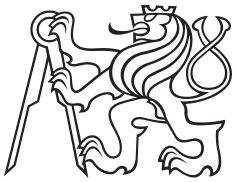

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Bachelor's Thesis

Image retrieval via CNNs in TensorFlow 2

Vyhledávání obrázků pomocí CNN v TensorFlow 2

Author: Jekatěrina Jaroslavceva

Supervisor: doc. Mgr. Ondřej Chum, Ph.D.
Field of study: Cybernetics and Robotics
May 2021

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Jaroslavceva Jekatěrina** Personal ID number: **474416**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Image Retrieval via CNNs in TensorFlow2

Bachelor's thesis title in Czech:

Vyhledávání obrázků pomocí CNN v TensorFlow2

Guidelines:

Study details and implementations of DELF [1], GeM [2] and eventually other approaches to image retrieval. Reimplement GeM in TensorFlow 2, train the network and perform the necessary experiments to compare the implementations. Perform further experiments with different training strategies and loss functions.

Bibliography / sources:

[1] Hyeonwoo Noh, Andre Araujo, Jack Sim, Tobias Weyand, Bohyung Han: Large-Scale Image Retrieval with Attentive Deep Local Features, ICCV 2017
[2] F. Radenovic, G. Tolias, O. Chum: Fine-tuning CNN Image Retrieval with No Human Annotation, TPAMI 2019

Name and workplace of bachelor's thesis supervisor:

doc. Mgr. Ondřej Chum, Ph.D., Visual Recognition Group, FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **24.01.2021** Deadline for bachelor thesis submission: **21.05.2021**

Assignment valid until: **30.09.2022**

doc. Mgr. Ondřej Chum, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

I would like to express my sincere gratitude to everyone who has assisted me along my academic path. I am grateful for the opportunity to work in the Visual Recognition Group within the Center for Machine Perception at the Czech Technical University. My greatest thanks belong to my supervisor Ondřej Chum Ph.D. and consultant Giorgos Tolias Ph.D. I would also like to thank Andre Araujo, Dan Anghel, and Jaeyoun Kim for insightful discussions, code review, and providing computational resources for the project.

Most importantly, I want to express deep gratitude to my family for their unwavering love and support that enabled me to pursue my goals.

Declaration

I declare that I wrote the presented thesis on my own and that I cited all used information sources in compliance with the methodical instructions about the ethical principles for writing an academic thesis.

In Prague, 10. May 2021

Abstract

This thesis addresses the problem of instance-level image retrieval in large-scale picture collections, intending to find the greatest number of images corresponding to a query. Convolutional neural networks (CNNs) have demonstrated their ability to provide effective descriptors for content-based image retrieval (CBIR). Given the current knowledge, we focused our efforts on utilizing fine-tuned CNNs for global feature extraction with the goal of using those for image retrieval problems.

Firstly, we examined several methods proposed to improve image retrieval, such as GeM [RTC18] and DELF [NAS+17]. As the main result of this thesis, an extendable and highly-customizable image retrieval framework based on the work of Radenović *et al.* [RTC18] was re-implemented in TensorFlow 2. This approach produces state-of-the-art retrieval results, while using relatively short descriptors. As a validation, we trained the networks on the *SfM120k* landmark images dataset and performed experiments on two image retrieval benchmarks (revisited *Oxford5k* and *Paris6k*). Different training strategies, network architectures and loss functions were used in the experiments. The final project code was successfully merged into the official Tensorflow repository managed by Google, as a part of the DELF [Tena] research library.

Keywords: CBIR, image retrieval, landmark retrieval, TensorFlow, CNN.

Abstrakt

Tato práce se věnuje vyhledávání největší množiny obrázků příslušící vyhledávanému objektu v rozsáhlých datových kolekcích. Konvoluční neuronové sítě (CNNs) prokázaly svoji schopnost poskytnout efektivní deskriptory pro vyhledávání obrázků. Zabýváme se tedy použitím vyladěných CNN k extrakcí globálních deskriptorů pro použití v problému vyhledávání obrázků (CBIR).

V práci jsme studovali současný stav poznání metod vyhledávání obrázků jako například GeM [RTC18] a DELF [NAS+17]. Klíčovým přínosem této práce je TensorFlow 2 implementace rozšiřitelného a vysoce přizpůsobitelného frameworku pro CBIR, založená na práci Radenoviće *et al.* [RTC18]. Tento přístup poskytuje výsledky srovnatelné s nejlepšími současnými metodami, přičemž ale používá relativně krátké deskriptory. Pro ověření výsledků jsme natrénovali sítě na *SfM120k* datasetu a provedli experimenty na dvou standardních datasetech (revisited *Oxford5k* a *Paris6k*). Během experimentů byly využity rozlišné trénovací strategie, architektury neuronových sítí a ztrátové funkce pro komplexní zhodnocení implementovaného přístupu. Finální zdrojový kód byl přidán do oficiálního repozitáře TensorFlow 2, jakožto součást výzkumné knihovny DELF [Tena].

Klíčová slova: CBIR, vyhledávání obrázku, vyhledávání orientačních bodu, TensorFlow, CNN.

Překlad názvu: Vyhledávání obrázků pomocí CNN v TensorFlow 2

Contents

Figures

Tables

Acronyms

AP	Average Precision
BoW	Bag of Words
CBIR	Content-Based Image Retrieval
CNN	Convolutional Neural Network
DELF	Deep Local Feature
FC	Fully Connected Layers
GeM	Generalized-Mean Pooling
HOG	Histogram-Oriented Gradients
kNN	k-Nearest Neighbor
MAC	Maximum Activations of Convolutions Pooling
mAP	Mean Average Precision
mP@K	Mean Precision at rank K
ROxford	Revisited Oxford Buildings dataset
RParis	Revisited Paris Buildings dataset
SfM	Structure from Motion
SGD	Stochastic Gradient Descent
SNN	Siamese Neural Network
SPoC	Sum-Pooled Convolutional Pooling
TBIR	Text-Based Image Retrieval
TF	TensorFlow
t-SNE	t-Distributed Stochastic Neighbor Embedding

Chapter 1

Introduction

Content-Based Image Retrieval (CBIR), particularly instance-level image retrieval, is a computer vision problem of searching similar images in sizeable unordered image collections. The term *content-based* reflects the fact that the search analyzes the information that can be derived from the image pixels alone, rather than the metadata, such as labels, text descriptions, or image attributes. In terms of this work, we only consider answering the queries made using input images containing the object of interest. This problem is referred to as *instance image retrieval*.

Image retrieval systems are usually one of two categories, text-based (TBIR) or content-based (CBIR). The former approach requires much human effort and time for image annotation, and is also subject to human perception and annotation quality. The latter approach pays greater attention to global and local image information, such as the color, shape, and texture of an image [HTS+06]. Therefore, as opposed to the image meta-search, CBIR is better suited for the task of image retrieval.

Image retrieval is one of the fundamental problems of academic research and industrial development since it can be applied in many practical applications, such as but not limited to object detection, visual place recognition, marketplace visual search systems in e-commerce [OSXJS16], [LLQ+16], searching in large image and video collections, or computer-aided diagnosis [QAAM17]. On a personal level, CBIR can be helpful in searching and organizing private photo collections [GNPS18]. Additionally, CBIR systems are developed in order to search for similar images on the Internet. Nowadays, there is a number of publicly available content-based image retrieval engines, allowing users

to perform reverse image search, including Google Image Search [Goo], Bing Visual Search [Mic], Yandex Image Search [Yan], Pinterest [Pin], *etc* [Wik]. Moreover, CBIR can be utilized to block or filter malicious web content, be used in security and criminal investigations or copyright protection (such as searching in fingerprint databases or trademark databases, respectively). Those mentioned above as well as multiple other applications, motivate researchers to look for efficient and accurate approaches for content-based image retrieval.

Figure 1.1: Example of perspective and scale variability for the same object instance.

Figure 1.2: Example of the occlusions of various degrees (some of the points of interest corresponding to the object are covered by the surrounding environment).

Figure 1.3: Example of varying lighting conditions caused by the day cycle and seasonal changes.

Figure 1.4: Example of visually similar but non-matching objects.

Despite years of effort, image instance-level retrieval remains a pressing problem. This is attributed to problems related, firstly, to the visual contents and, secondly, properties of the image itself. The former is due to the dramatic variations in visual appearance and the ambient environment caused by factors such as an arbitrary direction of cameras capturing the object (Fig. 1.1), significant changes across time and space, background clutter and occlusions (Fig. 1.2), unstable lighting conditions (Fig. 1.3), and differentiating non-matching objects that share similar appearances (Fig. 1.4). As far as the

properties of the image are concerned, resolution can also have a drastic effect on the retrieval accuracy, as some of the features can be lost in the re-scaling process. Therefore, an adequate image representation for retrieval has to be able to find the characteristics of the objects that are stable within the class and discriminative between the other classes. At the same time, the search needs to be fast and run *online* (i.e., upon the query's submission) on the query image, even in extensive image collections.

With such a considerable interest in the area of CBIR, new image retrieval models are being developed every year, creating a need for benchmarking of the newly introduced approaches. This was one of the reasons for the creation of the DELF [Tena] under the TensorFlow Model Garden [Tenb] by the Google research team. In this repository, a number of implementations of the state-of-the-art research models aim to demonstrate the best practices of network implementation so that users can take full advantage of TensorFlow for their research and product development.

Fostering progress in the problem of instance-level image retrieval was one of the motivations behind this thesis' goal of re-implementation of the [RTC18] research paper in TensorFlow 2. The resulting implementation was added to the Model Garden, allowing researchers to compare their approaches with other state-of-the-art practices. This bachelor project is supported by Google Research and produced in conjunction with the Landmark Retrieval Challenge 2020/2021 [Kag], organized by Google on the Kaggle platform. This competition challenges Kagglers to build models that retrieve all correct database landmark images for a given query. The landmark retrieval competition is a complicated challenge as it provides a dataset containing a large number of classes (there are a total of 15K classes in this challenge) and an unbalanced distribution of training examples. The code developed during the work on the thesis is publicly available and can be used as a building block for the upcoming competitions.

1.0.1 Contributions

This thesis is concerned with the unsupervised fine-tuning of CNNs for instance-level image retrieval. The project was supported by Google Research and it seeks to provide a state-of-the-art benchmark for image retrieval in Tensorflow 2. In particular, during the work on the project, the following contributions have been made:

1. An extendable and highly-customizable image retrieval framework based

on the architecture proposed by Radenović *et al.* in [RTC18] was re-implemented[1] in TensorFlow 2 as a part of the official DELF TensorFlow image retrieval library [Tena]. The code adheres to the stringent code conventions imposed by the repository standards. On top of that, we ensured high modularization, enabling the code to function and adapt to various environments and making it useful across projects.

2. In addition, the networks were trained and fine-tuned to achieve similar results as stated in the original paper. For the validation of the implementation, we performed the retrieval experiments on two image retrieval benchmarks (revisited *Oxford5k* and *Paris6k* [RIT+18]). Experiments were run under different training strategies, network architectures and loss functions. In particular, we experimented with parameters such as different global pooling strategies (Section 5.10) and loss functions (Section 5.12).

3. Further, we investigated the problem of reoccurring clusters in the *SfM120k* dataset (Section 5.13) and experimented with the speed properties of the GeM pooling layer (Section 5.11), which generalizes existing common pooling schemes for CNNs.

1.0.2 Organization of the Thesis

The remainder of the thesis is structured as follows: In Chapter 2, we address common techniques for content-based image retrieval and investigate state-of-the-art methods based on convolutional neural networks. The datasets used for training and assessment, as well as the evaluation protocols, are covered in Chapter 3. Chapter 4 introduces the background of the re-implemented CBIR method and delineates the extracted image representation. Moving on, we present the pipeline of the CBIR retrieval system based on [RTC18] and provide technical details. Chapter 5 shows our experimental findings and retrieval results in various training configurations and, finally, Chapter 6 draws conclusions.

[1] Data, networks, and code can be found at: `https://github.com/tensorflow/models/tree/master/research/delf`.

Chapter 2

Theoretical Background

2.1 Image Representation

CBIR is built around feature extraction. Most computer vision tasks do not use raw image data directly for two major reasons: high data dimensionality and redundancy. First of all, the high dimensionality of images makes it impractical to use whole images. Secondly, much of the information embedded in the image is redundant. Rather than using the entire picture, a representation of only the most essential information is extracted. The process of extracting an appropriate image representation is referred to as feature extraction, and the resulting representation is called a feature vector or a *descriptor*. As a result, feature extraction can be thought of as mapping the image from image space to the feature space.

In general, image features can be classified as either local or global. Local descriptors represent low-level scene properties (key points in the image), while global features describe the image as a whole to generalize the entire object. Although global features have been successfully employed for image retrieval, they possess an inherent weakness that limits their efficiency – the resulting descriptors cannot differentiate between different image parts (i.e., the object of interest and the background). As a result, they are usually less accurate for retrieving cluttered and complex scenes, as suggested by [HTS+06]. A combination of global and local features is shown to improve the accuracy of retrieval while incurring computational overheads. Global descriptors include, for example, shape matrices, invariant moments, Histogram-Oriented Gradients (HOG), and global CNN-based features. SIFT, SURF, LBP,

BRISK, MSER, FREAK [Işı14], and local deep neural features are some examples of local descriptors.

2.2 Standard CBIR Pipeline

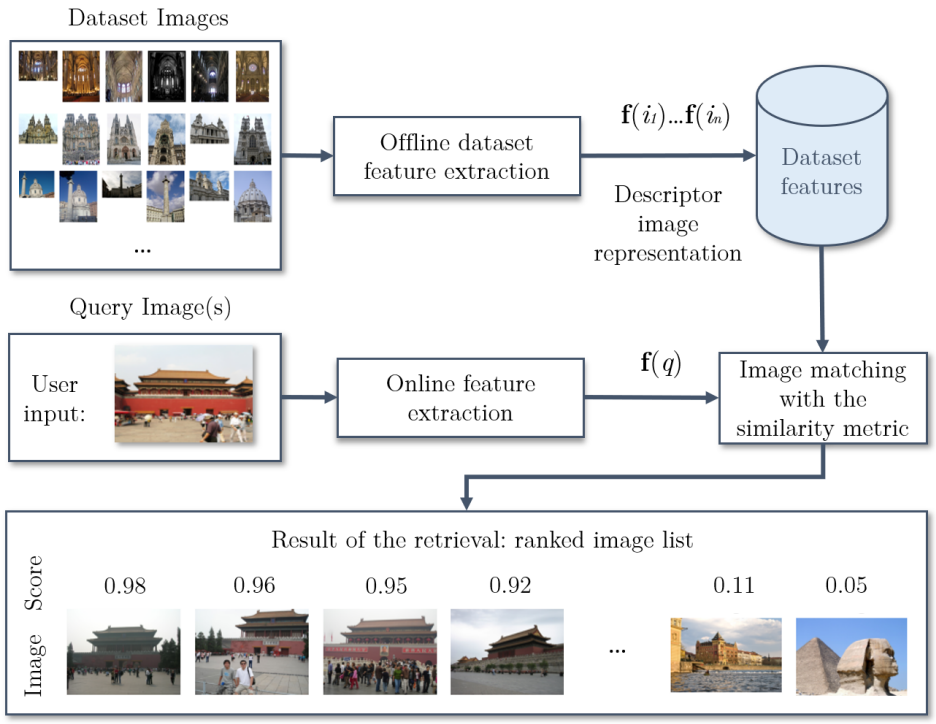

Figure 2.1: Abstraction of a typical CBIR image retrieval system. Different kinds of visual features, such as color, shape and texture, are extracted from the image pixels. The result is then a multidimensional feature vector that represents the image content. The set of feature vectors from all images in the collection in hand is stored in a feature database.

Image retrieval systems usually share the structure of the pipeline shown in Fig. 2.1. In the first step of the algorithm, image features are extracted from raw dataset images and converted into image *descriptors* that play the role of the digital signature of the images. All dataset image descriptors are then written into a collection structure. This is performed, usually, *offline*, meaning that the descriptors are already pre-computed upon the query's submission. For the test image, we extract the features in an *online* manner and perform image matching with the descriptors in the feature collection. Ideally, given an image query, the task of the image retrieval system is to retrieve all images matching this query from the provided image collection. However, a usual approach is to rank all images in the dataset according to how likely they are to be similar to the object of interest, choosing the first

N images as the pipeline output.

Contrary to image classification, where the object of interest is commonly assigned to a predefined class, image retrieval does not entail such assumptions. The number of possible object instances is indefinite, with new additions being possible over time. A major distinction can be observed between image retrieval and the process of adding new instances - that is as for every instance added to a standard classification scheme, a retraining of the classification algorithm would have to take place. In case of retrieval, on the other hand, if new images are added to the data collection, their feature vectors are extracted and added to the feature database in an *offline* manner without requiring any changes to the algorithm.

2.3 Content-Based Image Retrieval Approaches

This section offers an overview of the existing approaches to content-based image retrieval. CBIR methods are typically subdivided into two categories, namely, traditional and CNN-based, which are both discussed in the sections that follow.

2.3.1 Conventional Methods

CBIR methods were first introduced in the early 1990s. These low-level feature descriptor approaches, using, for example, color histograms [HKM+97], were primarily focused on color, texture or edge properties of images. Around the same time, researchers attempted to segment image color by separating images into spatial regions, producing a relationship between multiple color regions. However, these techniques were unable to capture the intricate image structures whose descriptors had been severely affected by image transformations and provided results that did not match the query image at an instance level. This led to the need of introducing more sophisticated approaches, incorporating more powerful image descriptors.

The first significant advances in the area of instance-level image retrieval were achieved by the aggregation of local features via SIFT [Low99] or SURF [BETVG08] image descriptors (and optionally complementary global features obtained by GIST [SI07] descriptors) by either employing the bag-of-visual-words (BoW) method [SZ03], or one of its more elaborate analogs.

These methods treat image features as words and are insensitive to local transformations and illumination changes [SDMP17]. Multiple methods were further employed to improve the accuracy of BoW. Geometric verification [PCI+07], query expansion [CPS+07], feature selection or creating compact image representations (VLAD [JDSP10], Fischer vectors [PLSP10]) are only some to mention.

2.3.2 CNN-Based Methods

Nowadays, since the success of Krizhevsky *et al.* [KSH12], the most advanced image representations for instance-level retrieval are based on CNNs. The idea that deep convolutional networks can extract high-level features in deeper layers led researchers to investigate ways to reduce the semantic gap in image retrieval [HTS+06]. High-level image representation provided by CNN-based approaches is not only efficient but also requires a minimal amount of memory resources.

The first CNN-based image retrieval approaches relied on networks pre-trained on the ImageNet [DDS+09] dataset, while using descriptors from the activations of fully connected (FC) layers. We can think of FC as a way to aggregate local information from convolutional layers – a characteristic that stems from the convolutional layers' attribute of having spatial dimensions, thus being able to carry local information. These methods were further improved by employing data augmentation, while bearing the cost of increased computational costs. It was later demonstrated that extracted features from both fully connected (global features) and convolutional layers (local features) could be used as image representations. Further approaches proposed using a range of pooling layers (MAC, SpoC, GeM) instead of FC layers, followed by the idea of spatial weighting on convolutional activations [BL15] and regional poolings such as (R-MAC) [TSJ16], which on top of retrieval allow localizing the object of interest in the picture. Proposed architectures start from pre-trained CNN models (i.e., VGG, ResNet, EfficientNet), which differ mainly in the top layers designed to aggregate local convolutional features, as illustrated in Fig. 2.2.

Later research [RTC18] discovered that fine-tuning, or starting the backbone with a pre-trained network and retraining it for a particular image domain, significantly improves networks' adaptability. When using a fine-tuned CNN model, descriptors are typically generated end-to-end, and the network produces the image representation without the need for additional explicit encoding or merging steps. Unlike classification tasks, where viewing same-class objects as equivalent is sufficient, in retrieval tasks, we care about

the similarity of objects even within the same class, making the concept of comparing images the crux of retrieval systems. A practical approach to learning relative distances between samples, a task that is often called *metric learning*, uses a Siamese-fashion network with an appropriate ranking loss. In these cases, however, training necessitates the use of both matching and non-matching picture pairs. As a consequence, the problem of data annotation becomes even more acute since labels must be provided for image pairs rather than separate classes, as in classification problems. Using manually labelled training data in a classification training fashion was one of the first effective fine-tuning methods for landmark image retrieval [BL15]. To compensate for the lack of human annotation, later methods suggested GPS-tagged image databases [AGT+16]. Furthermore, Radenović *et al.* [RTC18] proposed a completely automated solution to this issue, based on the geometry and camera positions available from 3D landmark data reconstruction.

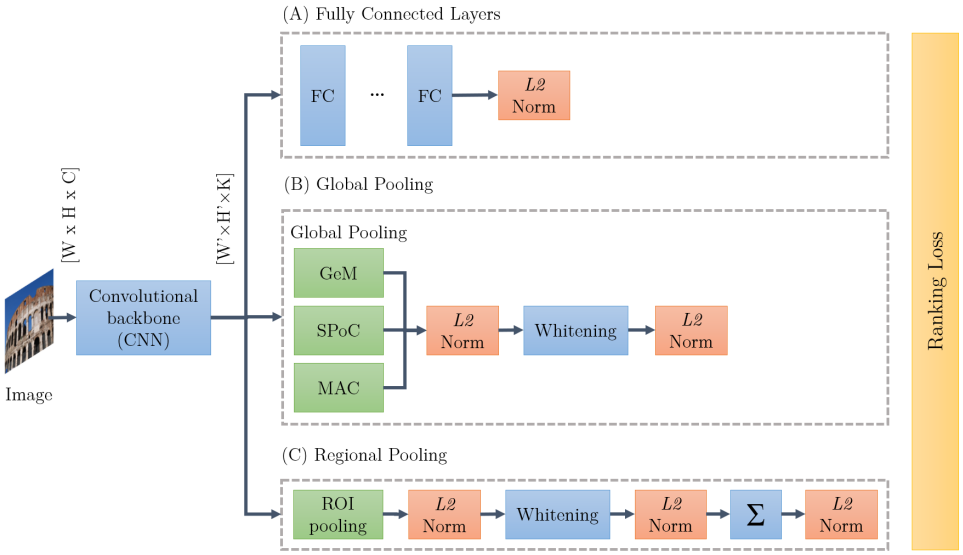

Figure 2.2: Examples of end-to-end CNN retrieval architectures. The approaches differ in how they build the final image representation after the pre-trained convolutional backbone: Local features can be aggregated with (A) fully connected layers or (B) via direct global pooling and additional post-processing (whitening); (C) To generate a set of region vectors, local descriptors can be locally pooled. An additional layer for region pooling, followed by region post-processing and sum-pooling, is added to accomplish this.

Chapter 3

Datasets for Metric Learning and Training Data Representation

A task of primary importance is generating training data for metric learning. The process of gathering a new image dataset is typically divided into web scraping and data cleaning. Web scraping is a procedure executed by a computer program that automatically searches for objects of interest through text queries on an image search engine. As a result, the search allows downloading a noisy collection of retrieved labeled images, some of which can often be incorrect or unrelated. Consequently, the outcome comprises substantial intra-class image variations (such as interior or exterior images of a particular building) and great diversity in image quality. Following web crawling, one of two data cleaning approaches is standardly used: manual or automated data cleaning. Manual cleaning entails examining all images by hand for moderate-sized datasets or using crowdsourcing platforms for large-scale datasets. Alternatively, one can use automatic data cleaning where the images' metadata, such as geolocation, is exploited to determine geometric consistency between images.

3.1 Structure-from-Motion (*SfM120k*) Dataset

The authors of the research paper *Fine-tuning CNN Image Retrieval with No Human Annotation* [RTC18] eliminate the need for manually annotated data and any assumptions on the training dataset. In essence, image labels are no longer required to find similar images. Instead, retrieving the desired images

is achieved by utilizing the geometry and estimated camera locations from 3D models that have been automatically reconstructed using a Structure-from-Motion (SfM) pipeline. The SfM pipeline receives an unordered image collection as input from which it attempts to create all possible 3D models. This is accomplished through local spatial verification, as described in [SRCF15]. Moreover, the SfM pipeline filters out the majority of mismatched images from the cluster, in conjunction to providing camera locations for all matched images. Redundant (overlapping) 3D models are excluded, but models reconstructing the same landmark from disjoint perspectives are considered non-overlapping.

For the initial noisy data, Radenović *et al.* [RTC18] used an extensive unlabeled image collection downloaded from Flickr with keywords of famous landmarks, cities, countries, and architectural sites. The largest reconstructed cluster includes 11042 images after data cleaning, while the smallest one solely contains 25. For clusters of 300 or fewer images, the number of training queries per cluster is 10% of its size, while for larger clusters, it is 30 images. In this manner, a total number of 181697 images (out of 91642 different clusters) is selected for training queries and 1691 for validation queries (out of 6403 clusters).

One of the challenges of the dataset is its high intra-class variability, which is caused by the creation of massive 3D reconstructions, including multiple points of interest, during the dataset creation. At the same time, not all overlapping clusters are eliminated from the dataset, introducing a high probability of selecting false negatives during the network training phase (Section 5.13).

3.1.1 Training Image Tuples

Once the annotated training dataset is available, it is critical to choose an adequate strategy for selecting training tuples. For training purposes, we create training tuples consisting of $(q, p(q), n_1(q), \cdots, n_N(q))$, where q represents a query image, $p(q)$ is a positive image that matches the query, and $n_i(q)$ are negative images that do not match the query (Fig. 3.1). These tuples are used to generate training image pairs, with each tuple representing $N+1$ pairs. We want negative images to be similar to query images, therefore generating the most significant loss and making training more effective. Moreover, due to discrepancies in views or intra-class heterogeneity, a random query selection of positive images from the same class is not obligated to include an appropriate positive pair.

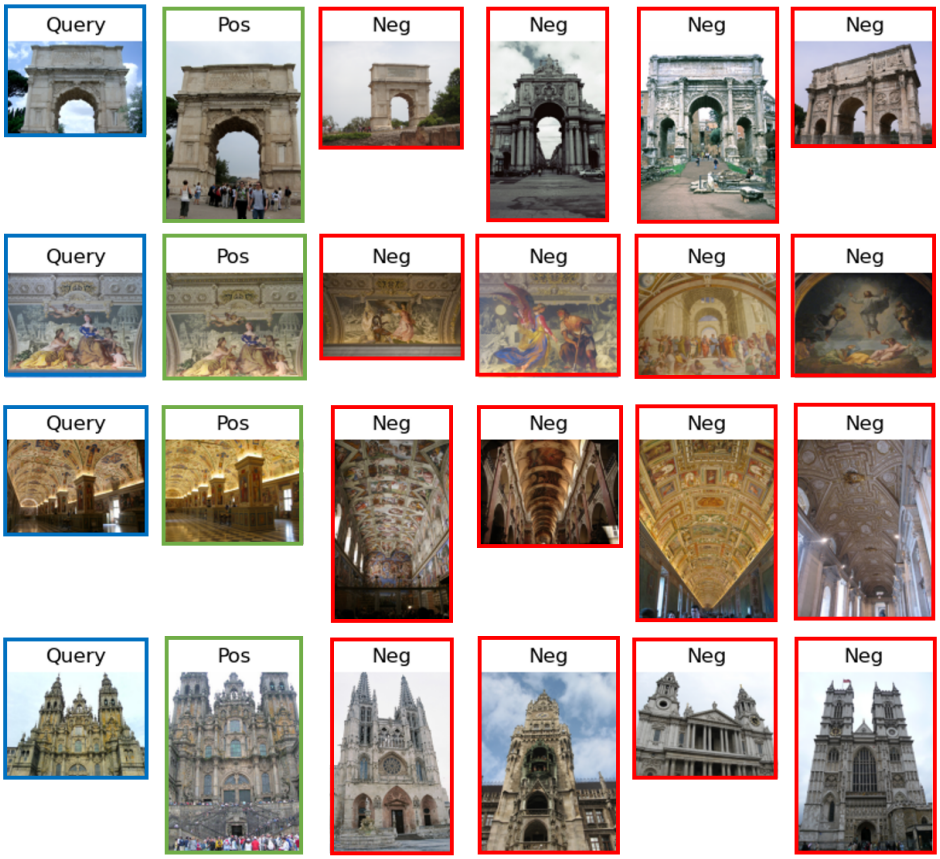

Figure 3.1: Examples of the training tuples consisting of 1 query image, 1 positive image and 4 hard-negative images. Each row represents a separate training tuple. Positive images are hardcoded, while negative images are mined for a particular query.

Hard-Negative Mining

As previously mentioned, randomly sampling image tuples is an inefficient strategy, as many of them may already fulfill the contrastive and triplet loss margin criteria. In other words, the CNN model's weights remain unchanged, and no learning occurs since an error is not created, nor are gradients backpropagated. A typical technique for dealing with negative pairs entails iterating over non-matching images that are "hard" negatives, meaning they are similar in the descriptor space and incur a high loss. Since the clusters are presumed to be non-overlapping, negative examples are chosen from clusters other than that of the query image. Out of all non-matching images, the k-Nearest Neighbors (kNN) are chosen, out of which the subset containing at most one negative image per cluster is selected for training. Re-mining is repeated every N epochs of model training in order to train the network with even more complicated cases, as hard negatives are chosen based on the current CNN parameters.

Positive Image Pairs

A typical technique for selecting positive pairs consists of sampling images from the same class, 3D point, or clusters. For instance, some methods choose positive pairs with the shortest distance within the embedding space. To avoid sampling very similar images, Radenović *et al.* use a matching pipeline from 3D reconstruction to single out the positives sharing the minimum amount of local matches while having the closest camera positions to the query. This process ensures that selected positives depict the same object as the query while guaranteeing variability of perspectives. As far as the *SfM120k* dataset is concerned, hard positive pairs are chosen offline and are fixed throughout the training phase.

Once matching and non-matching pairs are selected, the model can be trained in a Siamese manner (Section 4.1) with ranking loss to distinguish similar images from non-similar ones, resulting in accurate vector representations for each image.

3.2 Revisited *Oxford and Paris Buildings* Datasets

One of the standard data collections for image retrieval analysis includes the *Oxford and Paris Buildings* datasets representing the major tourist attractions in Oxford and Paris. Radenović *et al.* [RIT+18] recently renovated these datasets, and we use the newly introduced revisited *Oxford* (further *ROxford*) and *Paris* (further *RParis*) datasets to evaluate the search performance of the networks. Both databases include 70 queries as well as 4993 and 6322 database photos, respectively. A bounding box is annotated on the object whose other images are to be retrieved, evaluating the object's visibility out of four possible labels: *Easy* (the object is clearly visible and there is no significant perspective change), *Hard* (25% of the object is clearly visible with a possible change of the original viewpoint), *Negative* (object of interest is missing) and *Unclear* (the object of interest is heavily occluded or depicts a different side of the query object). All photos that do not appear on the list of possible positives are automatically marked as *Negative* [Rad19].

3.2.1 Performance Evaluation

Image retrieval is usually formulated as a ranking problem, with the aim of ordering database images in decreasing order of similarity to the object of interest [RARS19]. In order to evaluate the image retrieval's quality, the top N retrieval result is obtained for each of the query images and their relevance is evaluated based on whether they display the same landmark. The mean average precision (mAP) is a widely used metric the evaluating model performance in retrieval and object detection tasks. It is computed by averaging the precision (AP) through all of the test set's queries.

$$mAP = \frac{\sum_{q=1}^{Q} AP(q)}{Q}. \tag{3.1}$$

In the equation above, the variable Q represents the number of query images. It is worth noting that average precision (AP) computation does not employ the finite sum method, widely used in the literature on information retrieval. Instead, the implemented AP averages two adjacent precision points (by interpolation), then multiplies by the recall step, allowing to integrate over the precision-recall curve. This is the convention for the revisited Oxford and Paris datasets. In addition to mAP, the mean precision at rank K (mP@K) can be evaluated, considering only top K retrieval results.

We use a standard assessment method for the revisited Oxford and Paris datasets. As far as queries are concerned, only the regions of test images surrounded by bounding boxes are used. By treating labels (*Easy*, *Hard*, and *Unclear*) as positives or negatives, or ignoring them, three evaluation setups of varying difficulty are defined for these datasets:

- **Easy:** Easy images are considered positives, whereas Hard and Unclear are ignored.
- **Medium:** Easy and Hard images are considered positive, while Unclear are ignored.
- **Hard:** Hard images are considered positive, while Easy and Unclear are ignored.

In the scenarios, where the system does not return any positive images for a particular query as an output (for example, when only easy images are retrieved for a query under the Hard evaluation protocol), said query is eliminated from the mAP calculation.

Chapter 4

Network Architecture and Image Descriptors

In this chapter, we discuss the architecture of the re-implemented end-to-end retrieval network based on the work of Radenović *et al.* [RTC18], as well as explain the extracted image representation obtained as the output of the network. As previously mentioned, it is possible to obtain a vector representation of an image by extracting local features from a pre-trained CNN-backbone before the pooling layer. Following that, the authors perform a global pooling operation to reduce dimensionality, normalization procedures, and post-processing, which result in the output of the image descriptor. Then, it is possible to fine-tune the network to boost its performance for a specific domain using Siamese metric learning with an appropriate ranking loss (such as triplet or contrastive losses) by training a network with positive and negative image pairs. Furthermore, we discuss how whitening can be applied, either learned end-to-end with the network or added as a post-processing step after training.

4.1 Metric Learning with Siamese Architectures

Siamese Neural Network [KZS15], or simply SNN, is a type of neural network that uses the same architecture and weights in multiple instances of the same model (Fig. 4.1). This architecture demonstrates its strength when learning with limited data and successfully addresses the issue of adding new instances to the dataset without the need for constant network re-training. This means

that once a network has been tuned, we can use powerful discriminative features to generalize the network's predictive abilities not only to new data but to entirely new data classes. Aside from these benefits, the fundamental justification for using SNNs for retrieval tasks is the concept of the retrieval problem. Precisely, in order to find the most similar images to a given query, an image retrieval system must calculate a similarity score between the images in the test set and the query. SNNs allow direct training of networks for metric learning, which is the process of learning the similarity function over objects, thus providing a solution to the retrieval task.

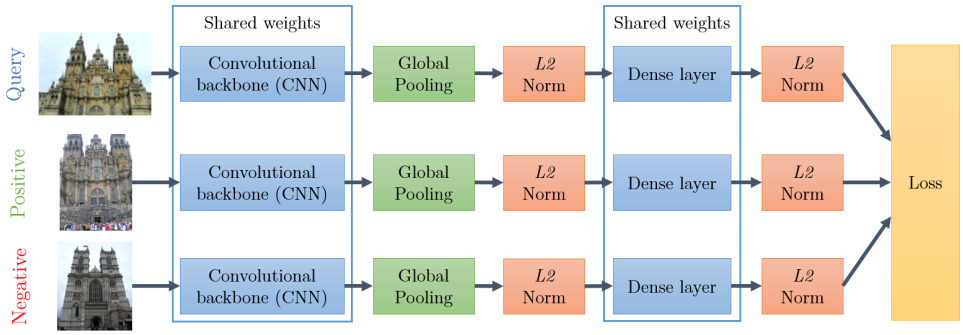

Figure 4.1: Used Siamese network architecture. During the training phase, image triplets are sampled, and a ranking loss (triplet or contrastive) is applied simultaneously to every image in the training tuple.

As previously stated, the aim of SNNs is to build several stream networks that all share the same parameters. The network receives a labeled tuple of images as input (query, positive, and negatives). After the network generates an embedding for those images, we compute their distance (in our case, Euclidean distance) between sample representations. The model is then optimized to minimize the distance for similar samples and maximize it for dissimilar samples. As a result, the model learns similar representations for samples defined as positives and distant representations for samples defined as negatives. In our case scenario, learning in the network can be achieved with either triplet or contrastive loss. Losses are initially calculated in a pair-wise manner and subsequently aggregated for the whole training tuple.

4.1.1 Triplet Loss

The most common ranking loss used with Siamese architectures is the triplet loss. Similar to the contrastive loss, it operates with triplets of samples and consists of:

1. **Query:** anchor sample, which is the reference of the triplet.
2. **Positive:** a positive sample similar to the anchor.
3. **Negative:** a negative sample, which is dissimilar to the anchor.

The triplet loss is formally defined as follows [AGT+16]:

$$\mathcal{L} = \max\left(0, d^2\left(\bar{\mathbf{f}}(q), \bar{\mathbf{f}}(p)\right) - d^2\left(\bar{\mathbf{f}}(q), \bar{\mathbf{f}}(n)\right) + \alpha\right), \tag{4.1}$$

where $d(a, b)$ is the Euclidean distance between vectors a and b, α is the loss margin parameter, q, p, and n are the representations of the query, the positive, and the negative images, respectively. Here and in the following sections, $\bar{\mathbf{f}}(i)$ is the $\mathcal{L}2$-normalized descriptor (the output of the network) of image i.

The negative vector forces learning in the network, while the positive vector serves as a regularizer, as seen in equation 4.1. The margin ensures that the model does not linger enlarging the distinction between the positive and negative samples of a triplet when it already does so properly, allowing it to concentrate on more challenging triplets. In other words, the margin determines whether a triplet's network efficiency is already optimal.

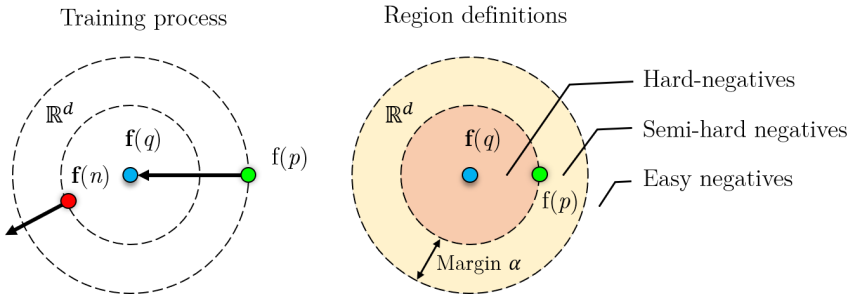

Figure 4.2: Visualization of triplet loss. The learning process is depicted on the left side of the picture, in which the model learns to minimize distance for similar samples and maximize distance for dissimilar samples. The hard-, semi-, and easy-negative samples are separated on the right side based on their distance from the query feature vector.

4.1.2 Easy, Hard, and Semi-Hard Triplets

During the loss computation, three forms of triplets (see Fig. 4.2) can be observed depending on the distances between triplet samples:

- **Easy triplets:**

$$d\left(\bar{\mathbf{f}}(q), \bar{\mathbf{f}}(n)\right) > d\left(\bar{\mathbf{f}}(q), \bar{\mathbf{f}}(p)\right) + \alpha \tag{4.2}$$

  Easy triplets are the ones where the negative sample is sufficiently far from the anchor sample in the descriptor space compared to the positive sample. In this case, the gradients and the loss are both zero. As a result, when training with easy triplets, network weights do not get updated and the network efficiency is not improved. Hence, it would be optimal to eliminate easy triplets from network training data to save computational time.

- **Semi-hard triplets:**

$$d\left(\bar{\mathbf{f}}(q), \bar{\mathbf{f}}(n)\right) < d\left(\bar{\mathbf{f}}(q), \bar{\mathbf{f}}(p)\right) + \alpha \tag{4.3}$$

  As for the semi-hard triplets, the negative sample representation is further away from the anchor than the positive descriptor, but the difference is still less significant than the loss margin. As a result, the network has to widen the gap between the positive and the negative samples.

- **Hard triplets:**

$$d\left(\bar{\mathbf{f}}(q), \bar{\mathbf{f}}(n)\right) < d\left(\bar{\mathbf{f}}(q), \bar{\mathbf{f}}(p)\right) \tag{4.4}$$

  In hard triplets, the negative sample descriptor is closer to the query sample than the positive descriptor. This suggests that the network is unable to differentiate between the positive and negative samples in hard triplets. Therefore, those triplets have the highest loss and are the best candidates for network training.

4.1.3 Contrastive Loss

The training input for contrastive loss [CHL05] is a tuple of image descriptors corresponding to the query, positive and negative, similar to the triplet loss. The following formula is used to measure the pair-wise contrastive loss:

$$\mathcal{L}(i,j) = \begin{cases} \frac{1}{2} d^2\left(\bar{\mathbf{f}}(q), \bar{\mathbf{f}}(j)\right), & \text{if } Y(q,j) = 1 \\ \frac{1}{2}\left(\max\left(0, \alpha - d\left(\bar{\mathbf{f}}(q), \bar{\mathbf{f}}(j)\right)\right)\right)^2, & \text{if } Y(q,j) = 0 \end{cases} \tag{4.5}$$

Label $Y(i,j) = 1$ indicates that the pair of images is matching, whereas $Y(i,j) = 0$ is used for a non-matching pair, and α is a margin hyperparameter that determines when non-matching pairs are separated by a wide enough distance for the loss to disregard them.

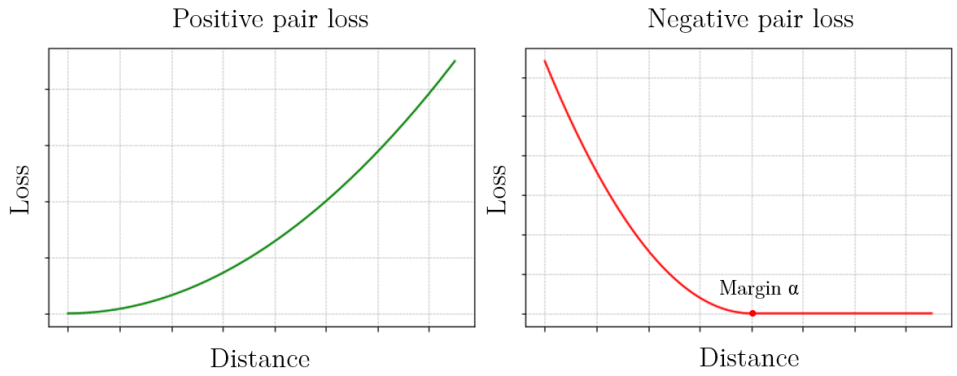

Figure 4.3: Plots of the pair-wise contrastive loss based on image descriptor distance. The positive pair loss is depicted on the left plot, while the negative pair loss is depicted on the right.

4.2 Network Architecture

In practice, when initializing the network for image retrieval, one of the popular pre-trained architectures for classification tasks (such as ResNet [HZRS16], VGG [SZ14], or EfficientNet [TL19]) is used as the network's backbone. The fully connected layers of such architectures are discarded, resulting in a fully convolutional backbone. Then, given an input image of the size $[W \times H \times C]$, where C is the number of channels, W and H are image width and height, respectively; the output is a tensor X with dimensions $[W' \times H' \times K]$, where K is the number of feature maps in the last layer (Fig. 4.4). Tensor X can be considered as a set of the input image's deep local features. For deep convolutional features, the simple aggregation approach based on global pooling arguably provides the best results. This method is fast, has a small number of parameters, and a low risk of overfitting. Keeping this in mind, we convert local features to a global descriptor vector using one of the retrieval system's global poolings (MAC, SPoC, or GeM). After this stage, the feature vector is made up of the maximum activation per feature map with dimensionality equal to K. The final output dimensionality for the most common networks varies from 512 to 2048, making this image representation relatively compact (Tab 4.1).

Vectors that have been pooled are subsequently$\mathcal{L}2$-normalized. The obtained representation is then optionally passed through the fully connected layers (global whitening discussed in Section 4.5) before being subjected to a new $\mathcal{L}2$ re-normalization. The finally produced image representation allows comparing the resemblance of two images by simply using their inner product.

Supported Network Architecture	Number of feature maps K in the last layer	Occupied space per one image representation [B]
VGG16	512	2048
VGG19	512	2048
ResNet50	2048	8192
ResNet101	2048	8192
ResNet101V2	2048	8192
ResNet152	2048	8192
DenseNet121	1024	4096
DenseNet169	1664	6656
DenseNet201	1920	7680
EfficientNetB5	2048	8192
EfficientNetB7	2560	10240

Table 4.1: Table of supported TensorFlow network architectures, along with their output descriptor dimensions and memory requirements.

The network is then fine-tuned after being initialized with the *ImageNet* [DDS+09] weights. As demonstrated by Radenović *et al.* in [RTC16], fine-tuning accounts for much better retrieval accuracy since the CNN fires fewer to ImageNet classes, such as vehicles, people, and geological formations. Additionally, the fully connected whitening layer is pre-computed from the same training data before the model training, which increases network training speed and accuracy.

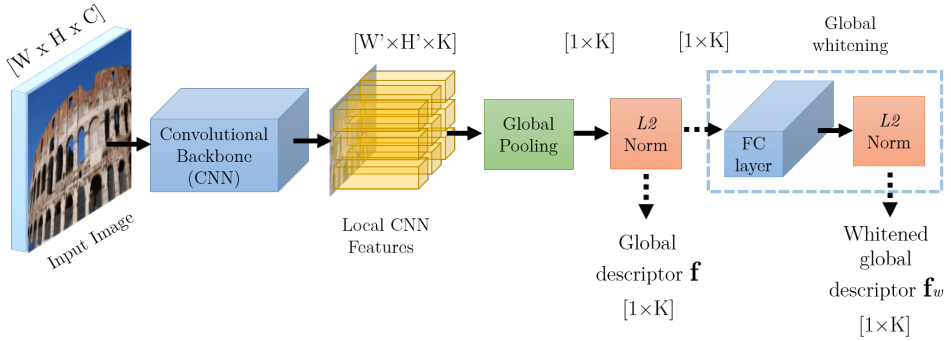

Figure 4.4: Network pipeline: The convolutional backbone processes an input image with dimensions of $[W \times H \times C]$. The CNN produces a 3D tensor with $[W' \times H' \times K]$ dimensions representing the set of image's local features. In the next step, global pooling is applied, and the obtained vector is $\mathcal{L}2$-normalized. Finally, the global descriptor $\boldsymbol{f}$ is optionally whitened by passing through the fully connected (FC) layer and subsequently re-normalized.

4.3 Global Pooling

The output of the fully convolutional network is tensor X_i - a set of all $W' \times H'$ activations for feature map $i \in \{1...K\}$ (Fig. 4.3). Instead of down sampling feature map patches, global pooling down samples the entire feature map to a single value in a layer-wise manner, reducing the dimensionality of the network output by producing solely a K-dimensional vector. On top of that, global pooling drops redundant spatial information, resulting in an increase in geometric invariance.

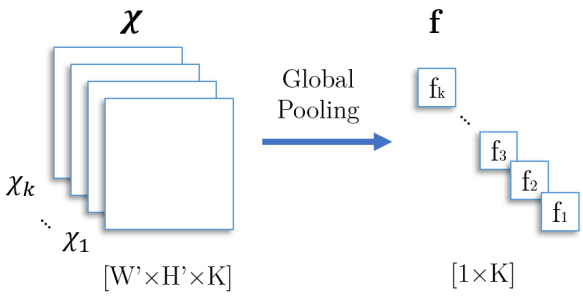

Figure 4.5: Global pooling dimensionality.

In earlier work, a variety of methods for local features aggregation were used – these range from fully connected layers to various global-pooling layers and regional aggregation strategies. In terms of this work we experimented with MAC, SPoC, and GeM global pooling layers.

- **Maximum Activations of Convolutions (MAC)** [TSJ16] is simply constructed by max-pooling over all the dimensions per feature map. The method follows a similar strategy as the local Max pooling of CNN pooling layers, but uses a global kernel over each feature map:

$$\mathbf{f} = [f_1, ..., f_K]^T, \text{ with } f_i = \max_{x \in X_i} x. \tag{4.6}$$

  Otherwise stated, we take the maximum value for each feature map X_i to get a K-length long vector representation of the image.

- **Sum-Pooled Convolutional features (SPoC)** is given by [BL15]:

$$\mathbf{f} = [f_1, ..., f_K]^T, \text{ with } f_i = \frac{1}{|X_i|} \sum_{x \in X_i} x. \tag{4.7}$$

  In other words, for each feature map X_i, the average value is taken.

- **Generalized-Mean (GeM)** pooling layer generalizes MAC and SPoC poolings and is based on a generalized-mean with learnable parameters.

$$\mathbf{f} = [f_1, ..., f_K]^T, \text{ with } f_i = \left(\frac{1}{|X_i|} \sum_{x \in X_i} x^{p_i} \right)^{\frac{1}{p_i}}. \tag{4.8}$$

  For p_i equal to 1, GeM becomes the average pooling, while as p_i approaches ∞ it converts to maximum pooling. The pooling parameter p_i can be either manually set or learned since it is differentiable and can be backpropagated. In terms of this work, we set parameters p_i to be identical for each feature map.

4.4 Normalization

In order to compare images at the level of $\mathcal{L}2$ norms and obtain a global view of image similarities, the last network layer comprises an $\mathcal{L}2$-normalization layer. $\mathcal{L}2$-normalization modifies the values in the vector such that the sum of the squared values in a given axis becomes equal to one:

$$\bar{\mathbf{f}} = \frac{\mathbf{f}}{\sqrt{\max\left(\sum_{f_i \in \mathbf{f}} f_i^2, \varepsilon\right)}}. \tag{4.9}$$

In the equation above, ε is the lower bound value for the norm. This way, the $\sqrt{\varepsilon}$ will be used as the divisor for small norms, preventing mathematical precision errors. $\mathcal{L}2$-normalized network output is denoted as $\bar{\mathbf{f}}$ and constitutes the image descriptor, allowing the similarity between two images to be evaluated with the inner product.

4.5 Descriptor Whitening

Whitening is a data-processing procedure that removes redundant information from image representation data. Whitening decreases the degree of correlation between adjacent pixel or feature values, which can be high in real world scenarios. In the network implementation, we provided two ways of achieving the descriptor whitening: end-to-end learned whitening in a form of a fully connected layer and whitening as a post-processing procedure learned in a discriminative manner.

4.5.1 Whitening as a Fully Connected Layer

The former approach suggests adding a fully connected layer as the last layer in the network and training it end-to-end with the rest of the network. The layer is represented in the following manner:

$$\mathbf{Y} = \mathbf{A}\mathbf{X} + \mathbf{b}, \tag{4.10}$$

where $\mathbf{X}$ is the $\mathcal{L}2$-normalized pooled fully convolutional network output (i.e., descriptors $\bar{\mathbf{f}}$ of the inputted images), $\mathbf{A}$ is the kernel matrix, $\mathbf{b}$ is the bias, and $\mathbf{Y}$ are the whitened descriptors.

4.5.2 Whitening as a Post-Processing Step

The other approach is applying whitening as a post-processing step for image descriptors:

$$\mathbf{Y} = \mathbf{P}^T(\mathbf{X} - \boldsymbol{\mu}), \tag{4.11}$$

where $\mathbf{P}$ is the projection matrix, and $\boldsymbol{\mu}$ is the mean descriptor vector used to perform centering. This approach allows pre-computing $\mathbf{P}$ and $\boldsymbol{\mu}$ on the training data and further using these values in the evaluation process. To achieve this, we begin by obtaining the descriptors of the images in the training dataset by a forward pass through the convolution-only network with subsequent pooling and normalization, and calculating the mean descriptor vector $\boldsymbol{\mu}$ over the dataset. Then, we are able to calculate the projection matrix in the following manner:

$$\mathbf{P} = \mathbf{C_S}^{-\frac{1}{2}} \mathrm{eig}\left(\mathbf{C_S}^{-\frac{1}{2}} \mathbf{C_D} \mathbf{C_S}^{-\frac{1}{2}}\right), \tag{4.12}$$

where $\mathbf{C_S}^{-\frac{1}{2}}$ is the interclass covariance matrix calculated for the positive pairs as:

$$\mathbf{C_S} = \sum_{Y(i,j)=1} \left(\bar{\mathbf{f}}(i) - \bar{\mathbf{f}}(j)\right) \left(\bar{\mathbf{f}}(i) - \bar{\mathbf{f}}(j)\right)^T, \tag{4.13}$$

And $\mathbf{C_D}$ is calculated in a similar manner for the non-matching pairs:

$$\mathbf{C_D} = \sum_{Y(i,j)=0} \left(\bar{\mathbf{f}}(i) - \bar{\mathbf{f}}(j)\right) \left(\bar{\mathbf{f}}(i) - \bar{\mathbf{f}}(j)\right)^T. \tag{4.14}$$

Moreover, whitening allows to reduce the dimensionality of the final image descriptors. By using only the eigenvectors corresponding to the D largest

eigenvalues, we can minimize the descriptor dimensionality to D dimensions. When applying the mentioned approach, the network is optimized first, followed by the whitening parameters $\mathbf{P}$ and $\boldsymbol{\mu}$. After the fine-tuning of the CNN is completed, whitening acts as a post-processing stage.

4.5.3 Conversion Between the Whitening Representations

Whitening consists of vector shifting and transformation, which are modeled in a straightforward manner with a fully connected layer, as discussed above. We observed that using a pre-computed whitening transformation to initialize the fully connected layer enhances network convergence time and accuracy. Hence, we convert the discriminatively learned projection matrix $\mathbf{P}$ and the mean vector $\boldsymbol{\mu}$ to the FC layer by simply setting the kernel matrix $\mathbf{A} = \mathbf{P}^T$ and bias equal to the projected shifting vector $\mathbf{b} = -\mathbf{P}^T\boldsymbol{\mu}$. This substitution stems from: $\mathbf{Y} = \mathbf{P}^T(\mathbf{X} - \boldsymbol{\mu}) = \mathbf{P}^T\mathbf{X} - \mathbf{P}^T\boldsymbol{\mu} = \mathbf{A}\mathbf{X} + \mathbf{b}$.

4.6 Image Similarity Search

After the network has been trained, the retrieval of images is as simple as conducting an exhaustive Euclidean search over database descriptors in relation to the query descriptor. The inner product of all the database vectors with the query image representation is computed, and the results are sorted from highest to lowest. The images with the highest scores are evaluated as most similar, whereas those with the lowest scores are assessed as least similar. The representation is rather compact and, therefore, suitable for efficient *online* database searching. For example, when using the *ResNet101* backbone, each descriptor requires 2048 *float32* numbers, making up 8192 bytes per image to be stored.

To boost the system's scale invariance, which has already been learned to some extent during training, we use multi-scale processing of the submitted query, which adds additional invariance without requiring any further learning. This is achieved by feeding the network with differently sized query images. Finally, the multiple descriptors are pooled and re-normalized, yielding a scale-invariant representation of the query image.

Chapter 5

Image Retrieval Experiments

This chapter goes through the specifics of the training's implementation, examines various aspects of the re-implemented instance-based image retrieval method, and compares different training setups. Additionally, we further experiment with the *SfM120k* dataset's re-occurring cluster problem and various global pooling strategies.

5.1 Training Configuration

This work was implemented in *Python*, making use of the TensorFlow 2 and Keras frameworks. For the sake of this project, the Google Research team generously provided the experimental environment on the Google Cloud Platform, composed of Intel(R) Xeon(R) processors, four instances of Tesla P100-PCIe GPU with 16 GB of memory, Debian GNU/Linux 10, TensorFlow version 2.2, and CUDA version 11.0.

The implemented framework allows for a highly customizable training pipeline. To name a few its parameters, the user can choose from multiple backbone network architectures and initialization options, including but not limited to momentums, weight decays, and one of the three supported global pooling types (MAC, SPoC, and GeM). In addition, we provide the option of training the network end-to-end with an FC whitening layer or applying whitening as a post-processing procedure after the training is completed. Moreover, the system supports several optimizers (SGD or Adam) and loss

functions (triplet and contrastive losses) with appropriate hyperparameters. As far as the training tuples are concerned, the user can set the number of query images per training epoch, the size of the negative pool from which hard-negative images are chosen, and the number of negative images N per training tuple $(q, p, n_1, \cdots, n_N)$. Moreover, the framework in place supports the ability to resume training from previously saved checkpoints and extract the trained model in a format required by the Kaggle Landmark Retrieval Challenge.

5.2 Implementation Details

When training a Siamese network, there is a few practical considerations to keep in mind. Firstly, we need to consider training tuple sampling, a factor that was previously discussed in Section 3.1.1. Random sampling of the training data would almost always yield triplets that are too easy, and therefore do not incur any loss nor network weights updates. As was previously mentioned, the positive pairs are predefined and are not altered during network training. In contrast, negative samples are chosen by the network on the fly with a bias towards hard triplets generating a high loss. It is worth noting that, in principle, the set of suitable candidates for the non-matching pairs should be re-calculated every time the model is updated, which is highly time-consuming. However, in practice, most of the hard triplets remain hard even after the model weights get updated several times. Therefore, we can only update the set of successful candidates every N iterations. In the case of this work, the pre-selected setup updates the training data every training epoch, which by default consists of 2000 query images, with one positive pair and 5 negative samples, resulting in a total of 12000 triplets per epoch. With the batch size of 5, this results in 400 network weight updates per epoch. Furthermore, the network implementation allows for the frequency of such training data re-mining to be set, resulting in a substantial reduction in time demands.

Another factor to consider is the amount of memory exploited during training since we use high resolution images (resized to 1024 pixels with the aspect ratio preserved) with up to seven network streams (one for query, one for positive, and five for negative images) at once. GPU memory constraints are partially overcome by associating each query with a tuple that includes a query in conjunction with corresponding positive and n hard-negative images. Consequently, the network is fed with $n+2$ photos, which represent n training pairs. Granted that we are using triplets rather than tuples, the naive method would require n triplets in need of pre-processing and memory storage of $3n$ images instead of solely $n+2$. Under our setup, we could only fit one training

tuple in memory on a single 16 GB GPU while using the VGG16 or VGG19 architectures. To perform weight updates in the network with a batch of size greater than one, we follow two steps. Initially, we compute and aggregate the gradients of the loss concerning the network parameters for each training tuple sequentially. Then, we perform the actual update after every B tuples, with a default batch size B set to 5.

The problem becomes even more complicated when using more extensive networks like ResNet101. With such architectures. For instance, there is not enough memory with such architectures to process even a single training data tuple consisting of seven full-sized images. This led to the necessity to use an alternative method detailed in Algorithm 1 in lieu of reducing the image dimensionality, which usually results in a loss of pivotal information. Rather than processing all of the images in a tuple simultaneously, this method allows to process them sequentially using a single stream. This process produces identical gradients but provides a significant reduction in memory requirements at the cost of lowering computational efficiency. This way, we are able to reduce memory requirements from the original 15 GB down to a sole estimate of 6 GB per training batch. Thus, deeper architectures can be used to train the retrieval network without reducing the size of the training images.

5.3 Computational Cost

In the default setup, to perform the fine-tuning, we initialize the network with pre-trained convolutional layers of ResNet101. The fully connected whitening layer is learned end-to-end and is pre-initialized with the pre-computed weights (Section 4.5.3). We use a $5 \cdot 10^{-7}$ learning rate, exponential decay with a 0.01 exponent, a momentum of 0.9 for the Adam optimizer, weight decay of 10^{-6}, and loss margin for triplet loss of 0.85. The default batch size is set to 5 training tuples, and we use GeM global pooling as the aggregation layer. Training images are resized to a maximum resolution of 1024 pixels for the larger image dimension while maintaining the original aspect ratio. Training lasts at most 100 epochs, with the best network being chosen based on validation set mean Average Precision (mAP). On a single GPU, fine-tuning ResNet101 for one epoch, including hard-negative re-mining with a pool size of 20000 images and number of queries equal to 2000, takes on average 1.3 hours. In particular, hard negative re-mining under the default setup can take up to 42 minutes, while training requires around 36 minutes.

The default model converges after approximately 40 epochs or 2.5 days

Algorithm 1: Memory efficient model training

Define Process_Training_Tuple
Input: $(Q, P, N_1, \cdots, N_n)$, where
Q: Query image,
P: Positive image,
N_i: Negative image.
Result: Updates model weights for one training batch.
Main:
accumulated_gradients = list of zeros of the size of network trainable parameters
// We load tuples into memory consequently.
for *tuple in batch_size* **do**
    // Record gradients and loss through the network.
    Start Record Gradients
    descriptors = empty list
    for *img in tuple* **do**
        // Compute descriptor vector for each image by a forward pass through the network and append them to the list of descriptors.
        descriptors.append(*model*(*img*, training=True))
    end
    // Based on the knowledge that the first image in the tuple is a query, the second is the positive and the rest are negatives, calculate one of the ranking losses.
    // Losses are initially calculated in a pair-wise manner as in Equations 4.1 and 4.5 and subsequently aggregated for the whole training tuple.
    loss = ranking_loss(*descriptors*)
    grads = **Stop** Record Gradients
    accumulated_gradients.add(*grads*)
end
optimizer.apply_gradients(*accumulated_gradients*)

of training when initialized with the pre-trained classification weights and pre-computed whitening layer. If the whitening layer is not pre-initialized and is instead initialized with random weights, the convergence rate is slower, taking about 75 epochs or approximately 5 days of training. After the network has been fine-tuned, extracting the descriptor of one image on a single GPU is performed within 130 milliseconds, corresponding to an image rate of about 8 images per second. Computing the similarity between two images boils down to computing the dot product between their representations, which is highly efficient. A standard processor can perform millions of such comparisons per second.

5.4 Retrieval Results

Figure 5.1: Example of Top 50 retrieval results for a query image from *ROxford* dataset. Easy, hard, unclear (junk), and negative retrieved images are represented with the frames of corresponding colors. Retrieval results are sorted based on the relevance score from left-to-right and top-to-bottom.

As discussed in Section 3.2, we evaluated we evaluated the trained networks on revisited Oxford and Paris datasets. All training clusters, containing images from the *ROxford5k* or *RParis6k* test datasets, are omitted for the training phase, resulting in no overlap between the training and testing data.

According to the top 50 retrieval results for one of the query images shown in Fig. 5.1, the network generalizes properly and is unaffected by over-fitting. We can see that out of 50 images retrieved from the *ROxford* dataset for the given query, we got 31 easy, 8 hard, 6 unclear, and 5 negative examples. As predicted, easy examples have higher retrieval scores than the majority of the other categories (i.e., in Fig. 5.1, easy images are generally closer to the beginning of the retrieved list). Furthermore, negative examples for this question are only found in the last ten retrieved results.

Furthermore, we have used t-distributed Stochastic Neighbor Embedding (t-SNE) to visualize the images in datasets based on their descriptors and examine similarities produced by the network. t-SNE allows to reduce descriptor dimensionality to 2 and plot the result in a 2D space. The database images of both *ROxford5k* and *RParis6k* are depicted in Figs. 5.2 and 5.3. We can observe that even when there is a noticeable shift in perspective,

scale, or lighting conditions or occlusions, pictures of the same landmark are clearly grouped. Likewise, we observe that the *RParis6k* dataset seems to be more distinctly divided into clusters, and therefore, we expect the network to have a higher retrieval efficiency on this dataset compared to the *ROxford5k* dataset. In further experiments (Section 5.7), this prediction was proven to be correct, with mAP on Paris landmarks being on average 11% higher than that on Oxford landmarks for medium evaluation difficulty.

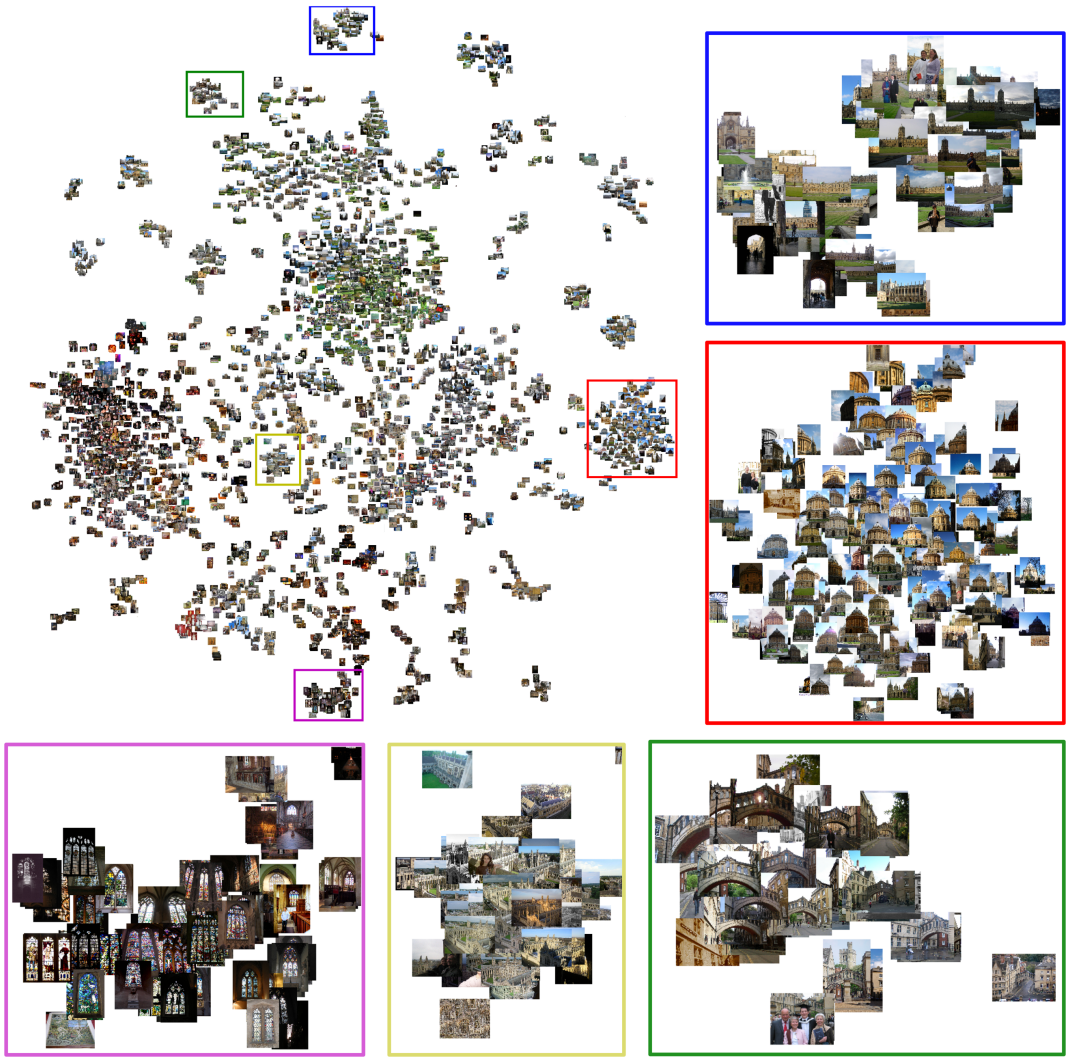

Figure 5.2: Visualization of the *ROxford5k* dataset with t-SNE and several zoomed in selections of clusters.

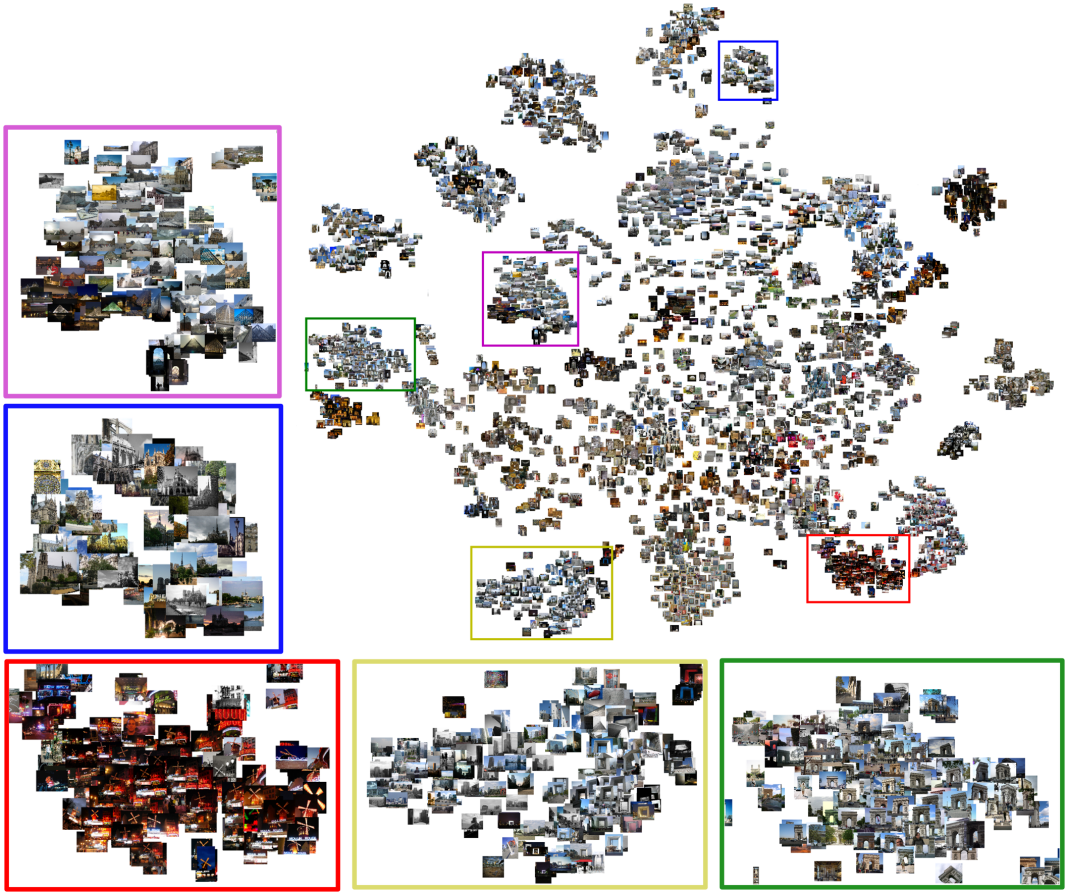

Figure 5.3: Visualization of the *RParis6k* dataset with t-SNE and several zoomed in selections of clusters.

5.5 Convolutional Backbone Architecture

For the convolutional backbone of the network, we evaluate several popular classification architectures: VGG16, VGG19, ResNet50 and ResNet101. Initially, we begin fine-tuning from publicly available models pre-trained on *ImageNet* data in all of the cases. Then, fully connected whitening layer is initialized with pre-computed whitening weights, as discussed in Section 4.5.2. All subsequent fine-tuning is performed on the *SfM120k* Landmarks dataset. Given that ResNet101 considerably outperforms VGG16 and VGG19 for all the evaluation scenarios (Table 5.1), we only experiment with ResNet101 in further sections since we assume that its intricate network design allows learning a more invariant representation of the images.

Architecture	ROxford (E, M, H)			RParis (E, M, H)		
ResNet50 GeM	81.23	63.51	38.23	88.92	75.12	52.98
ResNet101 GeM	**84.50**	**65.86**	**39.76**	**91.64**	**76.26**	**54.33**
VGG16 GeM	77.38	60.22	33.62	84.27	68.94	44.27
VGG19 GeM	78.94	62.17	33.89	85.63	69.51	43.98

Table 5.1: Single-scale mAP of the end-to-end learned networks with whitening for several different convolutional backbone architectures with GeM aggregating layer. Networks are evaluated on *ROxford5k* and *RParis6k* benchmarks under Easy (E), Medium (M) and Hard (H) setups.

5.6 Off-the-Shelf and Fine-Tuned Network Comparison

To demonstrate the benefits of fine-tuning, we performed a comparison of the off-the-shelf network pre-trained on ImageNet with the fine-tuned network. The first significant accuracy improvement, as compared to the off-the-shelf network, was the introduction of descriptor post-processing, which boosted the network performance by a substantial margin (13% for *ROxford5k* and 8.5% for *RParis6k* on a Medium setup), as indicated by Table 5.2. The ability to fine-tune the network for a particular domain increases network performance even further (by 24.5% on *ROxford5k* and 12.2% on *RParis6k* for a Medium setup compared to the naive implementation without whitening).

	ROxford5k (E, M, H)			RParis6k (E, M, H)		
Off-the-shelf network, no whitening	57.02	41.36	15.53	83.21	64.07	36.04
Off-the-shelf network with whitening as a post-processing step	73.30	54.38	25.77	88.99	72.56	49.62
Fine-tuned network with whitening learned end-to-end	**84.50**	**65.86**	**39.76**	**91.64**	**76.26**	**54.33**

Table 5.2: Comparison of the off-the-shelf network with the fine-tuned networks. The demonstrated networks, sharing ResNet101 backbone architecture with GeM pooling, were evaluated on *ROxford5k* and *RPais6k* benchmarks under Easy (E), Medium (M), and Hard (H) setups.

5.7 Final Re-Implementation Results

We further compared the original paper [RTC18] implementation with its TensorFlow re-implementation to ensure that similar results are reproduced. From Table 5.3, we can notice that the original implementation demonstrates slightly better performance on the *RParis6k* dataset while showing lower mAP results on the *RParis5k* benchmark. Both network implementations are trained with three different training data randomization seeds to assert consistency throughout the experiments. The results indicate that the seed has a minimal effect on the network performances and the average standard deviation for the experiments is only around 0.03%. Overall, we declared that both implementations achieve comparable performance.

Implementation	ROxford5k (E, M, H)			RParis6k (E, M, H)		
Original impl. seed=0	84.11	65.69	39.48	91.86	76.45	54.69
Original impl. seed=1	84.15	65.64	39.52	91.80	76.39	54.60
Original impl. seed=2	84.09	65.58	39.43	91.80	76.38	54.62
Original impl. mean	**84.12**	**65.64**	**39.48**	**91.82**	**76.41**	**54.64**
Original impl. std	0.025	0.045	0.037	0.028	0.031	0.039
TF re-impl. seed=0	84.55	65.86	39.74	91.67	76.20	54.31
TF re-impl. seed=1	84.49	65.81	39.72	91.61	76.31	54.39
TF re-impl. seed=2	84.47	65.91	39.83	91.64	76.27	54.30
TF re-impl. mean	**84.50**	**65.86**	**39.76**	**91.64**	**76.26**	**54.33**
TF re-impl. std	0.034	0.041	0.048	0.024	0.045	0.040
Mean difference	-0.39	-0.22	-0.29	0.18	0.15	0.30

Table 5.3: Comparison of the original implementation accuracy with the reproduced results. Networks for both implementations are trained with different seeds. In both cases, we used ResNet101 with the maximum training image size set to 1024, the number of negative images per training tuple equal to 4, and a batch of 5 training tuples. The mAP is evaluated on *ROxford5k* and *RPais6k* benchmarks under Easy (E), Medium (M), and Hard (H) setups.

5.8 Single- and Multi-Scale Evaluation

To incorporate information from various scales, we consider extracting and integrating features from images that have been resized to different resolutions. The aim is to enhance object matching and retrieval of small objects and database images at different scales. The network has a useful property in that it produces descriptors of the same length regardless of the input image's size. On the other hand, two different resolutions of the same image will

most likely result in different output descriptors. The network's first part is entirely convolutional, allowing it to process inputs of various sizes directly. At the same time, the pooling layer incorporates the size-dependent number of input features into a fixed-length representation. Following this logic, we extract descriptors from the image that has been resized at various scales and then merge them into a single final representation by sum-aggregation and subsequent $\mathcal{L}2$-normalization.

In the proposed setup, we use three sizes, with the larger side of the image scaled to $[1, \sqrt{2}, 1/\sqrt{2}]$ of the images with 1024 full pixel resolution while maintaining the original aspect ratio. The multi-scale evaluation method increases the computational time during feature extraction. Using three distinct picture scales in the evaluation phase takes about three times as long as single-scale evaluation. On the other hand, the cost of searching and storing is persistent. Table 5.3. presents single- and muli-scale mAP evaluation results for the identical networks. According to the findings, multi-scale mAP estimation increases network efficiency by 2% on average.

Setup	**ROxford5k (E, M, H)**			**RParis6k (E, M, H)**		
Single-scale mAP	84.50	65.86	39.76	91.64	76.26	54.33
Multi-scale mAP	86.83	67.89	41.83	92.31	77.61	56.29

Table 5.4: Single- and multi-scale (with scales $[1, \sqrt{2}, 1/\sqrt{2}]$) mAP results for the default network on *ROxford5k* and *RPais6k* benchmarks under Easy (E), Medium (M), and Hard (H) setups.

5.9 Randomly Initialized and Pre-Computed Whitening Comparison

As demonstrated in Table 5.2, whitening has a major impact on the network performance. Furthermore, we discovered that the network convergence is faster with the whitening layer, requiring less computational resources. However, the issue of the whitening layer's initialization arises. Essentially, we test two options: random FC weight initialization and discriminative pre-computing of the weights, as discussed in Section 4.5.2. We compared identical network training processes (one with random weight initialization and the other with pre-computed whitening) by plotting the evolution of mAP on the testing data as a function of the number of training epochs (Fig. 5.4). The results clearly showcase that using pre-initialized whitening improves the outcomes by a margin of 4% on both *ROxford5k* and *RParis6k* benchmarks under a Medium evaluation setup. Moreover, we observe that learning the fully connected whitening layer end-to-end after initializing it with random weights is inefficient due to the hindered convergence rate.

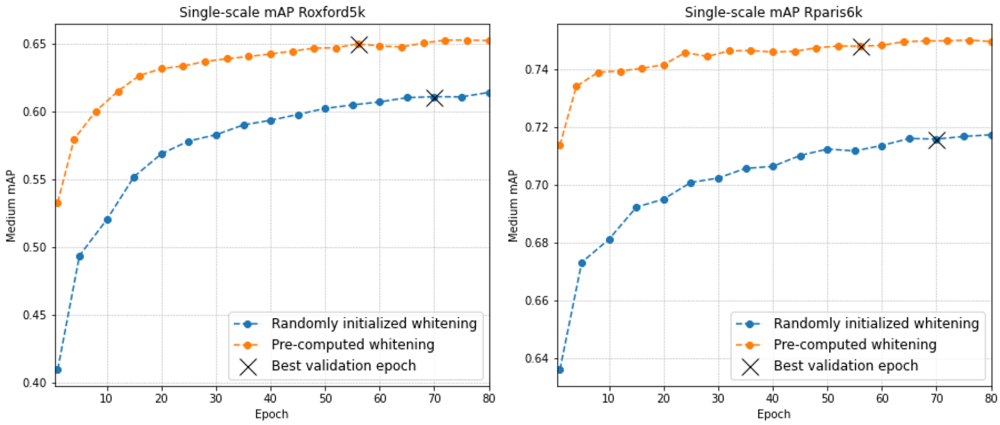

Figure 5.4: The plots exhibit the evolution of single-scale mAP as a function of the number of training epochs for a Medium evaluation protocol on *ROxford5k* and *RParis6k* benchmarks. The initial epoch 1 corresponds to the network after the first training epoch. The presented plots correspond to two identically initialized networks (ResNet101 with GeM pooling and FC whitening layer, with the maximum training image size set to 800 pixels), one with the pre-computed and the other with randomly initialized FC layer.

5.10 Global Pooling Layer Comparison

In Figure 5.5, we present the evaluation results of GeM (with a shared power $p = 3$), MAC, and SPoC descriptors on *ROxford5k* and *RParis6k* benchmarks in the form of training progress. In our experiments, GeM pooling appears consistently superior to the other mentioned pooling types. Furthermore, though SpoC pooling has the slowest convergence rate, it generally outperforms MAC, which, under our setup, turns out to be the weakest descriptor in terms of performance. Lastly, both validation and training losses for all of the pooling types gradually decrease with similar trends (Fig. 5.6).

5.11 GeM Pooling Properties

As mentioned in section 4.3, GeM pooling is a generalization of the other pooling types, such as MAC and SPoC poolings, for certain values of the power p. There is a different pooling parameter per feature map as described in section 4.3, but it is also possible to use a shared one. In this case, $p_i = p, \forall i \in [1, K]$ and we simply denote it by p. In Fig. 5.7, we compared the

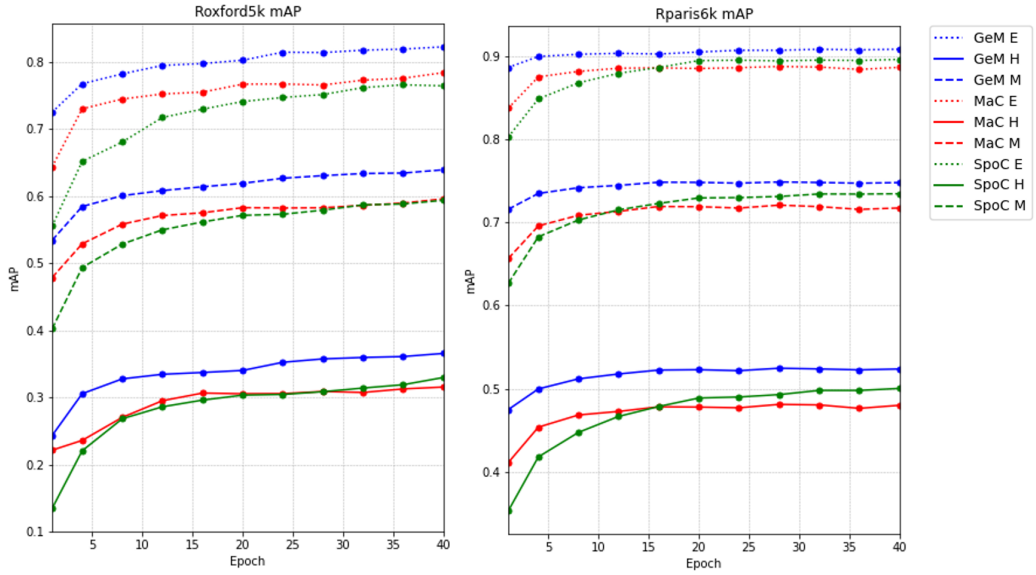

Figure 5.5: MAC, SPoC, and GeM (with power $p = 3$) pooling layer performance comparison with the same network architecture. The plots display the evolution of single-scale mAP as a function of the number of training epochs for Easy, Medium, and Hard evaluation setup on *ROxford5k* and *RParis6k* benchmarks. The experiment is performed with the ResNet101 architecture with whitening and one of the aforementioned pooling layers.

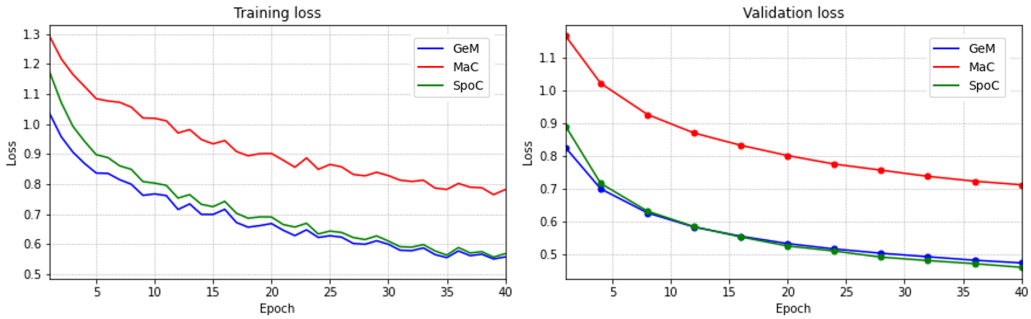

Figure 5.6: Losses for training with MAC, SPoC, and GeM pooling layers (with power $p = 3$) for the same network architecture.

similarity of the output matrices after MAC, SPoC, and GeM poolings for different values of shared p. As suggested by the definition of GeM pooling, for $p = 1$, we obtained results identical to SPoC, and with the increasing p, the result becomes more similar to the one after MAC pooling.

Further, instead of down sampling the entire feature map to a single value, we experimented with local GeM pooling to down sample patches of the

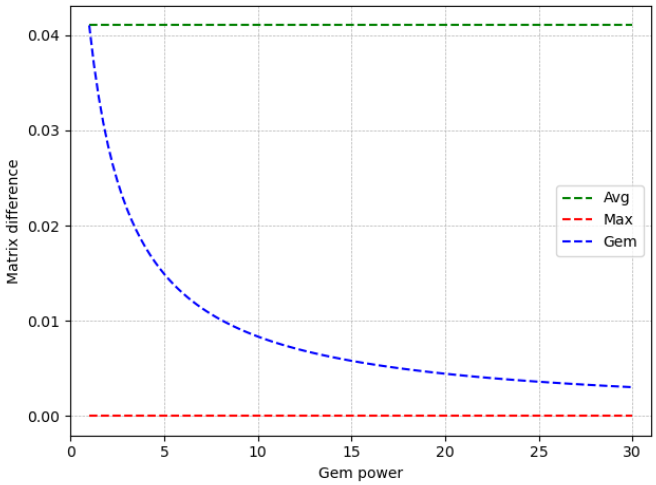

Figure 5.7: Plot of the similarity of the MAC, SPoC, and GeM poolings based on the value of the GeM power parameter p.

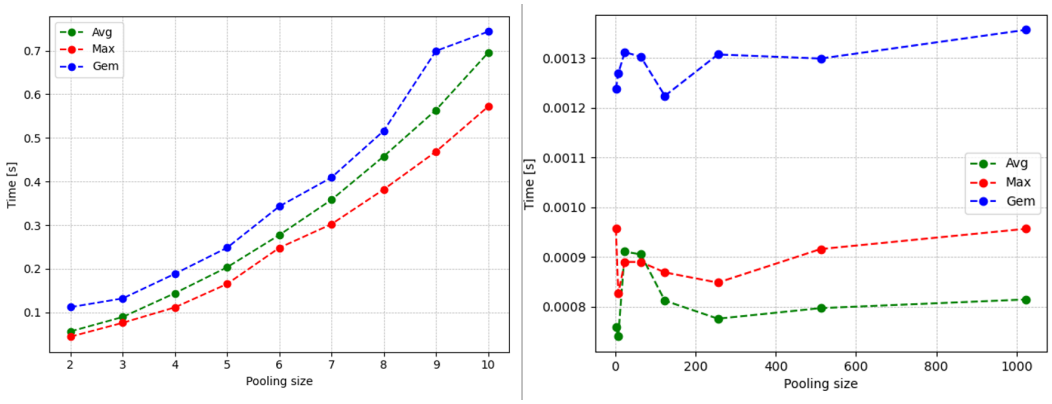

Figure 5.8: CPU and GPU speed comparison for Average, Max, and GeM pooling. For CPU setup (on the left), we used kernel sizes from 2 to 10. GPU (on the right) uses kernel sizes 2, 8, 16, 32, 64, 128, 256, 512, 1024.

input feature map, in a similar manner as pooling layers in convolutional networks. We hypothesized that replacing Max pooling layers with local generalized pooling may yield improvements in both speed and network accuracy in standard classification architectures (e.g., VGG16 or VGG19). The argument for speed enhancement is based on the assumption of faster arithmetical computations on the GPU compared to maximum element searching. Additionally, we reasoned that adding another learnable parameter (GeM power p) would result in improved model accuracy.

To validate our hypothesis, we benchmarked the runtime of Max, Average, and GeM pooling layers on 1024 x 1024 randomly initialized feature maps with both CPU and GPU setups (Fig. 5.8). Contrary to our expectations, in both experiments, GeM was the slowest performance pooling layer. A

possible explanation of this inconsistency might be the optimization of the bult-in Keras Max and Average pooling layers implementations or an effective multi-core CUDA parallelization.

5.12 Contrastive and Triplet Loss Comparison

In addition to the previous experiments, we performed a comparison of the effect of contrastive and triplet loss on the networks with identical hyperparameters, whose results are presented in Fig. 5.9. Unlike in [RTC18], the results display that the contrastive loss appears to be inferior under the experimental setup. Moreover, we observe slight oscillations of the validation loss for contrastive loss, which might imply overfitting.

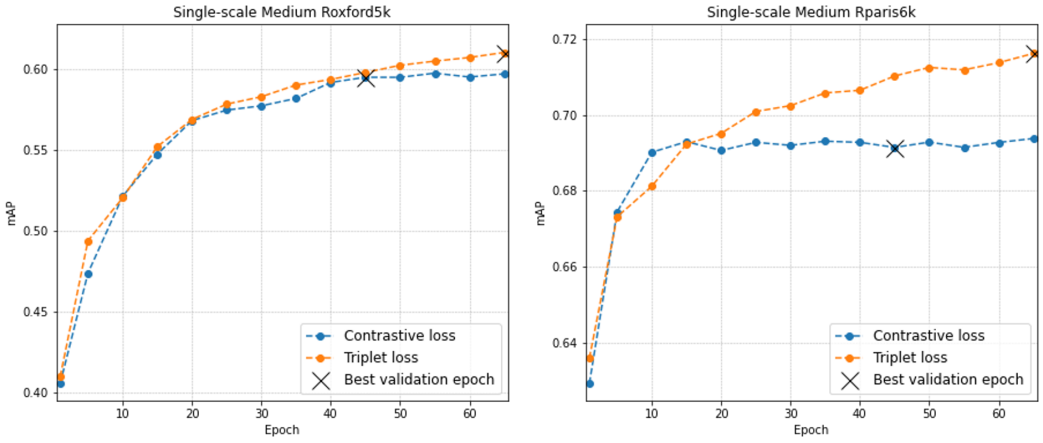

Figure 5.9: The plots show the evolution of single-scale mAP (for Medium evaluation protocol) as a function of the number of training epochs for two identical networks trained with contrastive and triplet losses.

5.13 *SfM120k* Cluster Re-Occurrence

One of the issues with the *SfM120k* dataset used for network training is that some of the initial image clusters were split up into several separate subclusters during the automatic creation phase of the dataset. As a result, we can encounter the same object of interest in various clusters and assign those images as false negatives for query images, which would be an invalid negative selection. Since hard negative mining is performed during network

training, those images would likely be chosen as negative examples because they would generate the highest loss. Indeed, when examining the selected training tuples, we discover a large number of false-negative examples, such as the one depicted in Figure 5.10. Since we only allow one image per cluster to be chosen for the training tuple, this example showcases that there must be at least three clusters in the dataset depicting the Sydney Opera House.

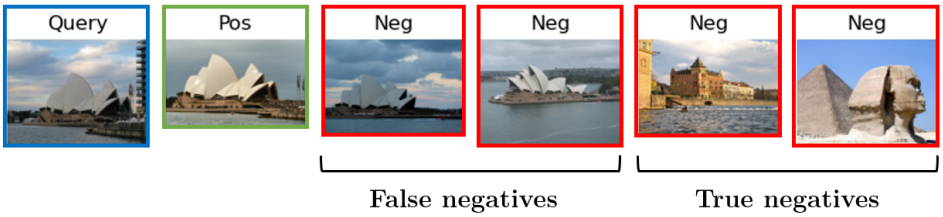

Figure 5.10: Example of the false-negative training images choice selected as a result of multiple clusters representing the same landmark. Two of the negative images correspond to the same landmark as the one depicted in the query image.

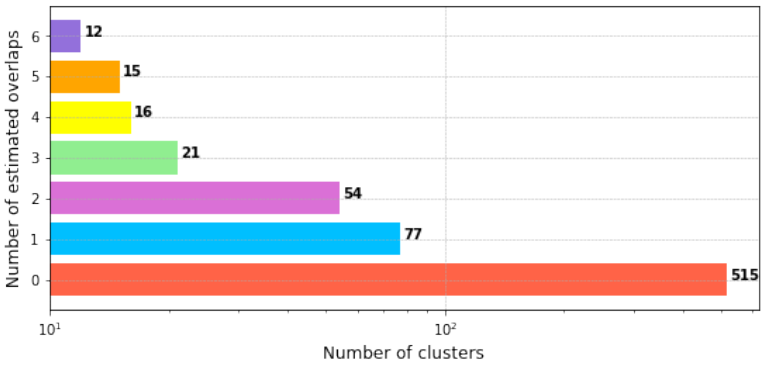

Figure 5.11: Bar graph of the possible number of similarities per cluster as suggested by the trained network.

To confirm our observations, we conducted the cluster similarity analysis of the *SfM120k* dataset. This was achieved by extracting the descriptor vectors for every image in the dataset with a fine-tuned network. Then, we used *FAISS* [JDJ17] - a Facebook research library for efficient similarity search and dense vector clustering on the GPU, to find the most similar descriptors for each image among different clusters and their similarity scores. By calculating the cumulative cluster similarity scores and setting the similarity threshold, we can determine how likely the clusters are to contain the same objects of interest. According to the analysis estimates, out of 712 total clusters, around 200 clusters contain instances of similar landmarks (Fig. 5.11).

Determining whether the found similar clusters indeed include the same objects of interest is not a trivial task. For example, Fig. 5.12 depicts example images from the overlapping clusters successfully found during the experiment. On the contrary, Fig. 5.13 demonstrates a wrong cluster similarity estimation.

Even though one of the discovered clusters is visually analogous to the query cluster, it does not depict the same object of interest.

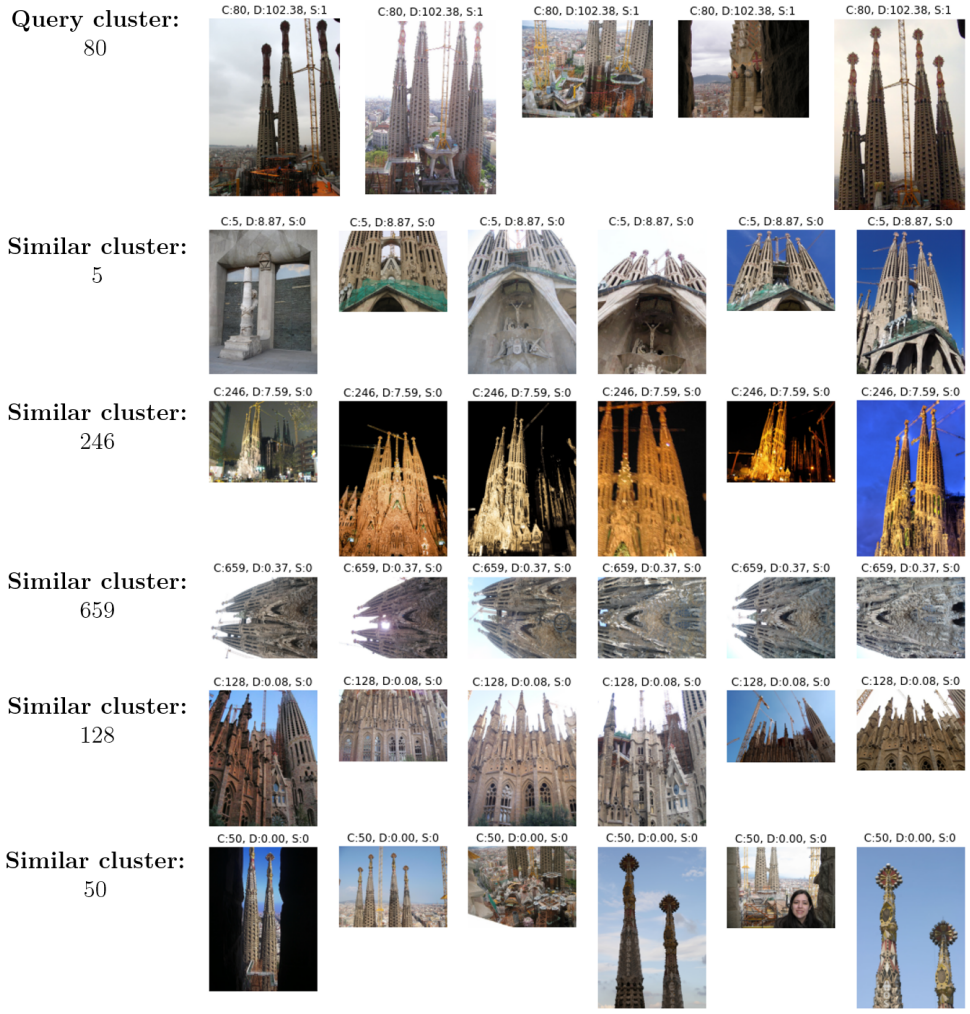

Figure 5.12: The most similar clusters found by the network for the query cluster 80 (Sagrada Familia, Barcelona). Each row consists of the example images from the corresponding cluster (on the left). All of the found similar clusters correspond to the same landmark.

5.14 The Impact of False Negatives

To assess the extent to which false-negative selection affects the training, we investigated deliberate false-negative introduction during the training process. In this experiment, false-negatives are represented by randomly selected images from the query cluster. However, due to the high intra-class variability, the chosen random samples are not guaranteed to depict the same

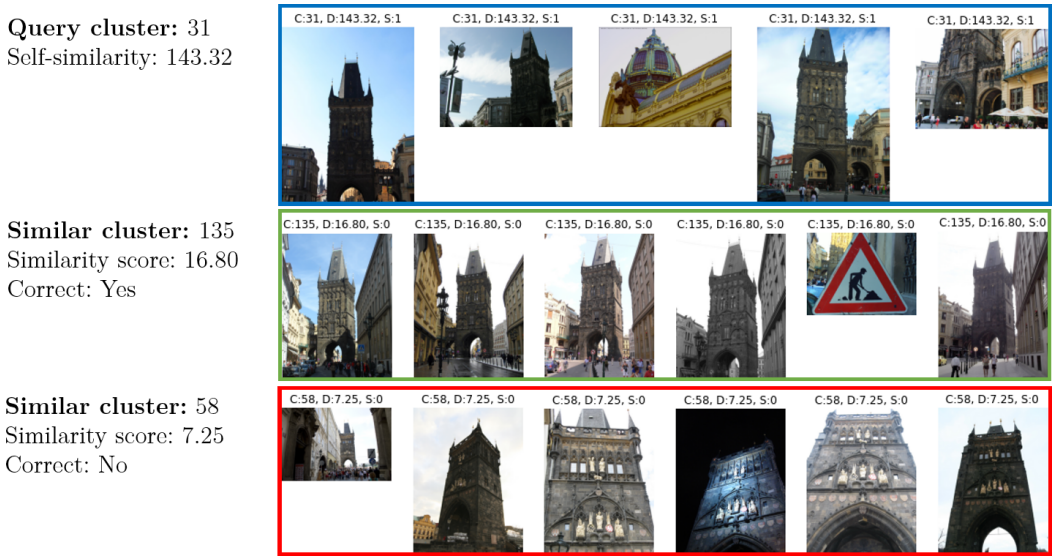

Figure 5.13: Similarities found by the network for cluster 31 (Powder Tower, Prague). Even though the pictures in cluster 58 are very similar to the query cluster, they do not correspond to the same landmark.

landmark and are, therefore, more prone to being less hard than the actual false-negatives. Plots in Fig. 5.14 showcase mAP as a function of the number of training epochs for identical networks with 0%, 10%, 25%, and 50% chances of infusing new false-negatives during the training tuple selection. From there, we see that the network is very robust to the inconsistencies in the training data since even for a 50% chance of false-negative pick for every negative in the training tuple of size 7 (1 query, 1 positive, 5 negatives), the performance downgrades only by around 1% on both benchmarks.

Based on the obtained results, we concluded that the network is resistant enough to deal with the mistakes in the training data. On the other hand, injecting errors in tuples with a smaller number of negatives or choosing hard false negatives is likely to downgrade the network performance even further than in the previous experiment, highlighting the need to clean the training dataset. These experiments might be the scope of future works.

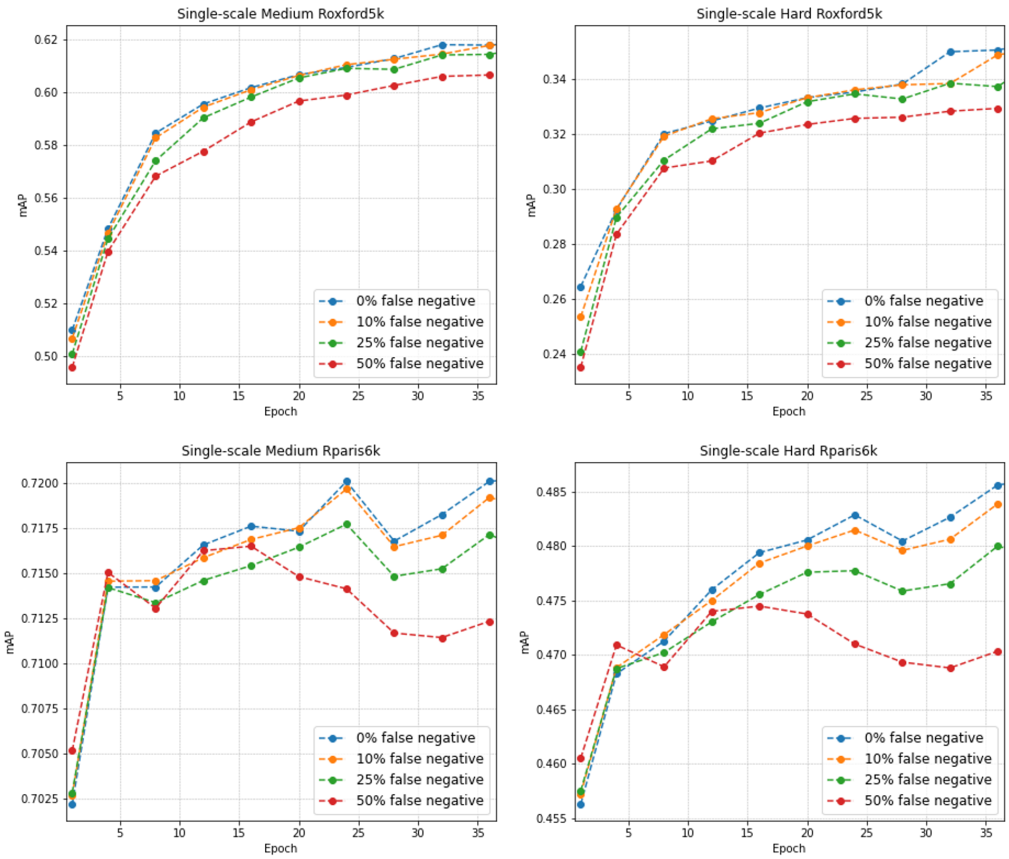

Figure 5.14: Plot of the effect of false-negative selection on the network accuracy. ResNet101 with GeM pooling and whiten layer, with the maximum training image size set to 362 pixels, and 5 negative images within the training tuple, was used in all of the experiments.

Chapter 6

Conclusions

In this thesis, we tackled the problem of instance-level image retrieval with the use of CNNs and re-implemented a state-of-the-art image retrieval framework based on the work of Radenović *et al.* [RTC18] in TensorFlow 2.

We fine-tuned CNNs for image retrieval from an extensive collection of landmark images and replicated the original implementation results. On top of that, we conducted a number of experiments discussing both the training dataset's issues as well as learning with different hyperparameters and strategies. The final project code was successfully integrated into the official TensorFlow repository, as a part of the DELF [Tena] library. This implementation can be further used as a benchmark for future research in the area of CBIR. Moreover, it is to be provided as one of the baseline solutions for the upcoming annual Kaggle Landmark Retrieval Challenge organized by Google.

Appendix A

Bibliography

[AGT+16] Relja Arandjelović, Petr Gronat, Akihiko Torii, Tomas Pajdla, and Josef Sivic, *Netvlad: Cnn architecture for weakly supervised place recognition*, 2016.

[BETVG08] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool, *Speeded-up robust features (surf)*, Computer vision and image understanding **110** (2008), no. 3, 346–359.

[BL15] Artem Babenko and Victor Lempitsky, *Aggregating deep convolutional features for image retrieval*, 2015.

[CHL05] Sumit Chopra, Raia Hadsell, and Yann LeCun, *Learning a similarity metric discriminatively, with application to face verification*, 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), vol. 1, IEEE, 2005, pp. 539–546.

[CPS+07] Ondrej Chum, James Philbin, Josef Sivic, Michael Isard, and Andrew Zisserman, *Total recall: Automatic query expansion with a generative feature model for object retrieval*, 2007 IEEE 11th International Conference on Computer Vision, IEEE, 2007, pp. 1–8.

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei, *Imagenet: A large-scale hierarchical image database*, 2009 IEEE conference on computer vision and pattern recognition, Ieee, 2009, pp. 248–255.

[GNPS18] Ido Guy, Alexander Nus, Dan Pelleg, and Idan Szpektor, *Care to share? learning to rank personal photos for public sharing*,

Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, 2018, pp. 207–215.

[Goo] Google, *Google image search*, online, [Accessed: 2021-04-12] `https://images.google.com/`.

[HKM+97] Jing Huang, S Ravi Kumar, Mandar Mitra, Wei-Jing Zhu, and Ramin Zabih, *Image indexing using color correlograms*, Proceedings of IEEE computer society conference on Computer Vision and Pattern Recognition, IEEE, 1997, pp. 762–768.

[HTS+06] Alaa Halawani, Alexandra Teynor, Lokesh Setia, Gerd Brunner, and Hans Burkhardt, *Fundamentals and applications of image retrieval: An overview.*, Datenbank-Spektrum **18** (2006), no. 14-23, 6.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, *Deep residual learning for image recognition*, Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[Işı14] Şahin Işık, *A comparative evaluation of well-known feature detectors and descriptors*, International Journal of Applied Mathematics Electronics and Computers **3** (2014), no. 1, 1–6.

[JDJ17] Jeff Johnson, Matthijs Douze, and Hervé Jégou, *Billion-scale similarity search with gpus*, arXiv preprint arXiv:1702.08734 (2017).

[JDSP10] Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez, *Aggregating local descriptors into a compact image representation*, 2010 IEEE computer society conference on computer vision and pattern recognition, IEEE, 2010, pp. 3304–3311.

[Kag] Kaggle, *Google landmark retrieval 2020*, online, [Accessed: 2021-04-12] `https://www.kaggle.com/c/landmark-retrieval-2020/overview`.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, *Imagenet classification with deep convolutional neural networks*, Advances in neural information processing systems **25** (2012), 1097–1105.

[KZS15] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov, *Siamese neural networks for one-shot image recognition*, ICML deep learning workshop, vol. 2, Lille, 2015.

[LLQ+16] Ziwei Liu, Ping Luo, Shi Qiu, Xiaogang Wang, and Xiaoou Tang, *Deepfashion: Powering robust clothes recognition and retrieval with rich annotations*, Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 1096–1104.

[Low99] David G Lowe, *Object recognition from local scale-invariant features*, Proceedings of the seventh IEEE international conference on computer vision, vol. 2, Ieee, 1999, pp. 1150–1157.

[Mic] Microsoft, *Bing visual search*, online, [Accessed: 2021-04-12] `https://www.bing.com/visualsearch`.

[NAS+17] Hyeonwoo Noh, Andre Araujo, Jack Sim, Tobias Weyand, and Bohyung Han, *Large-scale image retrieval with attentive deep local features*, Proceedings of the IEEE international conference on computer vision, 2017, pp. 3456–3465.

[OSXJS16] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese, *Deep metric learning via lifted structured feature embedding*, Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 4004–4012.

[PCI+07] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman, *Object retrieval with large vocabularies and fast spatial matching*, 2007 IEEE conference on computer vision and pattern recognition, IEEE, 2007, pp. 1–8.

[Pin] Pinterest, *Pinterest*, online, [Accessed: 2021-04-12] `https://www.pinterest.com/`.

[PLSP10] Florent Perronnin, Yan Liu, Jorge Sánchez, and Hervé Poirier, *Large-scale image retrieval with compressed fisher vectors*, 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, IEEE, 2010, pp. 3384–3391.

[QAAM17] Adnan Qayyum, Syed Muhammad Anwar, Muhammad Awais, and Muhammad Majid, *Medical image retrieval using deep convolutional neural network*, Neurocomputing **266** (2017), 8–20.

[Rad19] Filip Radenovic, *Visual retrieval with compact image representations*, Ph.D. thesis, Visual Recognition Group, Czech Technical University in Prague, 2019.

[RARS19] Jerome Revaud, Jon Almazán, Rafael S Rezende, and Cesar Roberto de Souza, *Learning with average precision: Training image retrieval with a listwise loss*, Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 5107–5116.

[RIT+18] Filip Radenović, Ahmet Iscen, Giorgos Tolias, Yannis Avrithis, and Ondřej Chum, *Revisiting oxford and paris: Large-scale image retrieval benchmarking*, CVPR, 2018.

[RTC16] F. Radenović, G. Tolias, and O. Chum, *CNN image retrieval learns from BoW: Unsupervised fine-tuning with hard examples.*

[RTC18] Filip Radenović, Giorgos Tolias, and Ondřej Chum, *Fine-tuning cnn image retrieval with no human annotation*, IEEE transactions on pattern analysis and machine intelligence **41** (2018), no. 7, 1655–1668.

[SDMP17] O. Seddati, S. Dupont, S. Mahmoudi, and M. Parian, *Towards good practices for image retrieval based on cnn features*, 2017 IEEE International Conference on Computer Vision Workshops (ICCVW), 2017, pp. 1246–1255.

[SI07] Christian Siagian and Laurent Itti, *Rapid biologically-inspired scene classification using features shared with visual attention*, IEEE transactions on pattern analysis and machine intelligence **29** (2007), no. 2, 300–312.

[SRCF15] Johannes L Schonberger, Filip Radenovic, Ondrej Chum, and Jan-Michael Frahm, *From single image query to detailed 3d reconstruction*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 5126–5134.

[SZ03] Josef Sivic and Andrew Zisserman, *Video google: A text retrieval approach to object matching in videos*, Computer Vision, IEEE International Conference on, vol. 3, IEEE Computer Society, 2003, pp. 1470–1470.

[SZ14] Karen Simonyan and Andrew Zisserman, *Very deep convolutional networks for large-scale image recognition*, arXiv preprint arXiv:1409.1556 (2014).

[Tena] TensorFlow, *Delf repository*, online, [Accessed: 2021-04-12] `https://github.com/tensorflow/models/tree/master/research/delf`.

[Tenb] Tensorflow, *Tensorflow model garden*, online, [Accessed: 2021-04-12] `https://github.com/tensorflow/models/`.

[TL19] Mingxing Tan and Quoc Le, *Efficientnet: Rethinking model scaling for convolutional neural networks*, International Conference on Machine Learning, PMLR, 2019, pp. 6105–6114.

[TSJ16] Giorgos Tolias, Ronan Sicre, and Hervé Jégou, *Particular object retrieval with integral max-pooling of cnn activations*, 2016.

[Wik] Wikipedia, *List of content-based image retrieval engines*, online, [Accessed: 2021-04-17] `https://en.wikipedia.org/wiki/List_of_CBIR_engines`.

[Yan] Yandex, *Yandex image search*, online, [Accessed: 2021-04-12] `https://yandex.com/images/`.