image into Regions

In the next presented approach, we will focus on using the spatial information of the images in the collage. This position of the objects can serve as highly distinguishing criteria while looking for the target image. In the previous subsection, we described the image in the dataset by only one prediction of the neural network. We obtain higher granularity for the data by splitting the image into multiple regions and then computing the feature vector on each of the regions separately.

Let us assume that we split the image into m regions. For each image of the dataset, we now need to store m times more information. Resulting feature vector is of space $\mathbb{R}^{nm}$, where n is the size of the DNN output. To avoid increasing the time complexity by the multiplicative factor of m, we investigate a principle, how to compare the query image only with the one or few out of these m regions and not to all. That way, we preserve the same time complexity except for some additive factor, compared to the Baseline.

First, we describe the selection of the regions, and then we discuss the principle of region selection. The overview of the information preserved for this approach is in figure 4.4.

Cutting the Image

Each image in the dataset is divided into $N \times M$ regions. The descriptor is then computed and stored for each region separately. The defined cutting is identical for all images in the dataset.

First, we discuss the choice of the shape of the regions into which we want to split the images. Since we plan on feeding the region's image into a CNN, we set a limitation on using only square regions. This limitation comes from the fact that common CNN architecture expects a square image on the input. We could, in theory, rescale rectangular regions into squares, but this would induce a non-necessary distortion of the image. We avoid this distortion by posing this simple condition on the regions' shape.

We test several sizes of the input squares in the experiments. Although, we limit ourselves to the input shapes, for which a pre-trained CNNs are available. This for example for MobileNetV2 includes the following shapes: $96 \times 96, 128 \times 128, 160 \times 160, 192 \times 192$ and 224×224.

We impose a second limitation on the choice of regions. We require that their union fully covers the input image. In that way, information from each part of the

image is extracted. Also, we do not allow the regions to extend over the image.

Our limitations summarized are square regions, not extending over the image and full coverage of the input image. Except for a particular case, when the width and height would be divisible by the input shape dimension, the regions must overlap. The special case does not arise in our dataset, as we work with images 320×180.

We propose a cutting, which for the desired number of regions $N \times M$ with the desired width s, produces evenly distributed regions. In case the regions cover a greater area than the image itself (that is, $sN > h$, where h is the height of the image), the regions' overlaps are evenly distributed over the axis.

The side effect of overlaps also plays an important role in this technique. With the rigid frame without overlapping, we could face a situation where a single object would be split into two separate parts. Both parts separately could lack enough visual information about the object to provide consistent results. With the excess distributed to all regions, we share the information alongside the cut to both regions.

Our fixed parameters are regions' width s and the number of regions we want to use $N \times M$ and image size h, w. We solve the task of choosing regions splits for one axis; the second is done analogously. We know that the last region has to end with the edge of the image. Therefore, the starting coordinate of the last region is $h - s$. We then split the remaining "space" (space not covered by the last region) over $N - 1$ regions equally. We call this distance *step* since it says the distance between the starting points of the regions. The starting coordinate r_i of the i-th region in a given axis is:

$$step_h = (h - s)/(N - 1)$$
$$r_i = i\, step_h \text{ for } i \in \{0, 1, 2, \ldots, N - 1\}$$

With the condition on full coverage of the image (i.e., $sN \geq h$, and $sM \geq w$ respectively), we obtain full coverage of the image by the regions. Overlaps are evenly distributed over both axes. Note that, with bigger sized regions, three or more regions may overlap at the same position. For example, if we split an image with width 180 into three regions with a width of 96 pixels, some areas of the image will be included in all three regions. This happens when the $step < s/2$.

Selecting Regions

Previously, we defined cutting into the regions for each item in the dataset. One of the possible steps further in the pipeline could be to compare all regions' feature vector to the query image feature vector. Although this would be an entirely valid approach, it has two caveats: firstly, it does not take advantage of knowing the position of the query image in the collage, and secondly, it multiplicatively increases the time required, when comparing to $N \cdot M$ more feature vectors, than before.

We discuss the options of selecting only relevant regions to the query image based on the position of the query image. Given one query image with its position and shape, we propose the following methods for choosing the relevant regions:

- choosing only the one, which is covered the most by the query image,

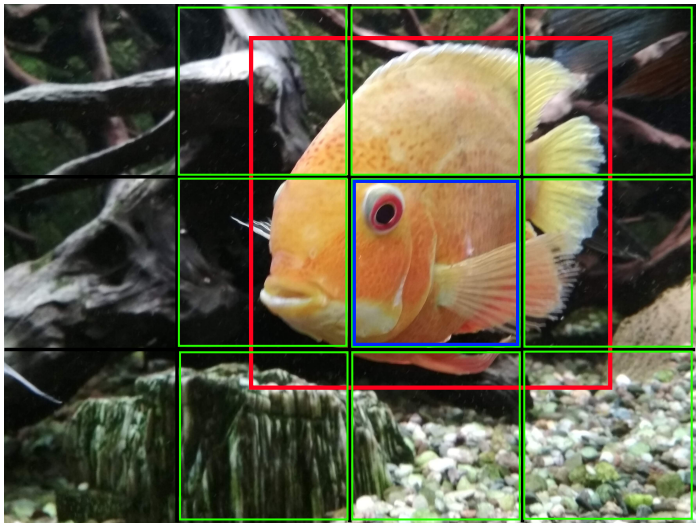


Figure 4.5: Example of choosing the corresponding regions. Red: query position; Green: all intercepted regions; Blue: region with highest IoU.

- choosing all regions, which have a non-empty intersection with the query
- or approaches in between, by setting a maximum to a number of relevant regions.

We visualize the problem in figure 4.5. The query would be an image of the fish placed as the red boundary shows. All regions with a non-empty intersection with the query image are highlighted in green. The region with the highest "coverage" is the blue one.

To make this idea over coverage measurable, we use the Jaccard index. We compute the Intersection over Union between the region and the crop, where the query image is located.

Intersection over Union (also known as Jaccard index[1]) measures similarity between two sets. We use it as a metric to express how much region and query image overlap.

The definition of the Intersection over Union is following:

$$J(A,B) = \begin{cases} 1, & \text{if A and B are empty} \\ \frac{|A \cap B|}{|A \cup B|}, & \text{otherwise} \end{cases}$$

In our case, the $|A \cap B|$ represents the area that belongs to both region and query image. The $|A \cup B|$ represents the area covered by the union of both. A visual representation of the formula is displayed in the figure 4.6. Using this index, we can order the regions based on their coverage with the query image. The more relevant regions are the ones with the higher Intersection over Union with the query image and vice versa. Please note, that in our particular case, measuring the IoU leads to the same results, as considering only the area of the overlap. However, this notation allows future work to use regions of various sizes.

From the regions ordered by their IoU to the query image, we select the first n. We leave n as one of our hyperparameters to evaluate in the experiments. Recall

[1] https://en.wikipedia.org/wiki/Jaccard_index

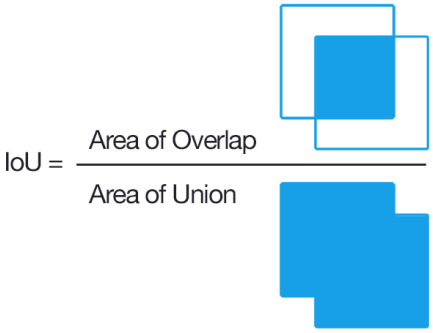


Figure 4.6: Intersection over Union between regions. Source: Wikipedia, CC BY-SA 4.0

that the required output of this stage is only one descriptor of dimension n, not multiple descriptors from multiple regions of total dimension k, where $k > n$. Therefore, when we compare the feature vectors with multiple regions, we select only a winner, with the lowest distance for the next phase.

4.4.3 Using the Representation Before Pooling

The disadvantage of the previously presented technique is fixed cutting. The cutting into regions does not adapt to the size of the collage image.

Let us retake a look at the fish in figure 4.5. We can see that fish covers approximately one-third of the image, but the individual regions cover only one-twelfth. In this case, only this one-twelfth of the image is compared to the query image. The method is missing any adaptation on the size of the query image.

After investigating the structure of the CNNs, we propose an approach based on the information obtained in the last convolution block. In standard architectures, after the last convolution block follows a pooling layer. So far, we only worked with the representation based on this pooling layer.

In this approach, we work with the results before applying pooling. As we have stripped two layers from the classification CNN — the fully connected layer used for classification, and the pooling layer, we also refer to this method as "antepenultimate" (meaning "last, but two").

Since the results we now work with are obtained before global pooling, i.e., from the last convolution block, they are from higher-dimensional space. The space of the feature vectors is now $\mathbb{R}^{k \times l \times c}$. Typically, this space is reduced to $\mathbb{R}^c$ by the global pooling layer.

Due to the way how CNNs work, we assume we could use the information about the query position, to work only with the part of the feature vector provided by the antepenultimate layer. The overview of the method is available in figure 4.7

Choosing a Region of Interest in the Layer

Layer before pooling on which we focus (antepenultimate) is the last convolution block. Therefore, it produces features from space $\mathbb{R}^{K \times L \times C}$, where K, L, C is the shape of the convolution block output. The first two dimensions represent the

```
Database:
    - image:
        - 1 feature vector (the result before pooling):
Query:
    - query_image: 1 feature vector after pooling
    - compared to: pooling over selected region
```

Figure 4.7: Overview of using the representation before pooling

spatial information, which is propagated from the previous layers. The third represents the channels (i.e., features).

To obtain only the part of this layer we are interested in (i.e., our query was placed in that specific region), we need to take only a subset over the first two dimensions. For a query $Q_i = (I, x_0, y_0, x_1, y_1)$ and a hidden layer L_i represented as a matrix $K \times L \times C$ we select a submatrix based on the query position.

Let the *rows* and *cols* be the indexes of the selected rows and columns, respectively.

$$rows = \begin{cases} \{\lfloor y_0 K \rceil, \lfloor y_0 K \rceil + 1, \ldots, \lfloor y_1 K \rceil - 1\} & \text{if } \lfloor y_0 K \rceil < \lfloor y_1 K \rceil \leq K \\ \{\lfloor y_0 K \rceil\} & \text{if } \lfloor y_0 K \rceil = \lfloor y_1 K \rceil < K \\ \{K - 1\} & \text{if } \lfloor y_0 K \rceil = \lfloor y_1 K \rceil = K \end{cases}$$

$$cols = \begin{cases} \{\lfloor x_0 L \rceil, \lfloor x_0 L \rceil + 1, \ldots, \lfloor x_1 L \rceil - 1\} & \text{if } \lfloor x_0 L \rceil < \lfloor x_1 L \rceil \leq L \\ \{\lfloor x_0 L \rceil\} & \text{if } \lfloor x_0 L \rceil = \lfloor x_1 L \rceil < L \\ \{L - 1\} & \text{if } \lfloor x_0 L \rceil = \lfloor x_1 L \rceil = L \end{cases}$$

where the $\lfloor (\cdot) \rceil$ represents rounding to the nearest integer. We then select the submatrix L'_i of the L_i as:

$$L'_i = L_i[rows; cols; \{0, 1, \ldots |C|\}]$$

Both MobileNetV2 and Resnet50V2 end with a convolution block with dimensions $(7, 7, C)$, where $C = 1280$ for MobileNetV2 and $C = 2048$ for Resnet50V2.

In this phase, we have selected a submatrix L'_i with respect to the position of the query image. We apply global average pooling, as was applied in the original network over the layer L'_i.

Performing the global average pooling over a given matrix A with the dimensions $K' \times L' \times C$ produces vector $(b_0, b_1, \ldots, b_{C-1}) \in \mathbb{R}^C$:

$$b_i = \frac{\sum_{k=0}^{K'-1} \sum_{l=0}^{L'-1} a_{k,l,i}}{K'L'} \ \forall i \in \{0, 1, \ldots C - 1\}$$

where $a_{k,l,i}$ represents an element of the A in k-th row, l-th column and i-th channel.

Finally, by performing the global average pooling over L'_i, we obtain our feature vector, which is later used.

Compared to the previous approaches, we aimed to avoid strict cutting and to provide more flexibility. The caveat of this approach is increased memory

consumption since we store 49 feature vectors per image in the dataset (7×7). It also requires to compute the average pool for each of the query images separately. Therefore, the time complexity is also increased. The factor is multiplicative, by the number of images in the collage and the size of the used layer.

4.5 Ranking

Let us remind an overview from figure 4.2. In the previous section, we have seen three approaches presented for feature extraction and the extraction of the relevant features. The first presented approach is the baseline; the second cuts the image into regions, and the third uses information from the antepenultimate layer of CNN. Now we shall focus on the remaining steps in the pipeline.

The first of these steps is computing distance between the descriptors of collage images and descriptors of dataset images. We evaluate all distance function mentioned in section 3.3.

Then we proceed to compute the dissimilarity score between the collage and images in the dataset. We do so by aggregating computed distances by the aggregation function A. Again, we test and evaluate various aggregation functions: minimum, maximum, and average.

As the final step, we compute the ranking, as defined in section 4.1 using computed dissimilarity scores.

4.6 Additional Parameters

In addition to different techniques of obtaining the feature vectors, we also inspect several other parameters of the system. These parameters include pre-processing of the collage images, post-processing of the features, and selecting the specific CNN. This section further discusses these parameters.

4.6.1 Padding

We talked about the importance of the square input in the regions section. Although, so far, we did not discuss how we handle the case when the user provides a non-square image to the collage.

The option to create non-square queries comes handy when the user wants to include, for example, a picture of a standing person, or a kayak. A question arises, what is the best way to edit the image before feeding it to CNN. We test three techniques: rescale, black padding, and white padding. An example of applying these techniques on an image is shown in Figure 4.8. We have previously discussed the caveats of the rescale — it distorts the images. If the person in the query image is in a tall and narrow box, the distortion will cause it to appear more full and shorter. The distortion becomes stronger with an increasing imbalance between the box dimensions. The alternative approaches, black and white padding, differ only in the filling color. The padding works by adding strips of the given color to make the originally shorter side the same length as the other. The stripes are of the same size, so the original image is centered along the axis that was shorter.

(a) Original image (b) Rescaled (c) Black padding

Figure 4.8: Overview of padding techniques

4.6.2 Dimensionality Reduction

In this section, we take a look at reducing the dimensionality of the extracted features. Dimensionality reduction could help us to scale our approaches to even bigger datasets and to decrease the query response time.

The extracted features from the neural networks are from high-dimensional space (for MobileNetV2, it is 1280 features, for Resnet50 it is 2048 features). Moreover, the dimensionality reduction can also have a positive effect on reducing noise if present in the feature vector. We aim to evaluate the system with a reduced number of features. For the dimensionality reduction, we use the Principal Component Analysis (described in section 2.2).

4.6.3 Neural Network Selection

Finally, we discussed all the hyperparameters we aim to evaluate. In the evaluation chapter, we run several experiments in order to find the best set of parameters. We use MobileNetV2 as our baseline feature extraction model due to its speed and small size.

After obtaining the best set of hyperparameters, we use the same set of hyperparameters for comparing the performance between different CNNs.

Aside from the MobileNetV2, we use two different instances of ResNet — ResNet50 and ResNet50V2. Both ResNet50V2 and MobileNetV2 were trained on the classification task with 1000 classes. We include the ResNet50 as model pre-trained on more than eleven thousand classes[2]. We aim to experimentally support the claim that the network trained on more classes can achieve a better performance level.

The downside of using ResNet50 trained on eleven thousands of classes compared to MobileNetV2 is slower evaluation and availability. Since the classification task on eleven thousand class is not one of the most common state-of-art challenges, the set of pre-trained neural network for this task is smaller. Also, used ResNet50 is only pre-trained with the input shape 224×224. We introduced upsaling of the input images in case of regions' approach to fit the input shape of the network.

[2]Source: `https://github.com/qubvel/classification_models`

5. Evaluations

In this chapter, we provide evaluations for each aforementioned parameter of the framework. The presented evaluations are performed on the collected collages.

5.1 Collected Collages

We manually created a set of queries to evaluate the proposed system's configurations. The Figure 4.1 shows one of such query — target image and collage. The annotated data consists of 102 collages. Each contains a visual description of a given target image. The average size of the images used in the collages covers 15% of the canvas. Five percent of the collected collages contain an image bigger than 80% of the canvas. We provide visualizations of the distribution of the annotated queries in Figure 5.1. We use a total of 199 images to create these 102 collages.

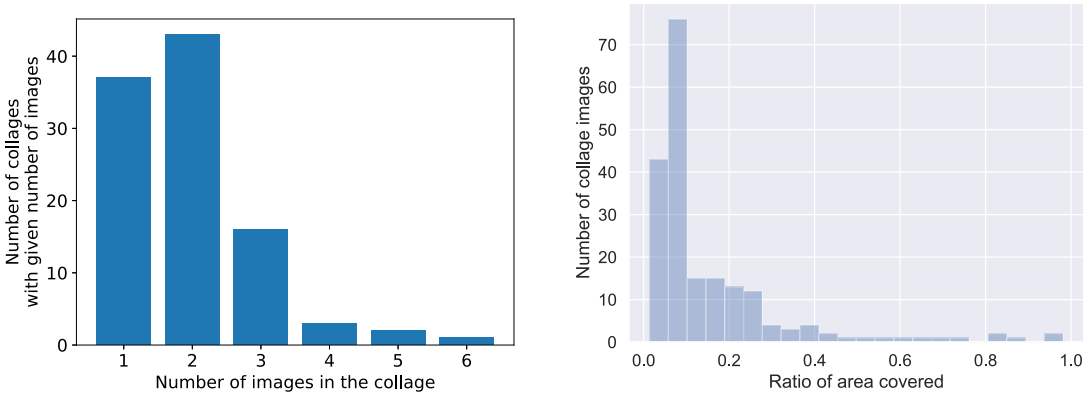
For the annotation, we used our application. It is possible to save the collage by clicking "Submit Collage." The average time spent on creating a single collage was 91 seconds. This time includes searching for images online, pasting them onto the canvas, and also waiting for the responses of the system.

We want to highlight that we created more than a hundred collages corresponding to the target images from the dataset. The annotations were done only by one person. We leave annotation from more people, to study the differences in behavior, for future work.

5.2 Plots Overview

Using the collected queries and based on the definition of the rank we can visualize the performance of the system.

For this purpose, we formulate a few additional terms. When measuring the performance of the system, we work on the set of collected queries X. For each



(a) Number of images per collage.

(b) Area covered by a collage image.

Figure 5.1: Collected collages properties

of the query collage Q_i, we say that the rank of the target image t_i is $r_D(t_i, Q_i)$ given the set of hyperparameters of the system θ.

We define a function $s_{D,\theta,X}$, which represents the performance of the system.

$$s_{D,\theta,X} : \{0, 1, \dots |D| - 1\} \to \{0, 1, \dots |X| - 1\}$$

$$s_{D,\theta,X}(h) = \sum_{(t_i,Q_i)\in X} [r_D(t_i, Q_i) \leq h]$$

In the plots we scale the both axis as the percentage of the dataset/collected queries. I.e., we display $r_D(t_i, Q_i)/|D|$ on x-axis and $s_{D,\theta,X}(h)/|X|$ on y-axis. For example, in Figure 5.2 we can see that about 80% (y-axis) of the target images were found within the first 20% (x-axis) of the dataset after sorting the dataset based on the corresponding collage. That way we obtain comparable results even in case of different datasets.

5.3 Summary

In this chapter, we focus on fine-tuning several hyperparameters of the task at hand. We achieved the best results using the approach of cutting the image into regions, with settings of 2×4 and 3×5 regions. Additionally, we showed that the best performing input shapes were 96×96 and 128×128. We did not found out that the number of selected crops would play a significant role in the performance of tested variants. For the distance function, we chose cosine distance since it performed significantly better than the other evaluated distance functions. To aggregate the distances, we selected the averaging function as it led to the best results. For the padding selection, it emerged from the experiment that use of rescale worked the best. From the tested networks, Resnet50 trained on eleven thousands of classes performed the best. Finally, we concluded that dimensionality reduction into 128 dimensions have a little, almost no negative effect on the system's performance. Therefore we recommend using it.

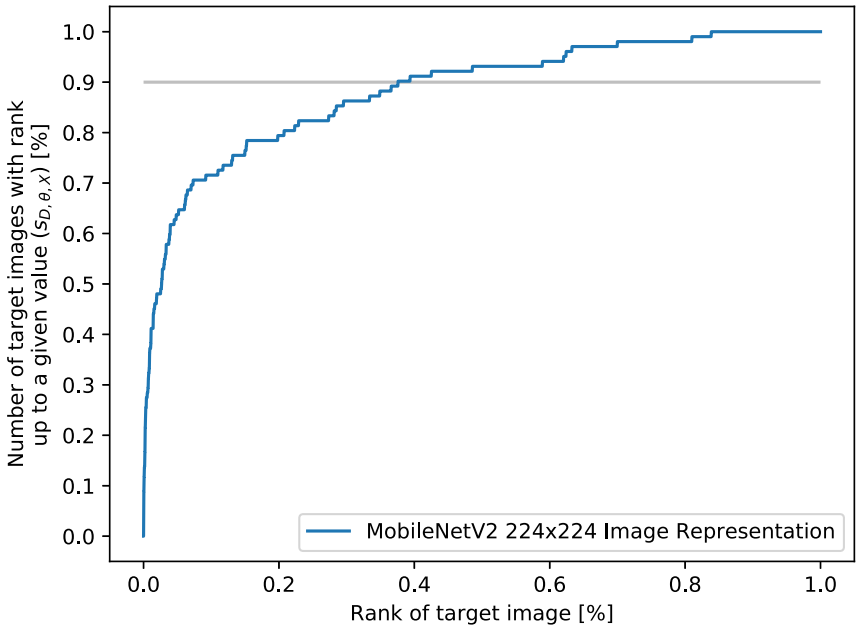


Figure 5.2: Performance of MobileNetV2 in the Baseline – Image representation setting

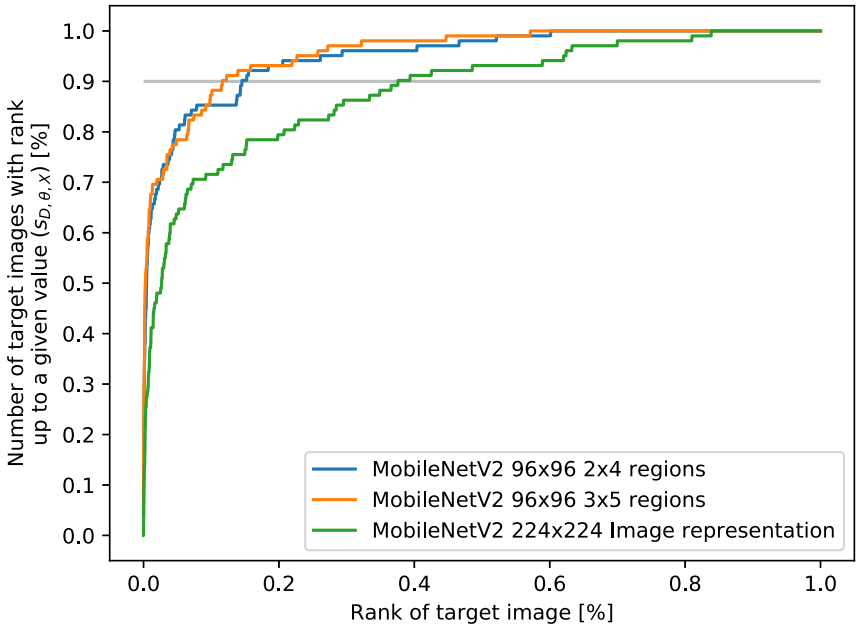


Figure 5.3: An experiment comparing the effect of the changing number of regions.

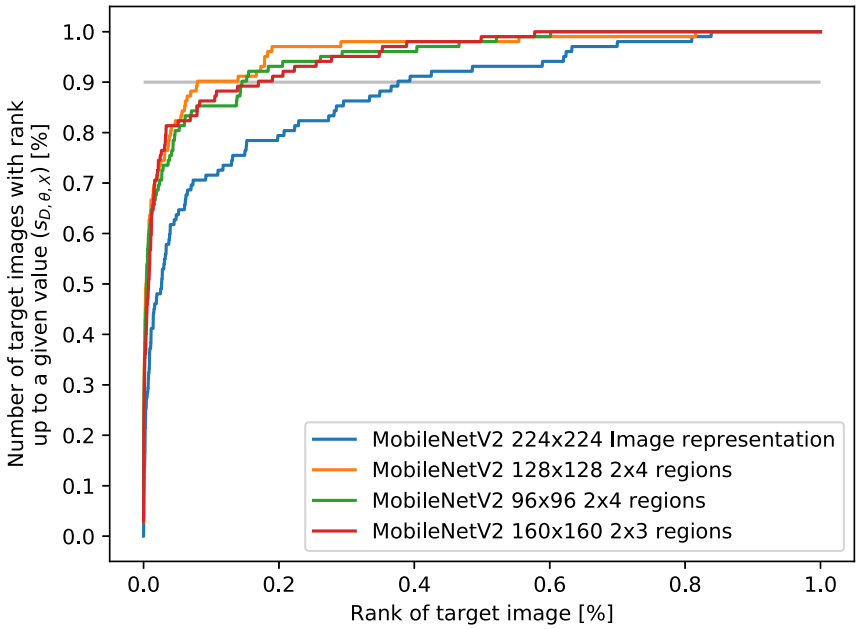


Figure 5.4: An experiment comparing the effect of the changing regions size.

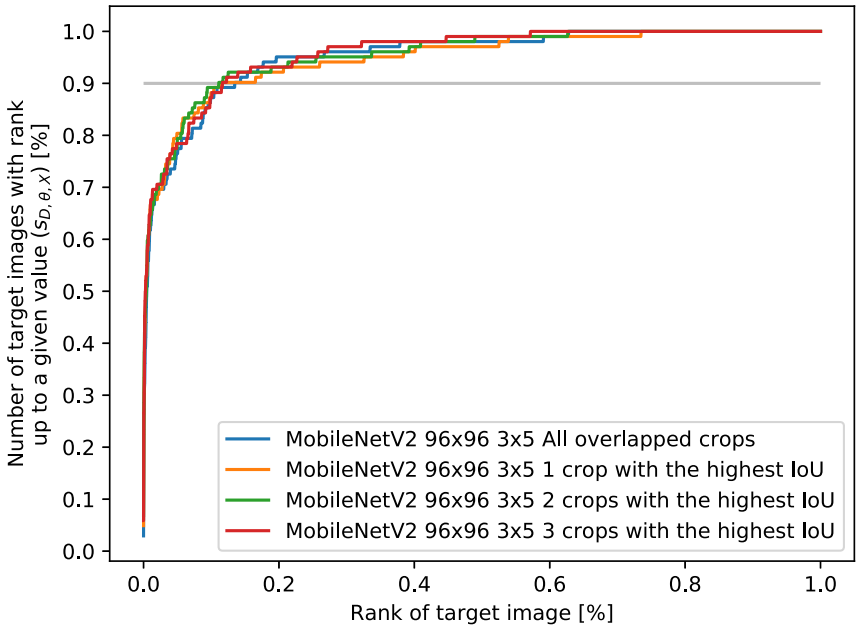


Figure 5.5: Performance of the system based on different number of chosen crops. This experiment shows only a minor change in performance with the increasing number of selected regions.

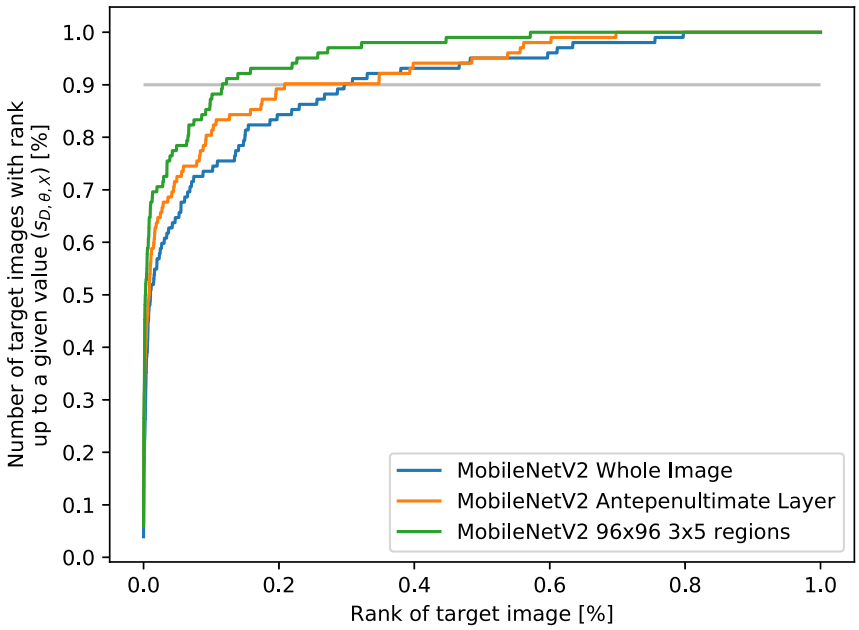


Figure 5.6: Comparison of baseline, regions, and approach using the output of the layer before the global pooling. Due to higher memory requirements, we had to sample the dataset. We randomly sampled one-tenth of the dataset. We also provide a baseline on this smaller dataset, the performance may slightly change, based on the selected samples. We can see that using the information from the layer before the global pooling layer improved the performance of the system, compared to the baseline. These are interesting results because the underlying network is for identical both approaches. This shows that we can extract the spatial information about the objects from the last convolution block.

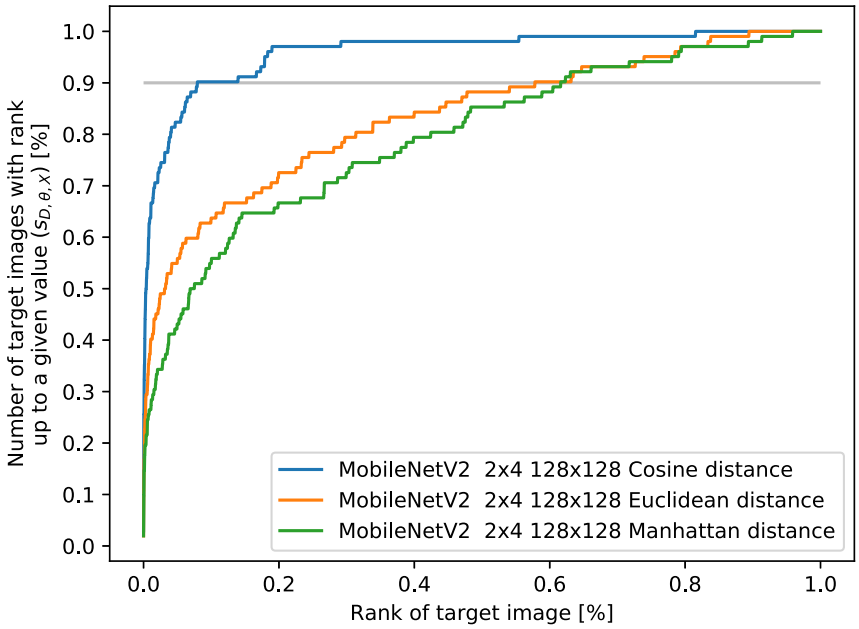


Figure 5.7: Comparison of the performance based on the chosen distance function. We see a superior performance of the cosine distance.

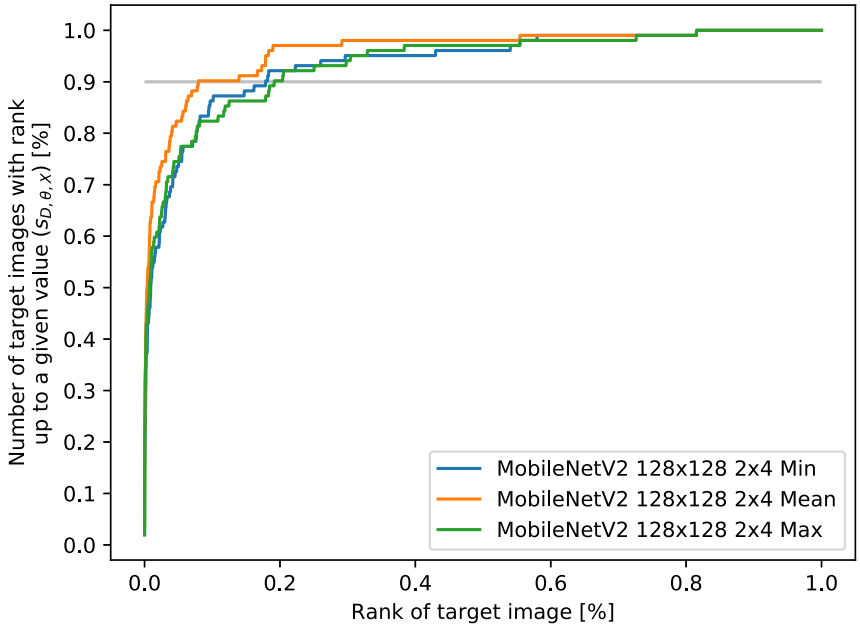


Figure 5.8: Performance of the system based on different aggregation function

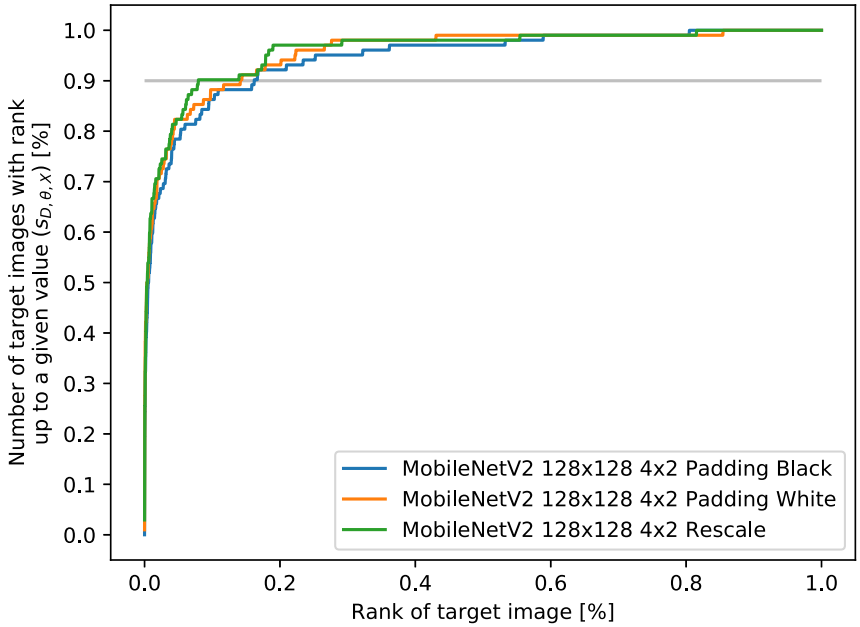


Figure 5.9: Comparison of different padding method for images in the query. Based on the experiment, rescaling is the best option.

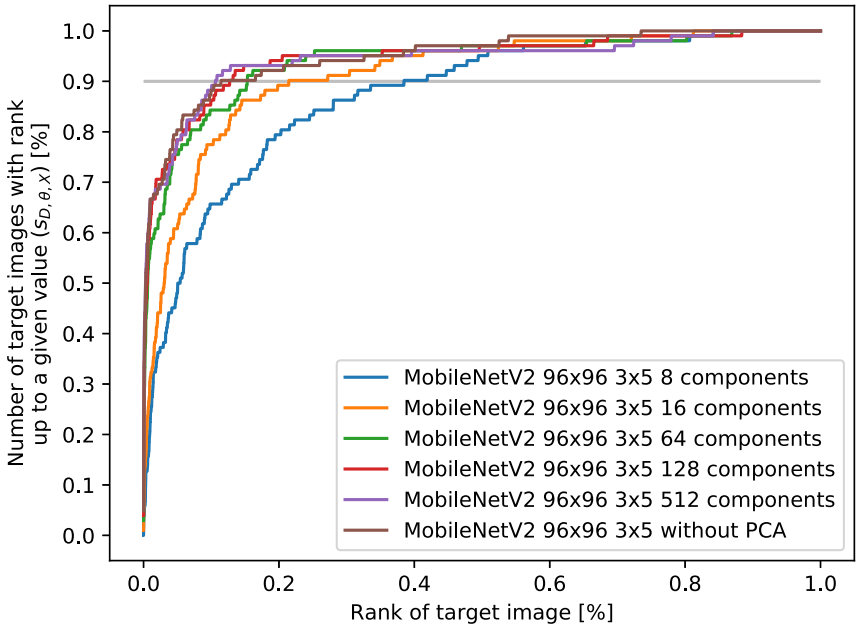


Figure 5.10: Effect of PCA on the performance of the system. The results offer several interesting observations. With the increasing number of components, the system performs better. With 512 components, it even performed slightly better than on the original data. For our purposes, we conclude that selecting 128 components offer the best performance–cost ratio. With the need for smaller feature vectors, we would select a minimum of 64 components.

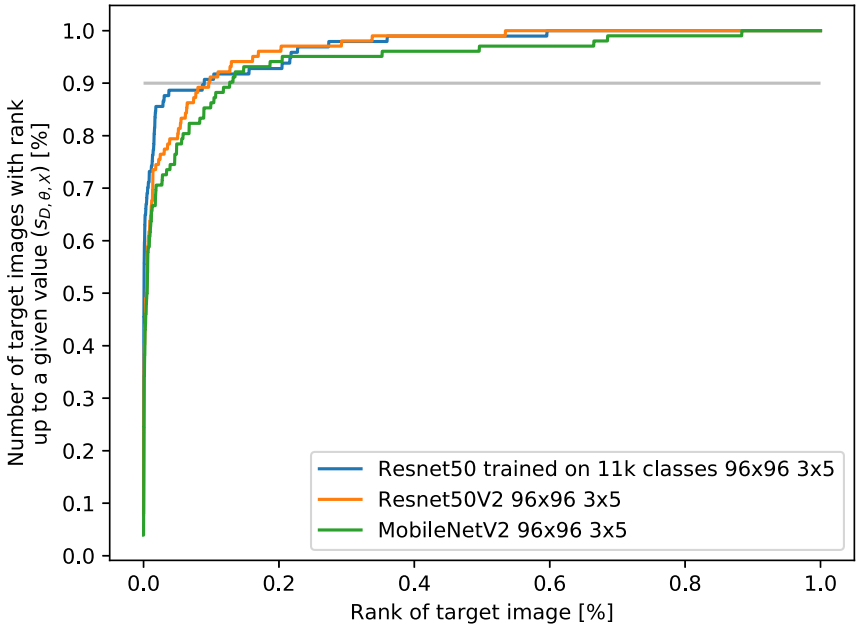


Figure 5.11: Comparison of the performance based on different feature extraction models. The performance of the Resnet50 trained on 11 thousands of classes is better compared to the other two. This experiment shows results after performing the PCA to 128 dimensions.

6. Search by Face Similarity

In this chapter, we propose another approach to the KIS task. This approach focuses only on the case, where the images include people. We investigate the option of finding the target image based on the person's face in the image and other faces in the dataset.

This approach arises from practical reasons. Once we investigated the V3C1 dataset, we realized that there are people in many of the images. We can use the previously investigated approaches based on the location and visual similarity to find the target image. However, with the increasing number of images showing people, it becomes difficult to retrieve the correct target image. Simply searching for a person in the top left corner would retrieve all images with a person in the said corner (not just the one we are looking for). Also, finding a good representative face with similar background becomes a challenging task.

In this chapter, we aim to provide a search mechanism based on the faces. We provide a simple search structure to investigate the faces in the dataset.

We first extract the faces from the dataset, and then we obtain a descriptor for each one of them. Based on these descriptors, we organize the faces into a traversal structure supporting navigational commands.

The task of comparing the faces of the people and saying which look more similar has its roots in the human perception of faces. Therefore, to evaluate the individual steps, we conduct experiments with human subjects.

Our experiments suggest that the feature space of the descriptors has a limited power to sort people based on the similarity in a way people do. Based on that, we implement the traversal structure, which can be used with the presented descriptors or any other descriptors of the faces that could be developed in the future.

6.1 Overview

Our goal is to propose a traversal structure based on the faces found in the dataset. Ideally, we would like to group similar faces so that users can quickly decide if the group of people corresponds to the target face they look for, or not.

The question we ask in the following sections is if the face descriptors can sort people based on the similarity, similarly to human perception. If yes, then we can argue that traversal structure based on these face descriptors may help. We leave experiments with additional descriptors for future work. In our case, we compare only one type of face descriptors to human perception, although, in future work, it may be experimented further with other descriptors.

6.2 Extraction of the Faces

If we look at the dataset, we notice that only a small portion of the people look directly into the camera. Since the images come from the videos, most of the people captured are doing some activity and not looking directly into the

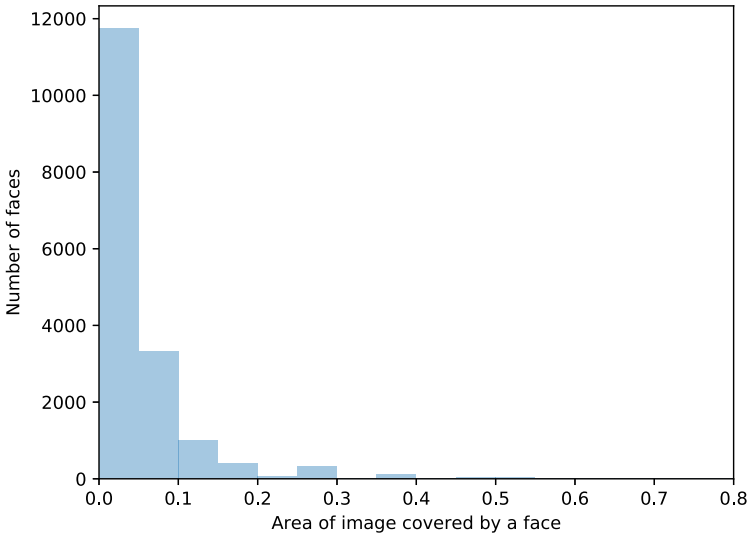


Figure 6.1: Area of image covered by a face

Figure 6.2: A random selection of faces extracted from the dataset.

camera. As discussed in subsection 2.1.5, we select the CNN based approach to extract the faces since it worked better with a wider variety of poses.

For our experiment, we use only a part of the original dataset. We extract faces only from the first 316 videos of the V3C1 dataset. Again, we firstly use image extraction from videos (refer to section 1.4), and only then we extract faces from images.

From these 316 videos, we were able to extract more than seventeen thousand faces. However, after investigation of the dataset, we found out that many of the extracted faces had a too low resolution. The majority of the extracted faces did not even cover 5% of the image. Therefore, we decided to filter out the dataset further and remove all the faces that did not cover at least ten percent of the image. The distribution of the area covered by a face is displayed in Figure 6.1. After the filtering, we obtained 2047 faces from the 316 videos. A random selection of faces is shown in Figure 6.2.

We then visually investigated the extracted faces. The dataset contains people of different ethnicity, age, and gender. The faces are displayed from a wide range of angles, including even full side views. Some of the people wear glasses, headphones or other accessories. The dataset also contains some pictures of children. The model also extracted a few drawn faces or faces of sculptures.

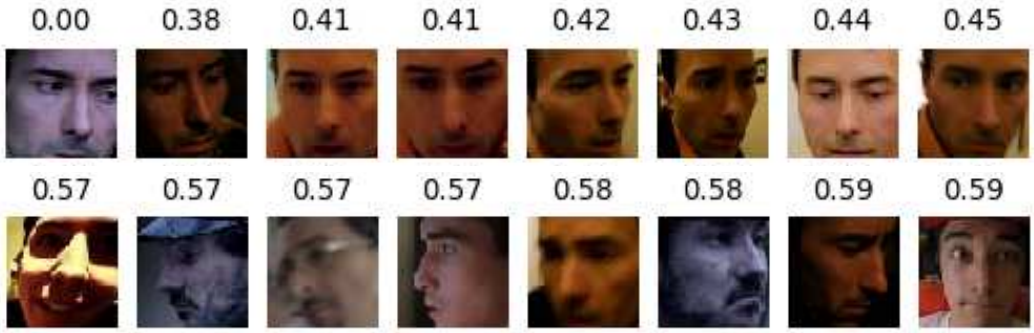


Figure 6.3: Examples of the retrieved closest faces to the target face (top left) based on euclidean distance. The number above each image represents the distance from the target.

6.3 Face Similarity Based on the Deep Features

For extracting descriptors (i.e., feature vectors), we use the dlib pre-trained model, as discussed in 2.1.5. The network outputs vectors from space $\mathbb{R}^{128}$. We use the output of the network directly as our feature vectors.

Note that often face features refer to the specifics of the face landmarks: the color of the eyes, size of the lips, etc. To avoid confusion, we use the term face encoding or descriptors for feature vectors obtained by the neural network. Such face encodings are usually from the space $\mathbb{R}^n$, in our case $n = 128$.

After obtaining the face encodings for our dataset of 2047 faces, we were interested in whether this feature space, created by the CNN is able to sort the faces based on the similarity in a way people do.

The library, from which we use the model, refers to a threshold 0.6 in the euclidean distance as the threshold with the highest accuracy on the verification task. The example of the closest faces based on this threshold is displayed in Figure 6.3. The top left image belongs to the target face. In the provided example, we can see that the face encodings close to our target face truly belong to the same person. Unfortunately, we can see that with increasing distance, other people appear even below threshold 0.6.

6.4 Case Study

In this section, we investigate a possible correlation between the given faces as perceived by humans and the distances between their encodings.

We conducted a study with 25 participants. We presented to them a 10×10 grid of randomly selected faces from the dataset. Then we showed them ten randomly selected target faces of different people (same set of faces to each participant). We asked the participant to select for each target face exactly three faces from the grid that looked the most similar.

Out of 10 target faces, three were also represented in the grid. Two of the faces were extracted from the same image, i.e., the same angle of the face. The third face, which was also present in the grid, had two representants in the grid. First identical to the target face and the second with only a minor change of the angle.

Firstly, we investigate how likely it is that the human subject notices that the face they are looking for is also present in the grid. In the first case, twenty-four out of 25 respondents selected the face corresponding to the same target person. In the second case, only 20 respondents found the face in the grid. We assume that the first case was "easier" because the face comes from black and white image, therefore, it was easier to notice. In the third case, two face views of the target person were available in the grid. Nine respondents selected both correctly, and eleven respondents selected only one of them.

As our next step, we empirically evaluate, if the model used for obtaining the face encodings, can sort the faces based on the distances similarly as human subjects do.

We further investigate the seven target faces, which were not present in the grid. Excluding the three cases where the target face is in the grid, we avoid overestimating the quality of our model since we know that the model is performing well on the verification task. For every face F from the grid G, we compute the euclidean distance from the target face $T \in D$, where D denotes our dataset of faces (distance between their encodings). We then reorder the faces from the grid based on this distance (similar to the ranking from the previous chapters).

Formally we define distance γ between faces $F, F' \in D$ based on the descriptor extraction function f and euclidean distance d_e:

$$\gamma(F, F') = d_e(f(F), f(F'))$$

This definition allows us to rank faces in the grid G based on the target face T:

$$r_T(F) = |\{F' \in G \mid \gamma(F', T) < \gamma(F, T)\}|$$

Again, to "break ties", we use any arbitrary total ordering $<_F$ over faces, as in section 4.1.

Note that this ranking function is a bijection between the faces in the grid and a set $\{0, 1, \dots |G| - 1\}$, given the target image. Therefore, it is reversible, i.e., we can obtain the face from the given rank. Based on this ranking, and the data obtained from the respondents, we aim to verify the correspondence of the ranking based on distances of the encodings and the most similar faces as identified by the respondents.

Based on the results of the survey, we can estimate the probability p that for a given target face T user selects a face with rank R. We also denote the number of respondents that selected face F for target face T as $N_T(F)$. The resulting probability then is:

$$p_T(R) = \frac{N_T(r_T^{-1}(R))}{25}$$

Finally, to provide comprehensive overview, we compute combined probability of selecting face with rank R by combining the results for all target faces $\mathcal{T}$:

$$p(R) = \frac{1}{|\mathcal{T}|} \sum_{T \in \mathcal{T}} p_T(R)$$

Note that as each user selects three faces, it holds: $\sum_{R \in \{0,1,\dots 99\}} p(R) = 3$.

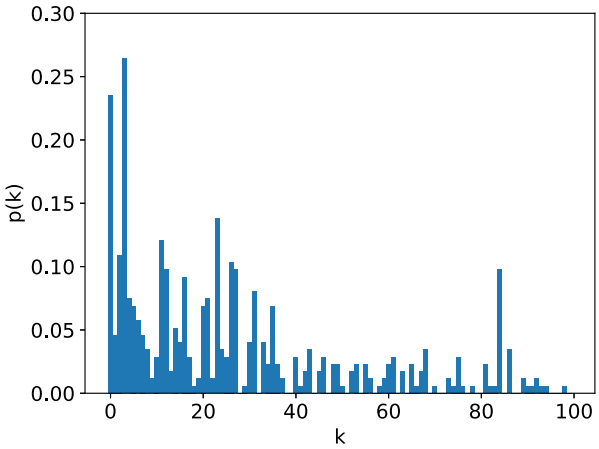


Figure 6.4: Collected statistic on how likely user selects k-th closest face to the target in the face grid

We plot the empirical probability p, of a user selecting the face on the k-th rank, given the target face. If the user is fully coherent with the model, they would choose the closest three faces based on the euclidean distance, and for the $R \in \{0, 1, 2\}$, we would see a probability $p(R) = 1$ and zero otherwise.

This plot of the probabilities is displayed in Figure 6.4. We do not normalize this plot so that we can talk about the probability, given our experimental settings, i.e., selecting exactly three faces.

We are interested in the distribution of ranks given the faces selected by the users. We want to know if there is any correlation between similarity in the feature space and the similarity perceived by the human respondents.

As the last investigation from the case study we provide a graph (Figure 6.5) displaying the expected value of the number of faces X_R selected up to rank R, which can be expressed in terms of p as $\mathbb{E}[X_R] = \sum_{R' \leq R} p(R')$. On average, one of the three selected faces by the user has a rank of less than 12. In the plot, we also present a "random selection". This would be the case of no underlying information on the face similarity from the model. Therefore, users would be equally likely to choose face at any given rank.

We leave for future work, a more comprehensive study with more target faces providing more reliable estimates. Based on the experimental results of this case study, it seems to indicate that the distance over encodings correlates with the similarity of the original faces as perceived by the human subjects.

6.5 Building a Traversal Scheme

In the previous sections, we extracted and encoded faces. In the case study, we investigated the feature space for the correspondence to the human annotations. Here we propose a solution of organizing faces into a multilevel view.

As discussed in chapter 1, a common solution for the traversal system is a 2D grid. In the 2D grid, a user can navigate using the following commands: move left, right, up, and down.

Most of the time, we want to browse more than a hundred images, it becomes

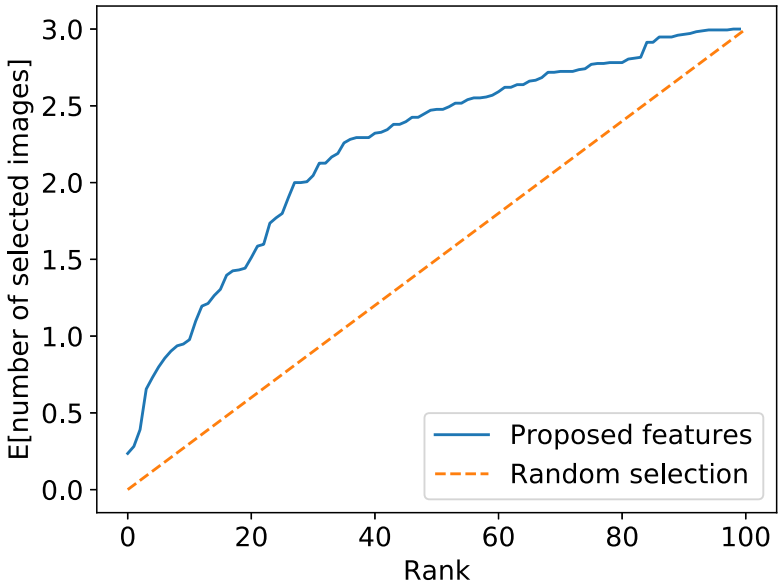


Figure 6.5: Expected value of the number of images selected up to a given rank.

inconvenient to preview dataset only in one layer. Therefore, we build a simple multilevel structure to ease the navigation.

6.5.1 Tree-based Structure

Our goal is to organize a dataset of images D to allow easy navigation. Let us assume that the dataset can be organized into a grid of size $N \times M$. We leave the choice of specific dimensions to the user. We prefer a square setting, although it is not required. In case that the size of the dataset $|D|$ is not square of any number, we recommend adding placeholder images.

We organize (so far in a non-specific way) the items from the dataset into a chosen 2D grid. This is the base layer for the tree structure, we denote this grid as L_0. The layer has dimensions $L_{0,h} \times L_{0,w}$. Based on this layer, we create the next layer as only a subset of this layer. Firstly, we select k, which represents the sampling size factor. It means that every k^2-th image is selected as a representant to the next layer. The overview of the structure is shown in Figure 6.6.

The layer L_{i+1} has $1/k$ of the size in both axis of the L_i, more specifically, if the L_i is of the dimensions $L_{i,h} \times L_{i,w}$, the layer L_{i+1} has following dimensions:

$$L_{i+1,h} = \lceil L_{i,h}/k \rceil$$

$$L_{i+1,w} = \lceil L_{i,w}/k \rceil$$

The item on the position m, n in the L_{i+1} is replicated from the layer L_i as follows:

$$L_{i+1,m,n} = L_{i,mk,nk}$$

We continue reducing the size of the layers until the layer will fit into a single display.

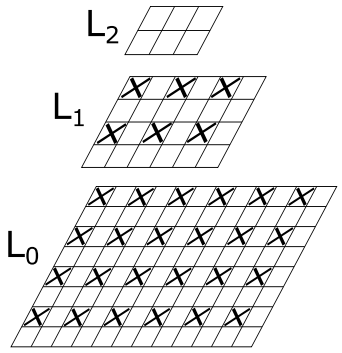


Figure 6.6: A preview of the tree-based traversal structure with $k = 2$. The marked items are selected for the layer above as representans.

In this structure, we support six types of navigation: left, right, down, up, in, and out. In and out represent operations of traversing between the layers, and other directional commands allow navigation within the layer.

6.5.2 Bottom Layer – Self-organizing Map

In the bottom layer, we would like to make use of the face encodings. As we reviewed in chapter 1, Self-organizing Maps showed a promising way to project high-dimensional data onto the 2D grid. We train a Self-organizing Map on our dataset of face encodings. For our particular dataset, we train a network of size 50×50. This offers us 2500 neurons, having more neurons than there are faces in the dataset.

To train a SOM, we use a MiniSom framework [Vettigli, 2013]. We use the default setting for the SOM, a gaussian neighborhood function with an asymptotic decay for the parameters. This setting showed the best results after visual inspection. We train the SOM for 200 000 iterations, i.e., each example was seen on average 100 times during the training. Since the training of the SOM highly depends on the random initialization, we first run the same settings for ten different seeds for 30 000 iterations. Based on the results, we selected a seed with the lowest quantization error (refer to section 2.3). We reran the training with the best seed and trained for 200 000 iterations.

The training errors are shown in Figure 6.7. We can observe that topographic error is at this stage stable. Even though the quantization error has not converged yet, at this stage, we did not achieve any improvements by our visual inspection.

Then we built the bottom layer for our traversal structure based on the trained SOM. This SOM represents our grid. For each neuron in the SOM, we assign a face whose encoding is closest to the given neuron's weight vector. We refer to this face as a representative of the given neuron. Note that this way of selecting representatives does not avoid duplicates (i.e., assigning the same face to multiple neurons).

We explored the assignments, and these are the observations we made: 345 faces out of original 2047 are not assigned to any of the neurons. However, only 22 out of these 345 faces have the distance to the closest face, which is assigned to a neuron, higher than 0.45. In the original study, the authors of the model suggest considering encodings closer than 0.6 to be of the same person.

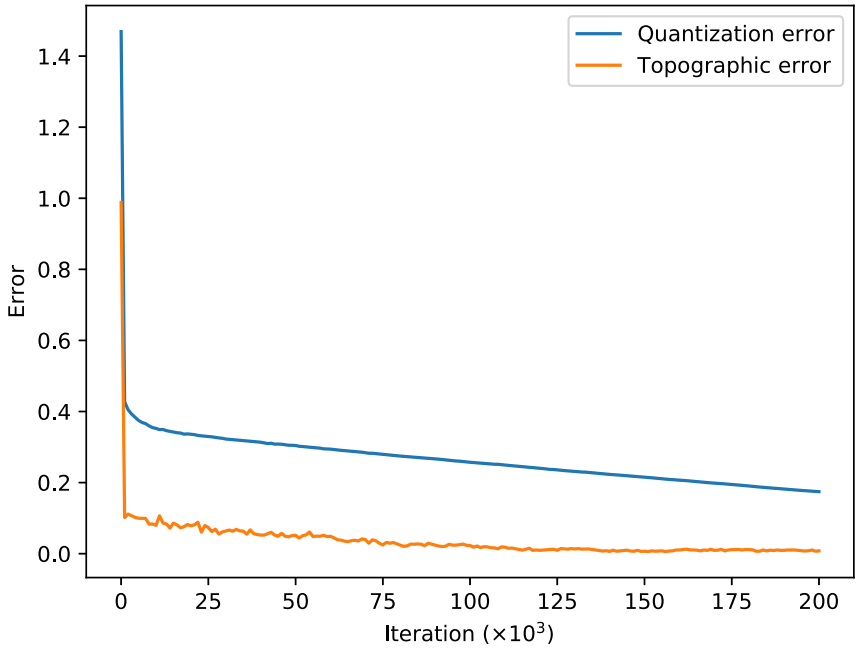


Figure 6.7: Errors during the training of 50 × 50 SOM, 200 000 iterations.

As our investigation of the dataset in Figure 6.3 shows, threshold 0.45 is strict for assigning the same person. Therefore, we expect that those 323 missing faces belong a person, which is already among the representants. The maximum distance for the excluded 22 faces is 0.56, which is still below the initially proposed threshold for assigning the same person. Nevertheless, we verified that all of these faces, which are not selected as representants to any of the neurons of the SOM, are still accessible from the structure by an additional utility that shows the most similar faces to a given face. We discuss this utility in the next section.

6.5.3 Accessing the Most Similar Faces

We enhance the tree-structure to support, displaying the most similar faces to a given face. A user can click on any of the displayed faces. Images, which contain a face considering to be the same person (threshold 0.6), are then displayed. This is helpful in finding the exact target image when the user finds the same person on the map. This also provides a way to learn for the user. The user can display close faces, and investigate the characteristics by which faces are grouped by. Based on this, the user can utilize information about the structure to perform the face search faster in the next searches.

6.6 Summary

Our experiments suggest that the space of encodings contains some information on the face similarity. We propose a multilevel structure to explore a dataset of faces. In this structure, we support navigational commands that allow users to navigate within a layer (move left, right, up, and down) as well as traverse between the layers. Lower layers contain more faces, while upper layers offer a

greater variety of the displayed faces.

We created the bottom layer with the help of a Self-organizing Map. We assigned to each neuron the closest face encoding (and corresponding face) from the dataset. To provide users with an additional insight into the data, we also support a search for closest faces to any given face in the traversal structure.

In the last section, we present a preliminary evaluation. More thorough evaluations, with more participants, would be needed to limit the effect of user's memory and other factors. This would require different participants searching the datasets using different approaches and evaluating the time needed to find a face. Although we do not provide such a complex study, we include the results of two respondents searching for ten faces using two different approaches.

6.7 Preliminary Evaluation

In this section, we present a preliminary evaluation of the proposed navigational system. The goal of this chapter was to propose an exploration method over the dataset of faces. Therefore, we evaluate it by conducting an experiment with users.

The task for the user is to find a target image in the dataset. As our baseline, we construct a grid of the faces sorted by their original videos. Therefore, multiple images of one person are grouped since they come from the same video. For searching in this grid, a user can only scroll up and down. We then test this approach against our traversal structure. The hypothesis we aim to prove is that our solution decreases the average time needed to find the target face compared to the search in a linear search.

For our experiment, we use ten randomly selected target images, which include faces. For each of them, we let the user search for the face in the dataset. Respondent A firstly searches for the faces in the baseline approach, then using the traversal structure. Respondent B searches in the opposite order, firstly they work with the traversal structure, and then with the baseline.

We measured the time required to successfully find the target image. We set the limit for this task to ten minutes. We obtained 20 measurements per approach. Respondent A was unable to retrieve a "face 4" within the time limit using either of the search approaches. These attempts were considered as 10 minutes of searching for the purposes of the evaluation. Respondent B successfully retrieved target images in all tasks.

Based on the experiment, we present results in Figure 6.8. In the case of both respondents, the median of the time required to solve the task was lower for the traversal search compared to the linear search. We also note that in terms of the average, we do not see any improvement.

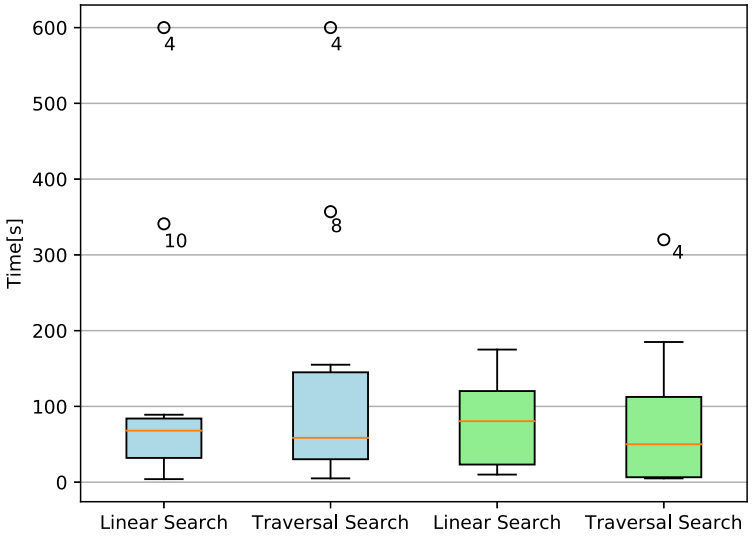


Figure 6.8: Comparison of the time required to find a target image. Light blue belongs to the respondent A, light green to respondent B. The outliers are identified by the index of the task.

7. User Guide

In the previous chapters, we developed and evaluated several methods for a Known-item search task. In this chapter, we provide a user guide for our implementation of the aforementioned techniques. This chapter only covers user interaction and does not cover experimenting with new datasets nor the overview of the implementation.

The user can access the application via a web-based interface. Once the webpage is opened, we can see by default a module for search based on the collage (refer to chapter 4). The second module includes face similarity search (chapter 6). We proceed to describe both modules.

7.1 Running the Application

The easiest way to run the application is to install Docker[1]. Once the Docker is installed, it is enough to run following commands from the top-level directory, i.e., where `README.md` is located.

```
# ./thesis-grizzly/
$ docker build . -t app
$ docker run -p 8000:8000  \
  -v $PWD/image_representations:\
      /diplomova_praca/static/image_representations \
  -t app
```

Once the application is running, we can access it via browser at following web address: `http://127.0.0.1:8000/`. The first initialization takes longer, the website will load itself once the initialization completes.

Included Dataset

The provided demo includes a small dataset of images to search on. These images come from Open Image Dataset[2]. The dataset contains 20 035 images, for which we computed the region splitting for 2x4 regions. For an approach using the antepenultimate layer, we only provide extracted features for half of the dataset, due to the size limit of the attachments. Finally, we extracted 528 faces from the dataset for the search by face similarities (with the same settings as described in chapter 6).

The data, which the model uses is in the directory `image_representation`. In the next chapter, we describe how to work with custom data. It is possible to either replace the existing ones or to create a new directory and update the path in the provided command.

[1] `https://docs.docker.com/desktop/`

[2] `https://opensource.google/projects/open-images-dataset`, CC-by 4.0

7.2 Modules

Our application is separated into two modules: spatial search and face search. Users can switch between the modules in the top right corner. Now we shortly describe a typical interaction of the user with the system for each module.

7.2.1 Search by Collage

The default module, which opens, is search by collage. On the screen, we can see a canvas for creating the queries and some control elements. On the first load, a target image (the searched scene) is displayed for several seconds over the canvas. During the creation of the collage, we can always access the target image by clicking on the image button ("Display hint"), see Figure 7.1.

Creating a Collage

We can create custom collages on the canvas. To add an image to the canvas, we can either paste it, or add it via the URL of the image. To paste an image, it needs to be available in the clipboard. The easiest way to do that for most images is to right-click on the image and select "Copy image". We can also recommend a selective screenshot feature, which can speed up the copying of the images and add the possibility of selecting only a part of the image. For Windows 10, it is possible via key combination Shift + Win Key + S. Linux users with KDE may use Spectacle (usually invoked via Print Screen key) or a similar utility.

There is no limitation on the number of images in the collage, although the increased number of images may reduce the performance (as they may give misleading hints), and also it prolongs the computation time needed for the query to be processed.

Once the image is placed in the canvas, we can resize it by grabbing the bottom right corner or move it by dragging. To remove the image from the canvas, click on the `x` button in the top right corner. To submit the query, click the "Query" button. By default, automatic querying is turned on, i.e., after each movement or resizing, the system automatically queries the dataset. This can be turned off, which comes handy if the model with longer inference time is used.

It is possible to switch between the approaches used for feature extraction. The list of available approaches can be accessed by clicking on the three dots, next to the query button.

Displayed Results

By default, the application displays only the best 100 matched results. The images are reloaded after each successful query. The top of the Results section also displays the rank of the searched item. This helps the user to learn how to use the tool more effectively. The winning regions are highlighted as well for the regions' search by an orange rectangle.

Results are presented based on the ranking, as we described in section 4.1. More similar results are displayed first. If the image is not present in the loaded dataset, the rank will not be displayed. This may be a case if the features for a given approach are not available for all images in the dataset.

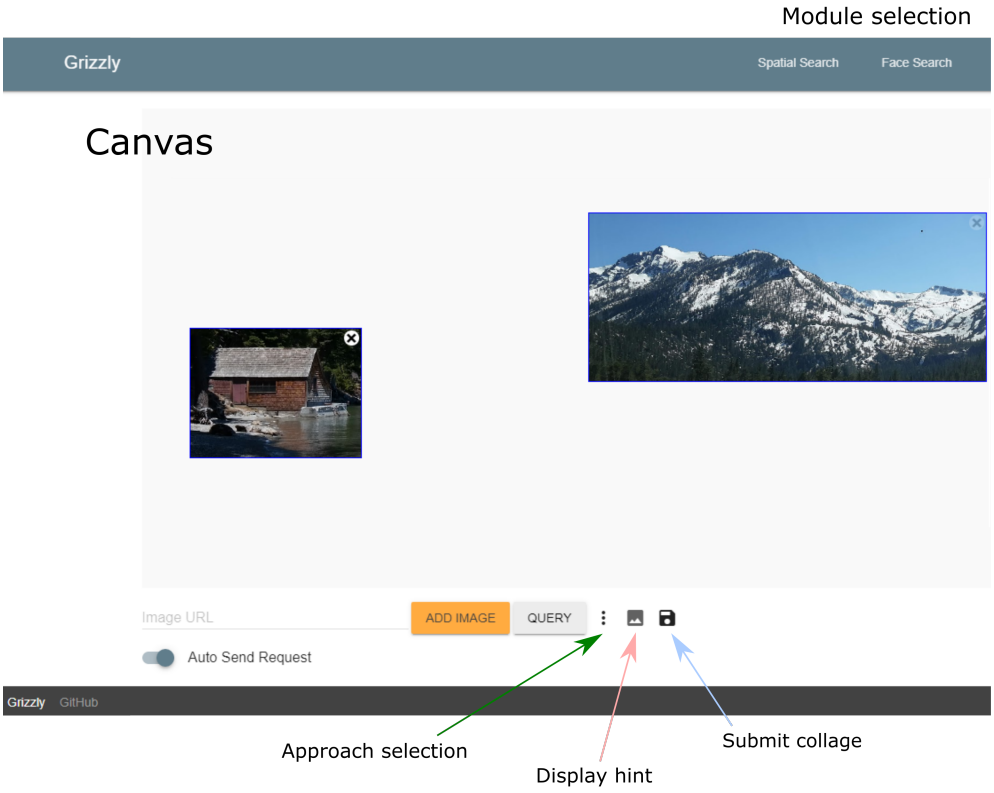


Figure 7.1: User interface for creating collages

7.2.2 Search by Face Similarity

Upon loading the face module, there is a grid with faces, and also a target image is displayed. Like the Search by collage module, the target image can be accessed at any moment by clicking on `Show the image`.

The grid, which is displayed, represents the top layer of the multilevel structure. This layer fully fits into a display; therefore, move commands do not have an effect. We can step up from any (except top layer, where it does not have any effect) layer by clicking on the `Zoom Out`.

By clicking at any face, we step down one layer. The image which we clicked is in the center of the next layer, if possible. This is not possible for images near the corners and edges. In such case, the grid is displayed in a way that full display is used, and the clicked image is as close to the center as possible.

Each image in the traversal structure also supports two additional operations: show images from the video and show the closest faces. In the case of images, which did not come from the videos (the case of the demo dataset), the images displayed by the first option, are arbitrary. The second option displays all images, which contain a face similar to the queried one. Both these options are available by corresponding icons in the top right corner of the face, available at any layer. A top layer for our demo dataset is displayed in the Figure 7.2

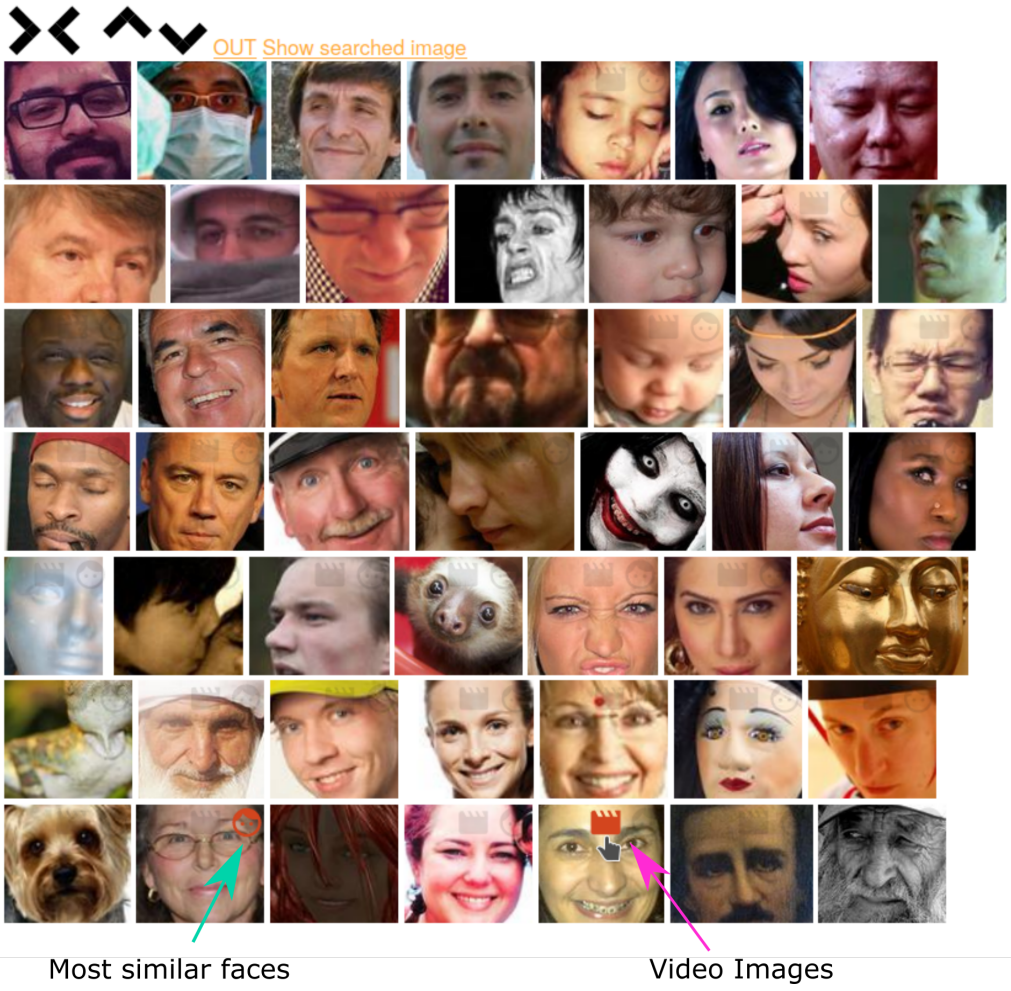


Figure 7.2: Top layer of our traversal structure.

8. Extracting Features for a Custom Dataset

In the previous chapter, we described how to run a demo with a preprocessed dataset of images. In this chapter, we describe the steps required to run the application for a new set of images. If we do not want to change the dataset but only evaluate different approaches (i.e., changing the underlying neural network), it is enough to do only the corresponding step.

Our feature extraction is a two-step process. We again use Docker, to avoid dependency problems. The extraction process belongs to separate module – `diplomova_praca_lib`. Therefore, we firstly change the working directory, and then we build a single docker container, which we use for all extractions.

```
$ cd diplomova_praca_lib
$ docker build . -t lib
```

The user needs to prepare a directory with images to be processed. If the images come from videos, this directory should be further divided into subdirectories. Each subdirectory should then contain images from one video.

8.1 Feature Extraction for Collage Approaches

In this section, we provide an example of extracting features for our system for both collage approaches: regions and the approach using the last convolution block. In this example, we extract features by `MobileNetV2`, using regions `2x4` with the `input_size=96`. The system expects equally sized images.

```
$ images="/path/to/images/"
$ intermediate_output="/path/to/intermediate_output/"
$ features="/path/to/features/"

$ docker run \
  -v $images:/images \
  -v $intermediate_output:/feature_records \
  lib \
   python diplomova_praca_lib/annotate_images.py \
    --images_dir=/images --save_location=/feature_records \
    --feature_model=mobilenetv2 --num_regions=2,4 \
    --input_size=96

$ docker run \
  -v $intermediate_output:/feature_records \
  -v $features:/features \
  lib \
    python diplomova_praca_lib/preprocess_data.py \
      --input=/feature_records --output=/features \
      --transform --regions --explained_ratio=128
```

Parameter options for feature extraction are:

- `--feature_model`: `resnet50v2antepenultimate`, `resnet50v2`, `mobilenetv2antepenultimate`, `mobilenetv2`,
- `--input_size`: depends on the model used, check with Keras Applications
- `--num_regions`: other values evaluated in this thesis were 2x3, 3x5. Other viable options are allowed.

The second step includes preprocessing with optional PCA dimension reduction. The parameters are:

- `--empty_pipeline`: no PCA is applied, data are only preprocessed forming structure for the application
- `--explained_ratio`: PCA into given number of components is applied
- `--regions`: to specify that the data reflect the regions' structure

The content of the `$features` directory can be then copied to the corresponding directory (see section 8.3) and immediately used, by running the app (see chapter 7).

8.2 Feature Extraction for Face Similarity

For the extraction of faces and their encodings, we use similar steps as in the previous section:

```
$ images="/path/to/images/"
$ intermediate_output="/path/to/intermediate_output/"
$ face_features="/path/to/face_features/"

$ docker run \
  -v $images:/images \
  -v $intermediate_output:/feature_records \
  lib \
   python diplomova_praca_lib/annotate_images.py \
    --images_dir=/images --save_location=/feature_records \
    --feature_model=faces

$ docker run \
  -v $intermediate_output:/feature_records \
  -v $face_features:/features \
  lib \
    python diplomova_praca_lib/preprocess_face_data.py \
      --input=/feature_records --output=/features \
      --crop_size=0.08
```

where the `crop_size` specifies filtering criteria on the minimum area covered by the face in the image.

8.2.1 Training Self-organizing Map

After obtaining the face encodings, we can train the underlying Self-organizing Map, which we use to obtain a grid of faces for the bottom layer.

```
$ face_features="/path/to/face_features/"
$ trained_som="/path/to/trained_som/"

$ docker run \
  -v $face_features:/features \
  -v $trained_som:/output \
  lib \
   python diplomova_praca_lib/train_som.py \
    --input=/features \
    --iterations=200000
```

For `train_som.py` additional arguments as `learning_rate`, `sigma`, `distance` may be explored. The scripts saves all intermediate Self-organizing Maps and also plots the errors during the training. It is possible to choose any of the stored SOMs for the application and observe the differences between them.

8.3 Running the Server with Custom Data

We can replace any of the existing data by the new ones, either in their original location `image_representations`, or we can create a new location for the data

with following structure:

```
image_representations/
    faces/
        features/
            faces.npz
        som/
            som.pickle
    images/
        000/
            image1.jpg
        001/
            image2.jpg
    regions/
        model-mobilenetv2,dir=000.npz
        model-mobilenetv2,dir=001.npz
    spatial/
        model-mobilenetv2antepenultimate,dir=000.npz
        model-mobilenetv2antepenultimate,dir=001.npz
```

It is not necessary to provide all the data for all the approaches. Only the modules with correctly provided data will work as expected.

Conclusion

In this thesis, we explored techniques to tackle the Known-item search task on a set of images. We started by reviewing the existing approaches presented at Video Browser Showdown. Based on the limits of existing systems, we proposed two approaches: search by collage and search by face similarity. We built both approaches utilizing state-of-art deep learning models.

Search by Collage

In the search by collage, we let the user create a collage of images. Based on this collage, we provide the most relevant (i.e., most similar) results. We proposed a multistage framework design, which allowed us to experiment with different settings for the individual stages.

We explored the possibilities to obtain the descriptors while utilizing spatial information about the collage images. We proposed two approaches. The region's approach cuts the images into a fixed set of regions. Based on the position of the image in the collage, we then select only relevant regions for further processing. Our experiments showed that this approach performed the best.

The second approach took advantage of the last convolution block of the CNN. We selected the interesting subset of the convolution block corresponding to the image's position in the collage. We showed, that computing the descriptors using only a part of this block, improved the performance.

As additional parameters, we evaluated three strategies for preprocessing the images before feeding them into a convolutional neural network. We either rescaled the image, added black, or white strips. As our experiments showed, rescaling the images achieved the best results.

Secondly, we investigated the effect of the dimensionality reduction of the features on the model. We were able to reduce the number of features to as low as 128 without losing too much performance of the whole system. Based on this, we propose a further use of our system in the competitions such as VBS, where the dataset is ten times bigger than the dataset we worked with.

Finally, we tested three different networks. MobileNetV2, as it is a fast and small network, ResNet50 and Resnet50V2. For the ResNet50, we used a pre-trained model on eleven thousand of classes compared to one thousand classes in the case of MobileNetV2 and Resnet50V2. The experiments showed that the ResNet50 on eleven thousand classes performed the best.

For the evaluations, we handcrafted a set of 102 collages. All experiments presented were tested on this set of queries.

Search by Face Similarity

In the second part of the thesis, we discussed a possibility to search through the dataset using only faces. Firstly, we extracted faces from the dataset. We investigated and explored obtained faces. We used a neural network trained to identify people to extract the descriptors. We conducted a case study, where we aimed to verify if the space of the face descriptors has an ability to sort people based on the similarity as people do.

We conducted a study, asking 25 respondents to find three most similar faces in the grid of 100 faces to a given face. Each user completed the task for ten different target faces. The experiments indicate a correlation between the people's answers, and the ordering based on distances in the high-dimensional features.

Based on these findings, we have organized the faces from the dataset into a 2D grid with the help of the Self-organizing Map. We further presented a traversal structure to allow quick searching within the grid. We also provided an option to the user to display the most similar faces to a given face.

Overall, we presented two distinct techniques for tackling the Known-item search task. It is important to note that this task is an open problem, and even though we proposed and tested promising solutions, this problem is not yet solved.

Future Work

Although our experiments were successful in providing new approaches to the Known-item search task, it left much room for further improvements. We provide a couple of suggestions for future work:

- Experiments with regions of mixed sizes in the Regions' based approach
- Exploration of other network architectures for the feature extraction step
- Implementing proposed approaches into existing tools, to support multi-modal queries
- Explore the effect of descriptors and extraction methods on the search by faces
- More thorough studies on the performance of our traversal system

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2020.

George Awad, Asad Butt, Keith Curtis, Yooyoung Lee, Jonathan Fiscus, Afzal Godil, Andrew Delgado, Jesse Zhang, Eliot Godard, Lukas Diduch, Alan F. Smeaton, Yvette Graham, Wessel Kraaij, and Georges Quénot. Trecvid 2019: An evaluation campaign to benchmark video activity detection, video captioning and matching, and video search & retrieval. In *Proceedings of TRECVID 2019*. NIST, USA, 2019.

Gregory T Breard. Evaluating self-organizing map quality measures as convergence criteria. 2017.

Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016.

François Chollet et al. Keras. `https://keras.io`, 2015.

Ritendra Datta, Dhiraj Joshi, Jia Li, and James Z Wang. Image retrieval: Ideas, influences, and trends of the new age. *ACM Computing Surveys (Csur)*, 40(2): 1–60, 2008.

Michel Marie Deza and Elena Deza. Encyclopedia of distances. In *Encyclopedia of distances*, pages 1–583. Springer, 2009.

Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. A deep convolutional activation feature for generic visual recognition. *UC Berkeley & ICSI, Berkeley, CA, USA*.

Adam Geitgey. Machine learning is fun! part 4: modern face recognition with deep learning. *Medium. Medium Corporation*, 24, 2016.

Andreas Girgensohn, John Adcock, and Lynn Wilcox. Leveraging face recognition technology to find and organize photos. In *Proceedings of the 6th ACM SIGMM international workshop on Multimedia information retrieval*, pages 99–106, 2004.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Cathal Gurrin, Tu-Khiem Le, Van-Tu Ninh, Duc-Tien Dang-Nguyen, Björn Þór Jónsson, Jakub Lokoč, Wolfgang Hurst, Minh-Triet Tran, and Klaus Schoeffmann. An Introduction to the Third Annual Lifelog Search Challenge, LSC'20. In *ICMR '20, The 2020 International Conference on Multimedia Retrieval*, Dublin, Ireland, 2020. ACM.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016b.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Gary B Huang, Marwan Mattar, Tamara Berg, and Eric Learned-Miller. Labeled faces in the wild: A database forstudying face recognition in unconstrained environments. 2008.

Peter Kalocsai, Wenyi Zhao, and Egor Elagin. Face similarity space as perceived by humans and artificial systems. In *Proceedings Third IEEE International Conference on Automatic Face and Gesture Recognition*, pages 177–180. IEEE, 1998.

D King. Dlib 18.6 released: Make your own object detector, 2017a.

Davis King. High quality face recognition with deep metric learning. `http://blog.dlib.net/2017/02/high-quality-face-recognition-with-deep.html`, 2017b.

Davis E King. Dlib-ml: A machine learning toolkit. *The Journal of Machine Learning Research*, 10:1755–1758, 2009.

Kimmo Kiviluoto. Topology preservation in self-organizing maps. In *Proceedings of International Conference on Neural Networks (ICNN'96)*, volume 1, pages 294–299. IEEE, 1996.

Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43(1):59–69, 1982.

Teuvo Kohonen and Timo Honkela. Kohonen network. *Scholarpedia*, 2(1):1568, 2007.

Markus Koskela et al. *Interactive image retrieval using self-organizing maps*. Helsinki University of Technology, 2003.

Gregor Kovalčík, Vít Škrhak, Tomáš Souček, and Jakub Lokoč. Viret tool with advanced visual browsing and feedback. In *Proceedings of the Third Annual Workshop on Lifelog Search Challenge*, pages 63–66, 2020.

Miroslav Kratochvíl, Patrik Veselỳ, František Mejzlík, and Jakub Lokoč. Somhunter: Video browsing with relevance-to-som feedback loop. In *International Conference on Multimedia Modeling*, pages 790–795. Springer, 2020.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Jakub Lokoč, Werner Bailer, Klaus Schoeffmann, Bernd Münzer, and George Awad. On influential trends in interactive video retrieval: video browser showdown 2015–2017. *IEEE Transactions on Multimedia*, 20(12):3361–3376, 2018.

Jakub Lokoč, Gregor Kovalčik, Tomáš Souček, Jaroslav Moravec, and Přemysl Čech. A framework for effective known-item search in video. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 1777–1785, 2019a.

Jakub Lokoč, Gregor Kovalčík, Tomáš Souček, Jaroslav Moravec, and Přemysl Čech. Viret: A video retrieval tool for interactive known-item search. In *Proceedings of the 2019 on International Conference on Multimedia Retrieval*, pages 177–181, 2019b.

J. Lokoč, W. Bailer, K. Schoeffmann, B. Muenzer, and G. Awad. On influential trends in interactive video retrieval: Video browser showdown 2015–2017. *IEEE Transactions on Multimedia*, 20(12):3361–3376, 2018.

Iacopo Masi, Yue Wu, Tal Hassner, and Prem Natarajan. Deep face recognition: A survey. In *2018 31st SIBGRAPI conference on graphics, patterns and images (SIBGRAPI)*, pages 471–478. IEEE, 2018.

Michael A Nielsen. *Neural networks and deep learning*, volume 2018. Determination press San Francisco, CA, 2015.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

Arun Ponnusamy. Cnn based face detector from dlib. `https://towardsdatascience.com/cnn-based-face-detector-from-dlib-c3696195e01c`, 2018.

Rajeev Ranjan, Swami Sankaranarayanan, Ankan Bansal, Navaneeth Bodla, Jun-Cheng Chen, Vishal M Patel, Carlos D Castillo, and Rama Chellappa. Deep learning for understanding faces: Machines may be just as good, or better, than humans. *IEEE Signal Processing Magazine*, 35(1):66–83, 2018.

L. Rossetto, R. Gasser, J. Lokoc, W. Bailer, K. Schoeffmann, B. Muenzer, T. Soucek, P. A. Nguyen, P. Bolettieri, A. Leibetseder, and S. Vrochidis. Interactive video retrieval in the age of deep learning - detailed evaluation of vbs 2019. *IEEE Transactions on Multimedia*, pages 1–1, 2020.

Luca Rossetto, Ivan Giangreco, Silvan Heller, Claudiu Tănase, and Heiko Schuldt. Searching in video collections using sketches and sample images–the cineast system. In *International Conference on Multimedia Modeling*, pages 336–341. Springer, 2016a.

Luca Rossetto, Ivan Giangreco, Claudiu Tanase, and Heiko Schuldt. vitrivr: A flexible retrieval stack supporting multiple query modes for searching in multimedia collections. In *Proceedings of the 24th ACM international conference on Multimedia*, pages 1183–1186, 2016b.

Luca Rossetto, Ralph Gasser, Jakub Lokoc, Werner Bailer, Klaus Schoeffmann, Bernd Muenzer, Tomas Soucek, Phuong Anh Nguyen, Paolo Bolettieri, Andreas Leibetseder, et al. Interactive video retrieval in the age of deep learning-detailed evaluation of vbs 2019. *IEEE Transactions on Multimedia*, 2020.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

Hideyuki Tamura, Shunji Mori, and Takashi Yamawaki. Textural features corresponding to visual perception. *IEEE Transactions on Systems, man, and cybernetics*, 8(6):460–473, 1978.

Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22, 2011.

Giuseppe Vettigli. Minisom: minimalistic and numpy-based implementation of the self organizing map. 2013.

John M Wandeto and Birgitta Dresp-Langley. The quantization error in a self-organizing map as a contrast and colour specific indicator of single-pixel change in large random patterns. *Neural Networks*, 119:273–285, 2019.

List of Figures

Acronyms

API Application programming interface. 13, 15, 65

BMU Best Matching Unit. 16, 65

CBIR Content-based image retrieval. 18, 65

CNN Convolutional Neural Network. 12, 25, 29, 31, 41, 42, 58, 65

DNN Deep Neural Network. 11, 26, 65

IoU Intersection over Union. 28, 65

KIS Known-item search. 3, 5, 6, 18, 21, 40, 65

PCA Principal Component Analysis. 15, 32, 39, 55, 64, 65

RGB Red Green Blue color model. 65

SOM Self-organizing map. 7, 16, 46, 47, 56, 65

VBS Video Browser Showdown. 3, 5–7, 9, 65

A. Implementation Overview

When we were creating our application, we kept in mind to make it as easy as possible for users to use. Therefore, we opt for a web-based interface with which users come in contact daily. From the user point of view, it does not require the installation of any additional application, and it can be used across many operating systems.

Creating a simple, shareable environment for users allows us to collect more annotated data. Compared to the classical applications, which need to be installed on each user's computer, a web-based application needs to be set up just on the server by the researcher and then users can simply access it by the web address. This way, it may be easier to reach a bigger audience.

For our back-end we decided to use Python, because it has a big machine learning community with ever-growing possibilities of open-source projects. As a result of this, we decided to use Django — easily scalable Python web-framework.

Since our implementation covers a wide range of tasks — from a web-based front-end to a library able to work with deep learning models, we do not include an exhaustive enumeration of the specific classes and functions in this chapter. Instead, we provide an overview of the individual parts of the application. We start with the separation of the application into three layers, as displayed in Figure A.1.

This figure represents the individual layers of our framework and provides a summary of the technologies used. Our application consists of two main Python modules: `diplomova_praca` and `diplomova_praca_lib`. In the figure, the front-end and back-end belong to the first one, and the library represents module `diplomova_praca_lib`.

We shortly describe each of the layer, as displayed in the figure.

A.1 Front-end

Our front-end consists of several Django templates (a way to create HTML files dynamically), with the corresponding styling. We then use jQuery to provide a user with an interactive environment in the application. The part of displaying the menu and the results are shared between both modules — face and spatial. The interactive content — canvas for the spatial and the grid for faces — is dynamically added based on the currently selected module.

Design

For a unified look, we follow the Material Design guidelines. We use some of the components from Material Design Lite. To complement the overall design, we also use Material Design Icons.

We style our own components with using the Cascading Style Sheets. We use Syntactically Awesome Style Sheets (commonly known as SCSS) to reduce the amount of code. One of the advantages of SCSS compared to CSS is the support of variables and nesting. We use this, for example, when setting the colors for the

whole application. This way, we can provide users with a unified color scheme, while the colors can be easily adjusted for a different color scheme, if needed.

Interaction

To provide a smooth user experience, without any unnecessary loading during the search, we use JavaScript. For some actions, like dragging, resizing we utilize functionalities from jQuery[1].

To avoid reloadings, we update the content using JavaScript and we obtain the data via asynchronous communication with our Django back-end. One of the advantages of this approach is that the user can continue experimenting with the collage while the system is retrieving the results.

We do the communication with the Django as asynchronous HTTP POST requests. We encode the data as JSON, or directly as a string, if possible. The data are of different types; for example, in the spatial module, we need to send all images with their relative position. In the case of the face module, we include the Tree View, i.e., the current level and position in the layer.

A.2 Back-end

The goal of our back-end is to process the HTTP requests from front-end and transform them into Python objects. This way, we can directly query our library by calling the functions and passing the Python objects. The back-end also initializes the datasets in the library. Additional functionality of the back-end is the choice of the target image out of available images. Secondary functionalities include logging submitted collages and additional data into the database.

A.3 Library

In the library, we implement all approaches mentioned in this thesis. The library includes all the feature extraction methods, image handling, training, plots generation, and more.

Since the content of the library is broad, we separate it further into four main blocks:

- Annotation and Preprocessing — scripts used for extraction of the features from a custom dataset
- `face_features/` – face module, which is responsible for creating the multilevel structure (`TreeView`). It also contains models for the extraction of faces and the face encodings.
- `position_similarity/` – spatial module, which includes used models and the feature extraction techniques (as regions or antepenultime approach). The entry point for any request is function `positional_request()` in the file `position_similarity_request.py`. We use this entry point also for our experiments.

[1] `https://jquery.com/`

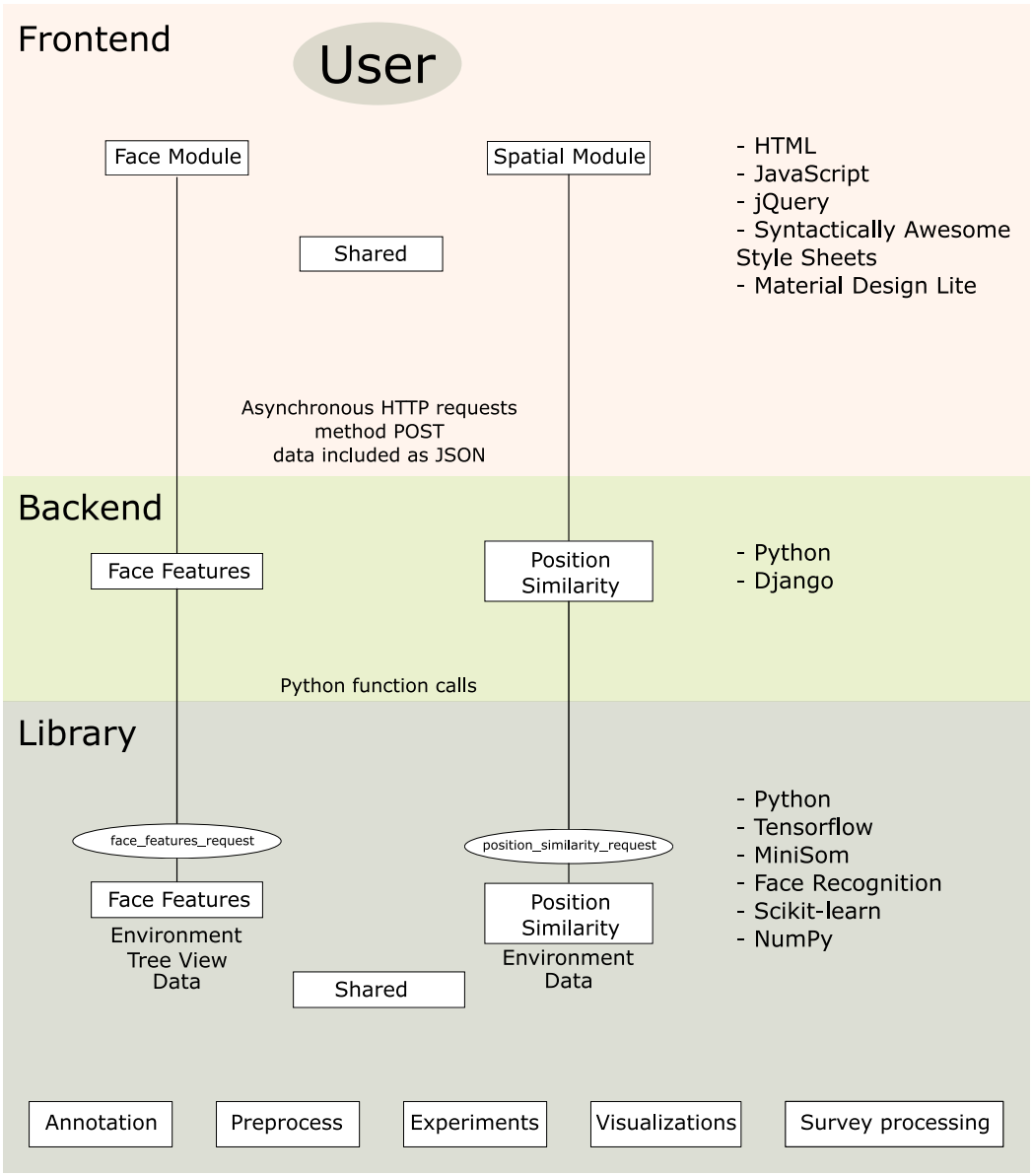


Figure A.1: Overview of the implementation

- Additional scripts (`helpers/`) – this includes all the scripts to run and evaluate the experiments. These are not necessary for the application itself, but useful while investigating the approaches or the data.

Attribution

At last, we would like to express our gratitude to the authors of all the used technologies provided by an open-source community. Additionally, to the projects mentioned in the text, we would like to thank the authors of crucial libraries such as Scikit-learn [Pedregosa et al., 2011] and NumPy [Van Der Walt et al., 2011].