



POLITECNICO DI TORINO

Master degree course in Data Science

Master Degree Thesis

Efficient and scalable visual place recognition

Supervisors

Prof.ssa Barbara Caputo
Dr. - Ing. Carlo Masone

Candidate

Emanuele MUNAFO'
matricola: s265060

ACCADEMIC YEAR 2020-2021

This work is subject to the Creative Commons Licence

Abstract

Visual Place Recognition (VPR) is the task of recognizing what place is represented in a given image. This task, which is being studied for decades, is rich of challenges and margins for improvement.

The main purpose of this thesis has been to study techniques to improve the scalability of VPR algorithms without penalizing their accuracy. This work contains many experiments that show which improvements can be achieved by applying similar architectures and what their limitations are. In this regard, by trying to overcome some of these limitations, we also experimented with different Domain Adaptation techniques and reported their results.

Moreover, a software made up by APIs back-end calls and a user-friendly front-end, was developed to potentially allow every user to perform searches on country scale in just a matter of seconds. The software also offers features like user roles, authentication, high performance and flexibility by design. The Flask API allows a more scalable deployment. Furthermore, we described how a VPR pipeline can be designed and deployed in a large scale environment by showing all the steps required for the implementation of a similar system.

Acknowledgements

Special thanks to my family for their continuous and invaluable support. A dutiful thanks to the Vandal team and the IIT foundation for the help and resources provided, without which it would have been impossible to carry out this work.

Contents

List of Tables	8
List of Figures	9
I First Chapter	13
1 Background	15
1.1 Problem Statement	15
1.2 High Level Pipeline	17
1.3 Challenges	19
1.3.1 Sources of geotagged images	19
1.3.2 Temporal sparseness	20
1.3.3 Spatial sparseness	21
1.3.4 Large Scale and Scalability	22
1.3.5 Domain Adaptation	24
2 Related work	27
2.1 Introduction	27
2.2 Evolution of SotA descriptors	30
2.2.1 BoW	30
2.2.2 VLAD	31
2.2.3 NetVLAD	32
2.2.4 GeM	34
2.3 Other approaches	34
3 Data and evaluation	37
3.1 Dataset	37
3.1.1 Introduction	37
3.1.2 Research dataset	37

3.1.3	Development dataset	39
3.1.4	Domain Adaptation dataset	40
3.2	Evaluation	42
3.2.1	Recall	42
3.2.2	Model selection	43
4	Architectures and benchmarks	45
4.1	Architectures	45
4.1.1	Overview	45
4.1.2	Mining	46
4.1.3	Backbones	47
4.1.4	Pooling layer	50
4.1.5	Loss	51
4.1.6	Domain Adaptation	52
4.1.7	Finetuning	56
4.2	Results and discussion	57
4.2.1	Overview	57
4.2.2	ResNet based architectures	58
4.2.3	EfficientNet based architectures	65
4.2.4	Classification approach	66
4.2.5	Domain Adaptation	68
5	Software	75
5.1	Overview	76
5.2	Requirements	78
5.3	Design and features	79
5.3.1	Efficiency	81
5.4	Results and performance	81
5.5	Improvements	85
6	Conclusion	87
7	Appendix	89
7.1	Appendix A: Development Gallery	89
7.2	Appendix B: API Example	90
7.3	Appendix C: GUI results page	93
7.4	Appendix D: Results example	94
	Bibliography	95

List of Tables

4.1	ArchSummaryComb	57
4.2	Results of ResNet18 architecture on Pitts30k (L2 norm + whitening)	59
4.3	Top results with ResNet18 on Pitts30k	59
4.4	Top results with ResNet18 on SVOX	60
4.5	ResNet18 architecture on Pitts30k (without L2 normalization)	60
4.6	Top results with ResNet50 on Pitts30k	63
4.7	Top results with ResNet50 on SVOX	63
4.8	Comparison of cache sizes over different aggregation layers	64
4.9	Top results with Eff-B0 on Pitts	66
4.10	Top results with Eff-B0 on SVOX	66
4.11	Top results with classification	68
4.12	EfficientNet accuracy with and without Data Augmentation	68
4.13	DA Baseline	69
4.14	FDA based DA	70
4.15	FastCUT based DA	71
4.16	RevGrad	72

List of Figures

1.1	Query and gallery example	16
1.2	Overall steps for retrieval based architectures.	17
1.3	Sparseness examples	23
1.4	Two samples of RobotCar in different lighting condition	25
2.1	Evolution of VPR SotA techniques over the time.	27
2.2	Local and global features	29
2.3	Representation of the BoVW approach.	30
2.4	NetVLAD architecture	33
2.5	GeM architecture	34
2.6	GeM compared with max and average pooling.	35
2.7	PlaNet: a classification approach.	35
3.1	SVOX composition showed on map.	38
3.2	FDA diagram[29].	41
3.3	Comparison between precision and recall.	43
4.1	High Level Component for a generic VPR architecture.	45
4.2	Residual and standard blocks compared.	48
4.3	ResNet architecture.	50
4.4	EfficientNet architecture	50
4.5	CMAP based architecture.	51
4.6	A generic RevGrad based architecture [33].	55
4.7	Comparison of different aggregation layers with a ResNet50 based architecture on Pitts30k (left) and SVOX (right).	62
4.8	On the left a picture from Oxford RobotCar taken at daytime. On the right the same picture shifted to night domain thanks to FastCUT.	69
4.9	Comparison of results for domain adaptation on target set (night condition).	73
5.1	Time requirements for each step in our pipeline.	75
5.2	A screenshot of the GUI showing the authentication feature.	80
5.3	Home page of a logged in user.	82

5.4	The search page	83
7.1	Results page relative to a search with a query and <i>NumPredictions=10</i>	93
7.2	On the top two search results in Turin. On the bottom the results relative to a search in Palermo; At the bottom right corner a wrong prediction due to a big occlusion.	94

Part I

First Chapter

Chapter 1

Background

1.1 Problem Statement

One of the most popular tools of the modern age, known as camera, allows us to capture images of each spot on earth. Computer vision, a branch of the more generic Machine Learning, emerged to try to understand the images taken by cameras to extract and process such data. Another widespread creation of modern technology is the GPS, used to identify the geographical position in almost every place.

Even if they look dissimilar, these different devices have an hidden still strong relation represented by the combination of both visual and space information. In other words, it's always legit to ask: *Where was this picture taken?*

Geo-localization finds numerous applications like organizing photo collections and robotics. The geographical location where an image or video was taken is a key aspect of many different tasks like:

- **Geo-tagging photos** Association of a location for a given image often showed in a map form (i.e. old photos, holiday photos)
- **Visual information retrieval** The task used to extract visual information from an image (i.e. to search for similar photos, to obtain information about the environment)
- **Self-driving vehicle** In autonomous driving visual place recognition is used for loop closure detection, in SLAM [2] [3], but is also relevant for localization.



Figure 1.1. On the left the distinction between the gallery and the query; on the right an example of a VPR task.

- **AR** Augmented Reality has some really interesting applications like a step by step guide or providing additional information about the pointed object.
- **Interactive VR** In a similar way the Virtual Reality needs also the spatial information to work in an interactive way.

The research exposed in this work investigates how to automatically derive geo-information from a photo. In particular, it embraces the challenge of estimating the longitude and latitude location of an image based only on its visual content. The goal of this thesis was to develop a scalable VPR system for images and to study the possibilities for improvement in both accuracy and performance.

In this work, we start by providing a comprehensive evolution of the state of the art in large scale visual VPR and discuss the main trends in this area. The thesis makes also other different contributions. The first being the design and implementation of a scalable, reliable and flexible framework and the second being the optimization of specific components in both the network and the pipeline. Based on the results presented in this thesis, we propose some directions where additional studies can be conducted. Thus, this research involved three key concepts whose definitions are given as follow:

Image retrieval Task that aims to find similar images to a query image among an image dataset (sometimes named *gallery* or *database*).

Large Scale - Involving large numbers or a large area.
(Of a map or model) made to a scale large enough to show certain features in detail.

Visual Place Recognition : The task of matching a view of a place with a different view of the same place taken at a different time.

Although these concepts are different they also tend to overlap because *Visual Place Recognition* is often approached as an *Image Retrieval* task and we usually want it to work on a *Large Scale*.

1.2 High Level Pipeline

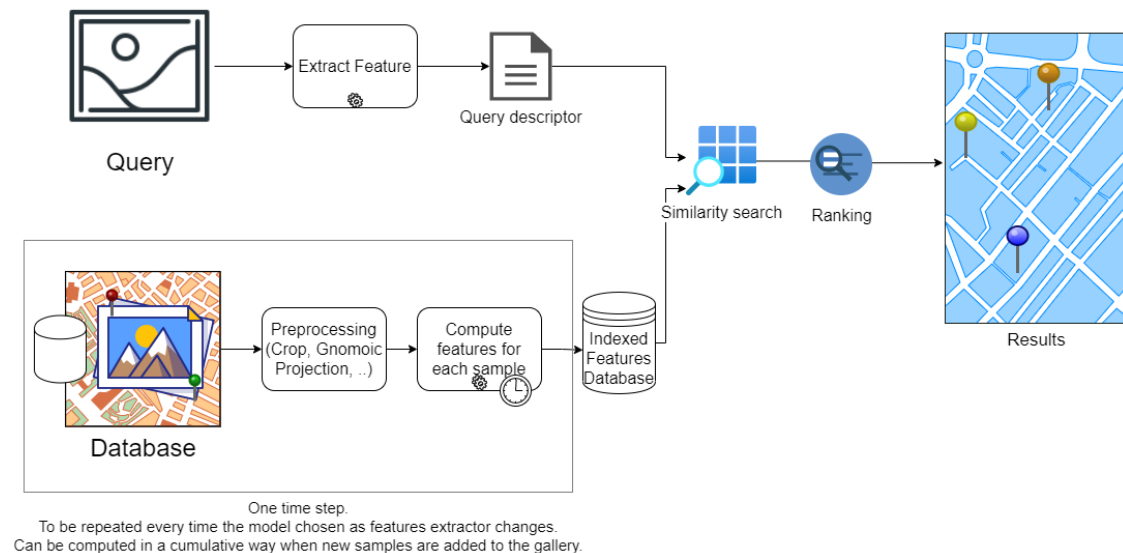


Figure 1.2. Overall steps for retrieval based architectures.

A VPR pipeline, independently from the specific architecture, can be thought as composed of the following steps:

- Represent the world by using a set of geo-tagged images
- Given a query image, find the best matching image according to some criteria
- Transfer the geo-tag of the best matching image to the query

The first point clarifies the need of a dataset of some sort. Ideally we would need a dataset containing at least one picture for every location to be recognized. The limitations and the difficulties involved will be clarified in the next section.

The other points highlight the need of two entities which have to be defined for the whole system to work. The first one is some sort of representation that should be used to encode the visual features into a compact and meaningful representation, procedure usually known as *feature extraction*. The second one is a method of finding the most similar images in the feature space.

By looking at the high level pipeline, shown in Figure 1.2, we can introduce some components of the system.

We can distinguish two kinds of image set: the *query* and the *gallery*.

The query and the gallery The images used to represent the world are commonly known as *gallery* or *database*. In particular, this can be thought as the set of images that are used to perform the search (hence the name). It should also be clear that these are the images from which the geo-tag is transferred to the query.

On the other side, the images for which the location should be discovered are called *queries*. For each query the system will estimate the location by searching the closest match into the gallery. The Figure 1.1 should clarify the distinctions between the two.

Preprocessing The gallery itself is just a bunch of images and metadata, like the geo-tag, and can't be used in its original form. In other words, we usually need to adopt some *preprocessing* techniques like *cropping*.

Feature extraction When the images are in a suitable format the next step is the extraction of features for each preprocessed image in the gallery. Both the preprocessing and the extraction of the features are performed only once in a while by saving the computed features to disk. Specific data structures and algorithms can be used to improve the searching process.

The query and the search process Let's go back to the query. When a query is given as input, at first we need to compute its descriptor, as previously done with all the images in the gallery. Then we need to perform a similarity search (i.e. k-NN) to compare the query image to all the images in the gallery to find the closest match. Since the comparison isn't made

between the *raw image representations* (RGB) but between their computed descriptors, we call it similarity search in the *feature space*.

The last step usually consists of an algorithm chosen to rank the different predictions returned from the similarity search.

1.3 Challenges

Although VPR has already been studied for decades, the challenge of estimating the image location from its visual content remains notable. This is particularly true when tackling VPR on a large scale both because of the great amount of image samples involved and because of the scalability and accuracy constraints to be respected. We can identify four main issues when dealing with VPR :

- Sources of geotagged images
- Temporal sparseness
- Spatial sparseness
- Large scale and scalability

The aforementioned challenges are particularly hard to solve in a large scale environment because even if we would like to have as many samples as possible, this would entail two more issues. The first being where to get, how to store and how to process a so big dataset and the second being an increased sparsity because of the bigger scale and dataset.

Moreover, as this subject has been studied by different communities, namely computer vision, robotics and machine learning, who often share their results in separate scientific outlets, there is an increased complexity for new researchers on the field [19].

The results reported in this thesis want to encourage the research community in pursuing this challenge.

1.3.1 Sources of geotagged images

The first step is to collect images for building the ground-truth database. Here comes the first issue: *What can be the source of images needed to compose the gallery?*

Such a dataset should be big enough to contain at least a sample per location to be recognized. At the same time, a dataset with more samples per

location is highly recommended because it improves the generalization capabilities of the network, thus allowing to easily recognize a place when the photo is taken under different lighting condition and with a different point of view with respect to the samples in the gallery.

Generally, a similar dataset can be obtained by two different sources:

- **Photo-community site (i.e. Flickr)** Similar collections have the advantage of being continuously and fast growing but the images aren't equally distributed (they concentrate on landmarks like the Eiffel Tower in Paris). Samples belonging to this category are also characterized by noisy tag and visual content (i.e. artistic photos).
- **Interactive Panoramas (i.e. StreetView)** Such collections usually cover almost every street (with limitations in some countries) and the position is more accurate. The downside is that these images are seldom updated because of the high resources and cost involved. Moreover, the samples provided in this dataset are usually available in a *pano format* (from which a perspective cutout can always be generated [16],[15]).

In this work all the used datasets fall in the second category and for the dataset built from StreetView a perspective cutout is generated. In particular, a Google Streetview panorama is in the form of an equirectangular projection. The panorama, in its original form, can't be used right away to compose the gallery, because all the samples are distorted and a retrieval NN can lead to poor accuracy when images come from different domains. A panorama also contains a lot of information that is neither unique nor relevant for the image, like the sky (on the top) and the street (on the bottom). A solution for the last issue is to cutout only the relevant parts of the panorama by preserving only a predefined horizontal band. Moreover, the perspective cutouts need to be converted to an undistorted image to avoid incorrect matching. This process is formally known as *gnomonic projection* and it represents the projection of the surface of the sphere from its centre onto a tangent plane.

1.3.2 Temporal sparseness

The environment changes with time and it's hard to think that two different pictures have been taken at the same place and at the same time. Furthermore, if this were the case, the problem could be solved with just an image

retrieval approach by performing a similarity research. Dealing with samples distributed over a period of time can be lead to a *temporal sparseness problem*. Thus, time influences photos because of different factors:

- **Lighting condition** (day/night/golden hour)
- **Structure** (advertisement poster, work in progress building, new buildings, ..)
- **Season** (winter/summer/autumn)
- **Weather** (sunny/rainy/snowy)
- **Dynamic object** (vehicles, pedestrian)

The geolocalization task is highly related to the dynamic nature of our world. Images of a place could contain moving elements such as pedestrians and vehicles that might add noise or even occlude relevant portions of the scene. The appearance of a place itself changes continuously and would be impossible each time to have a fresh image for a given query. Other major points are lighting/weather conditions and seasons. All that can make the same place appear totally different.

There aren't fully working solutions to overcome all the aforementioned challenges, but different ideas have been introduced with the intent of mitigating these issues. Of course, having a big and high quality dataset can improve the generalization capabilities of the model. One of the approaches is to adopt some Domain Adaptation or Generalization tasks in order to find a representation, on the feature space, that is as much as possible independent from these factors.

1.3.3 Spatial sparseness

Data is not only sparse on the temporal dimension but also in the spatial dimension. It would be wrong to think that all the photos taken in a given location have been taken exactly from the same position and with the same camera orientation. We can identify several sources of spatial sparseness:

- **Occlusions** An occlusion can be defined as an element that covers some interesting portion of the image. An example of occlusions is a person (i.e. in a selfie) or a bus parked in front of a building and so on.

- **Viewpoints** Another source of spatial sparseness is represented by the different viewpoints.
- **Recurrent patterns** Another challenge to be addressed are the recurrent architectonics patterns. In totally different locations a common pattern can be found (i.e. bridges, skyscrapers, ...).

In regards to the occlusions there isn't so much that can be done. What really matters is the size of the occlusion and the portion of relevant information it covers. Different experiments propose semantic segmentation approaches in order to identify some of the occlusions to help the network.

Another typical issue with VPR is the viewpoint mismatch between the query and the most relevant database picture. For example the Eiffel Tower is visible from many different locations.

To mitigate this issue different techniques can be used like exploiting 3D information or making usage of a big dataset in which the robustness, in relation to the viewpoints, can be learned in an autonomous way by the neural network.

Lastly, it is common to be affected by repetitive patterns that increase similarity among different places. Various solutions [15] have been studied to approach this problem like the introduction of an *attention module* that helps to focus only on relevant parts of the image[8].

1.3.4 Large Scale and Scalability

Large Scale As already mentioned this work is focused on large scale solutions for the geolocalization task. As a consequence other challenges need to be solved:

- **Representation** We need a compact but meaningful representation. We want to have a representation that is as small as possible to reduce both memory and time complexity. This also opens the door to other post processing techniques that can be further introduced in the pipeline with a small overhead. At the same time the features should be meaningful so that two images taken in a near location should be as similar as possible while images taken in different places should be as distant as possible on the feature space even if they are really close in the visual domain. The representation should also be robust (noise, sparseness, ...) and possibly fast to compute.

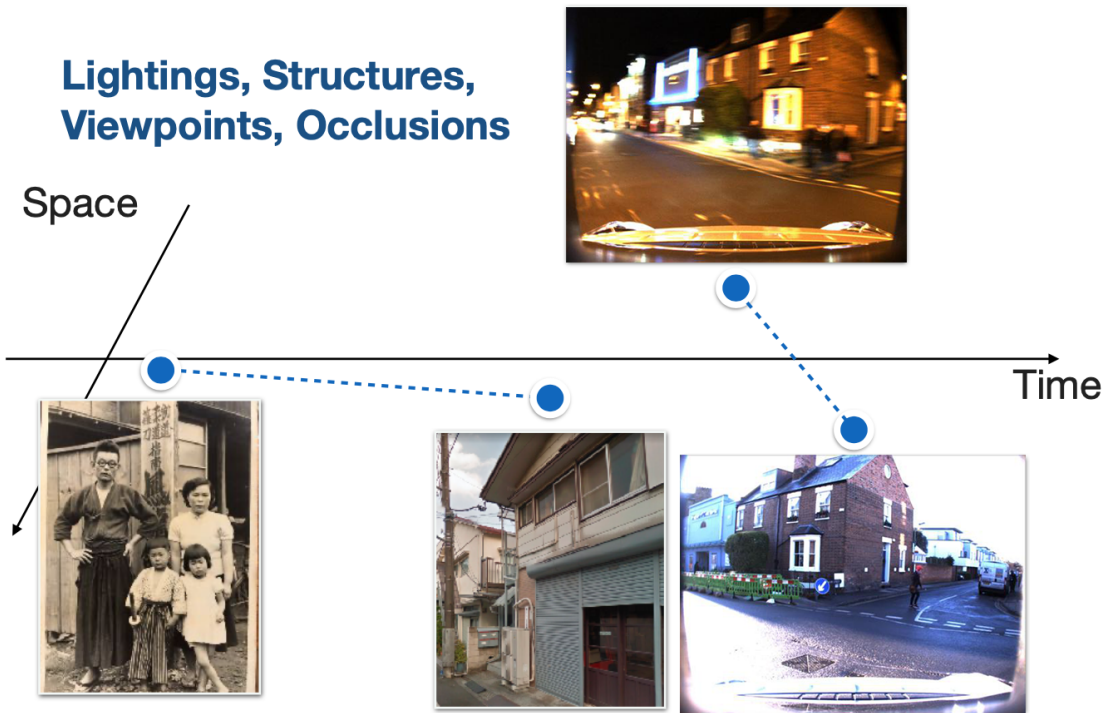


Figure 1.3. Sparseness examples. A real dataset is affected by both sparsity.

- **Performance** In a large scale environment, performance plays a key role. When talking about performance we are referring to both time and space complexity. In fact, just by intuition, geolocation indoors, like inside a museum, rather than on a countrywide, is totally different in terms of performance. More details about this point will be given in the following chapters.
- **Dataset** A dataset of a certain size and with a full coverage of places is required and it represents one of the biggest concerns in regards to the memory complexity because of the great amount of samples (tens of millions) involved.

Scalability Scalability can be considered as one of the current bottleneck for VPR architectures. Ideally, the system should be highly scalable so that a larger scale is always possible to be achieved by providing the system enough resources. For example, the architecture should support increasing numbers

of queries and numbers of places involved in the place recognition process without loosing in both accuracy and performance. In practice this is really hard to accomplish because of the several factors involved such as:

- **The number of concurrent queries involved** The system should also work under the condition in which many queries are given as input at the same time.
- **The number of places** The system should scale with respect to the geographical size of the places in the gallery.
- **The dataset size.** The architecture should be able to use millions of images.
- **Modularity** A good modular design should allow both vertical and horizontal scaling.
- **Flexibility** Components of the system should be defined as optional. Other components should be available according to the context.

1.3.5 Domain Adaptation

We have seen that approaching VPR on such a large scale requires to deal with different kinds of challenges, one of them being the several *visual domains* to which a photo can belong.

As we have already mentioned, it's legit to assume that the system could be fed with images taken in a sunny or snowy day, at night or during daytime and so on. Moreover, a picture taken at the same place but in a different time can appear so much different because of several factors. This phenomena can lead the network to bad performance because of its inability to generalize across different visual domains. When dealing with large scale environment, this becomes a serious issue, since both the gallery and the queries can often belong to different domains. For example let think of a street-view image taken at daytime compared to a query image taken by a smartphone at night.

One way to tackle this challenge is by using a domain adaptation approach. We can define the domain adaptation as the task of adapting models across different (but related) domains. We can distinguish between the *source domain* and the *target domain*.

With domain adaptation the goal is to access to both source and target samples to make it possible to build a robust representation that correctly fits both.

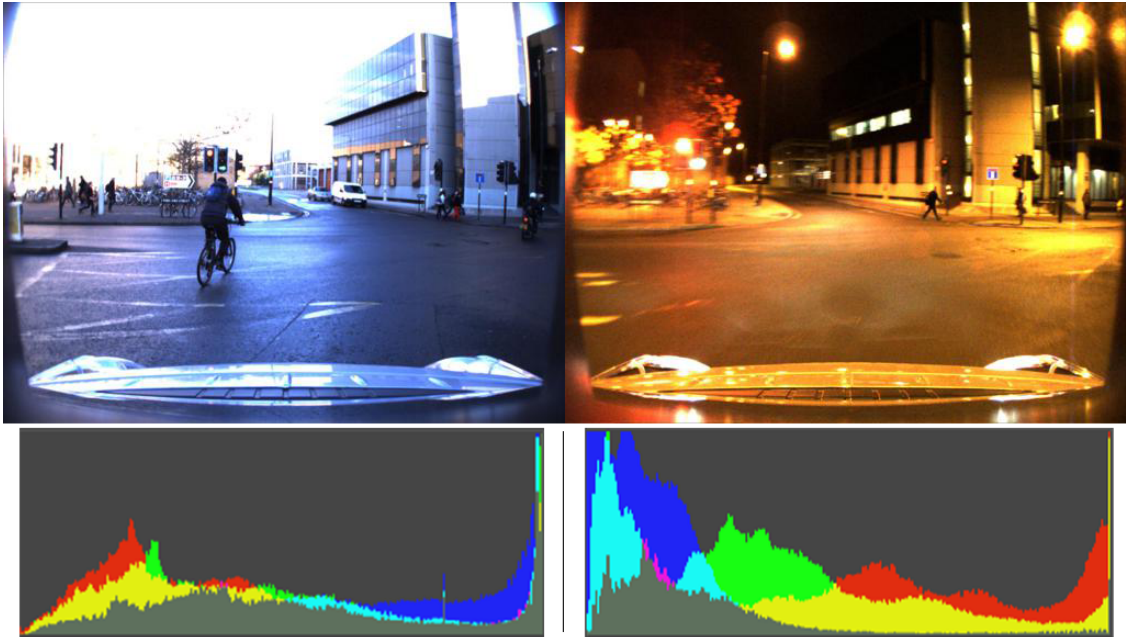


Figure 1.4. Two images from the Oxford RobotCar dataset. On the left a picture taken at daytime; on the right a photo at night. On the bottom their related histograms. You can see how the distribution of RGB pixels highly differs between the two.

Looking at the previous example, we want to find a descriptor that allows us to correctly represent both the day images in the gallery (*source domain*) and the night (*target domain*) query, in order to be able to correctly find a match between the two. From an analytic point of view, different domains mean data with a mismatched distribution. Domain adaptation aims to correct this distribution misalignment and to narrow down the distribution discrepancy. For an intuitive explanation, as you can see by looking at night and day samples from Oxford RobotCar dataset and their relative histograms (Figure 1.4), you should be able to see how night and day images differ and how their RGB distributions appear.

The literature contains several approaches to perform domain adaptation such as:

- Semi-supervised DA
- Weakly-Supervised DA
- Zero-shot DA

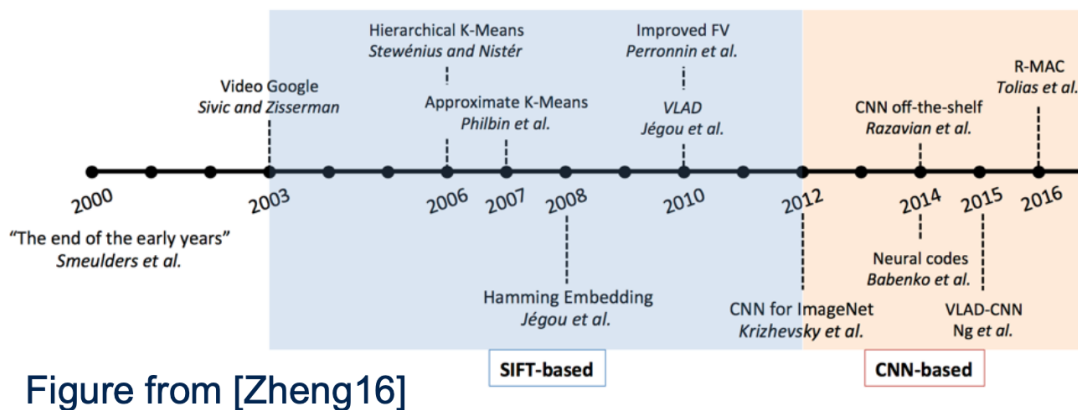
- One-shot DA
- Few-shot DA

For each of them, different solutions have been provided by researchers over the time. In this work, after presenting and showing the results for several architectures we will also present the results obtained by applying different techniques of domain adaptation to see if it's worth considering for geo-localization tasks and what margin of improvements can be achieved.

Chapter 2

Related work

Figure 2.1. Evolution of VPR SotA techniques over the time.



2.1 Introduction

The VPR topic has been considered of high interest for decades. This interest is motivated by both the relevance that visual place recognition has for many applications and its related challenges.

For instance, in computer vision VPR is often studied as the task of recognizing the location of a single image. In the autonomous vehicle, there are usually multiple sensors providing heterogeneous data, thus VPR algorithms can exploit 3D information (i.e. lidar), sequences of photos to exploit

the time dimension, or some knowledge about the actions or movement performed by such robot.

Another distinction can be made about the definition of a correct identified place. In other words, *how is a valid match defined?*

The answer can be different from application to application. Depending on the dataset and on the context, an acceptable result could be to find a match with the right city. In other applications, a match could be a POI (Point of Interest) or a location within a radius of 20 meters or even predicting the camera position and orientation.

Furthermore, in some cases, the algorithm needs to return just one location that has to be correct, whereas for other purposes, predicting the correct result in the top N suggestions could already be considered enough accurate. By experimenting with such architectures, these requirements need to be known, in order for the system to be designed and evaluated accordingly.

Visual place recognition is usually tackled as an image retrieval task. This approach is characterized by a collection of images (*database*) that contains all the places of interest. For each image in the database a geo-tag, representing the corresponding place, is associated i.e. a GPS coordinate. When an image is given as input (*query*), the system extracts a meaningful representation and searches for the closest match on the feature space. This involves the definition of a *feature extractor* which should produce a representation that has to meet the properties described in the previous section.

In the first decade the features were hand-crafted (i.e. DoG, SIFT, PHOW). In the last decade, thanks to the achieved progresses in the computer vision field and in CNN, the commonly approach has been to use a CNN as feature extractor. This approach allows to obtain a more powerful and generalized representation and opens the door to different improvements.

In addition we would like to highlight a subtle difference when talking about features and representation. Features, sometimes called descriptors, are information extracted from images in the form of numerical values and therefore incomprehensible to humans. Basically there are two main types of features based on the application. We distinguish the so called *local features* from the *global features*.

The first ones are used for object recognition/identification while the latter are usually used in image retrieval, object detection and classification. Local features are used to extract all possible local information. Some examples are SIFT, RootSIFT, SURF, BRISK, FAST, ORB.

On the other side, the purpose of global features is to describe the image as a whole and try to generalize the entire image whereas the local features try

to describe just a portion of the image (or *key points* in the image). Compared to the representation based on local descriptors, global descriptors are less robust to viewpoint changes, clutters and occlusions [19].

Furthermore, a global descriptor can be obtained both directly (i.e. think about the whole raw image as a global descriptor itself) and indirectly by using local features as starting point, by chaining a step of aggregation by which the local features are aggregated and reduced in size. Sometimes an hybrid approach is chosen [18], [17].

This concept is particularly relevant because both the local descriptors and the aggregation process represent a critical step in the VPR pipeline. Some of the approaches proposed over the years will be described in the next section.

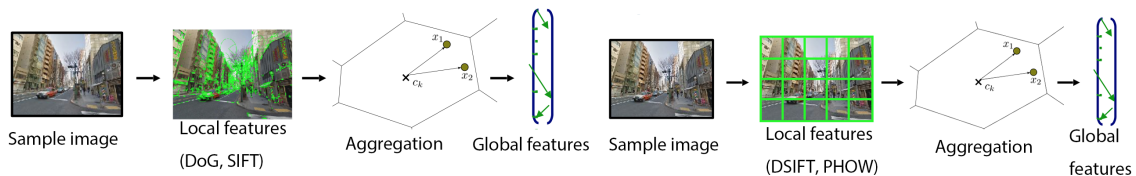


Figure 2.2. Two examples of local (hand crafted) features and global features obtained by aggregation.

Mining

Whenever VPR is approached as a metric learning problem, as in the case of contrastive loss or triplet loss, we need to define correct matches (positives) and false matches (negatives) examples for each training image.

The process of finding such positive and negative examples is called *mining*. The importance of this process should not be undervalued: if the samples are too easy the network will learn a sub-optimal solution. In a similar way, if the samples are too difficult this can lead to over-fitting and bad local minima [19].

Other crucial aspects about mining are performance and scalability. A first idea could be to extract the features for all the negative images but this would require inference for all the images in the gallery. Furthermore, this would represent a big waste in terms of computational resources for those images that wouldn't be chosen. The common approach is to use different strategies to tackle these challenges, such as:

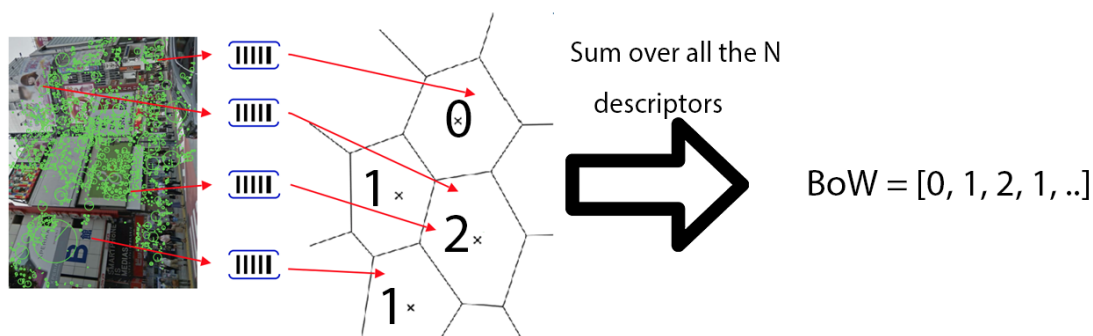
- **Sampling** Only a subset of negative samples is selected and used to compute the loss. Each iteration can reuse the *hardest negatives* mined in the last iteration.
- **Caching** The idea is not to compute the representation each time but only after a well defined number of iteration. That can be thought as a *caching* mechanism where the cache is invalidated and recomputed after N iterations. As a side note this value should be updated according to the learning rate.
- **Clustering** Queries are organized in clusters. Queries in the same cluster share the same negatives.

2.2 Evolution of SotA descriptors

2.2.1 BoW

One of the oldest approach for aggregation and global features generation is the Bag of Words, also known as Bag of Visual Words (BoVW) or more generically as Bag of Features (BoF). In the natural language processing (NLP) the bag-of-words model is an intuitive representation used in information retrieval, sentiment analysis, etc. In this model, a text (sentence or a document) is represented as the bag (set) of its words but keeping multiplicity. Going back to geolocalization, the proposed solution borrows the idea from the NLP domain and associates a vector whose dimension equals the size of the dictionary that contains the *visual word* frequency, or the related TF-IDF representation, for a specific image[1]. In this way, the similarity

Figure 2.3. Representation of the BoVW approach.



among the global descriptor vectors can be easily computed by a simple inner product of their respective visual word vectors. The dimensionality of the features is the same as the number k of the defined clusters. Usually the clusters (*visual words*) are defined by performing k -means clustering.

Thus, the BoW representation can be seen as the histogram which correlates the number of occurrences to visual words. So, it produces a k -dimensional vector, which is then normalized. There are several ways to normalize the histogram that proved to be helpful. If seen as an empirical distribution, the BoF vector is normalized by using the Manhattan distance. Another common choice is to first use the L2 normalization and then weigh the vector with IDF (inverse document frequency)[9].

2.2.2 VLAD

A further step from the BoVW approach is performed with the introduction of VLAD (Vector of Locally Aggregated Descriptors) [4]. The idea takes the old approach as a starting point, but it is based on the questions *Why to store just the presence information? Could be useful to store not just the presence in the cluster but the distance vector from its cluster center too?* Thus, the idea proposed by VLAD, involves a new method of aggregation that is not merely based on the presence but also on the distance from the center of the cluster.

Formally, VLAD computes and stores the sum of residuals (difference vector between the descriptor and its corresponding cluster centre) for each visual word. So, given N D -dimensional local image descriptors x_i as input, and K cluster centres c_k as VLAD parameters, the output VLAD representation V is KD -dimensional. We write V as a $K \times D$ matrix, then, after normalization, used as the image representation. The (j, k) element of V is computed as follows:

$$V(j, k) = \sum_{i=1}^N a_k(x_i)(x_i(j) - c_k(j))$$

where $x_i(j)$ and $c_k(j)$ are the j -th dimensions of the i -th descriptor and k -th cluster centre, respectively. $a_k(x_i)$ represents the membership of the descriptor x_i to k -th visual word, i.e. it is 1 if cluster c_k is the closest cluster to descriptor x_i and 0 otherwise. Intuitively, each D -dimensional column k of V contains the sum of residuals $(x_i - c_k)$ of descriptors which are assigned to cluster c_k . The matrix V is then L2-normalized column-wise

(intra-normalization [10]), converted into a vector, and finally L2-normalized in its entirety [4].

In this way we create a trade-off between the global descriptor size and the granularity of the stored information. In other words, the VLAD solution increases slightly the memory complexity because we need to store the VLAD vectors and not just a BoF but this allows us to have a more detailed information in the descriptor. In particular, the size of features can be formalized as $K * D$ where usually $K = 64$ is chosen.

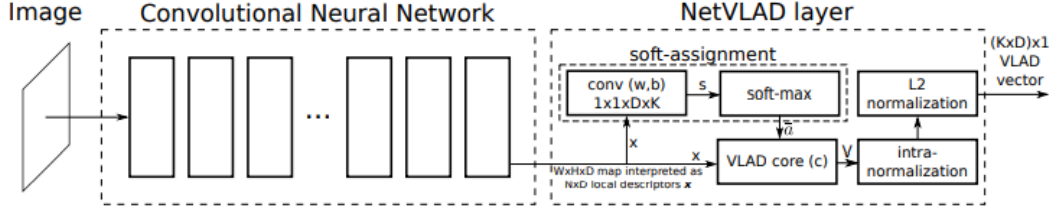
Since the memory complexity increases, for this method to be scalable, a further dimensional reduction technique should be used. One of the most common choice is the PCA (Principal Component Analysis). The usage of PCA partially solves the memory complexity by reducing the size of the features. However, this raises new issues. In a large scale system, it's safe to assume that a big gallery is involved and for each image in the gallery we need to compute its VLAD representation and then apply PCA to reduce the dimension. Thus, the addition of the features reduction technique increases the complexity of both the overall pipeline and the time.

2.2.3 NetVLAD

A few years later since the introduction of VLAD, a new improvement has been proposed [5]. The authors define a new layer called NetVLAD. The idea behind this altered version of VLAD is based on making the VLAD layer differentiable so that a full end-to-end training can be performed. In fact, in standard VLAD, the anchor is chosen as the cluster centre. With NetVLAD it is possible to learn a better anchor which causes the scalar product between the new residuals to be small. The new approach not only improves the SotA performance but also allows us for an end to end training of the whole architecture. Thus, the layer is readily pluggable into any CNN architecture and amenable to training via back-propagation. The proposed architecture and training procedure significantly outperform non-learned image representations and off-the-shelf CNN descriptors on challenging place recognition and image retrieval benchmarks.

The first step to create a *vlad differentiable layer* was to identify the source of discontinuity in VLAD. Intuitively, the hard assignment $a_k(x_i)$ of descriptors to clusters centres is such an issue. So, they replaced it with soft assignment of descriptors to multiple clusters by assigning the weight of descriptor x_i to cluster c_k proportionally to their distance. Formally they defined the

Figure 2.4. CNN architecture with the NetVLAD layer. The net can be implemented using standard CNN layers (convolutions, softmax, L2-normalization) and one easy-to-implement aggregation layer to perform aggregation joined up in a directed acyclic graph. Parameters are shown in brackets.



new task $\overline{a}_k(x_i)$ as:

$$\overline{a}_k(x_i) = \frac{e^{-\alpha\|x_i - c_k\|}}{\sum_{k'} e^{-\alpha\|x_i - c_{k'}\|}}$$

By expanding the square and by defining $w_k = 2\alpha c_k$ and the scalar quantity $b_k = \alpha\|c_k\|^2$, the above equation can be rewritten as:

$$\overline{a}_k(x_i) = \frac{e^{w_k^T x_i + b_k}}{\sum_{k'} e^{w_{k'}^T x_i + b_{k'}}$$

The final form of the NetVLAD layer is obtained by plugging the soft-assignment above into the VLAD descriptor (as defined in the previous section) resulting in:

$$V(j, k) = \sum_{i=1}^N \frac{e^{w_k^T x_i + b_k}}{\sum_{k'} e^{w_{k'}^T x_i + b_{k'}} (x_i(j) - c_k(j))$$

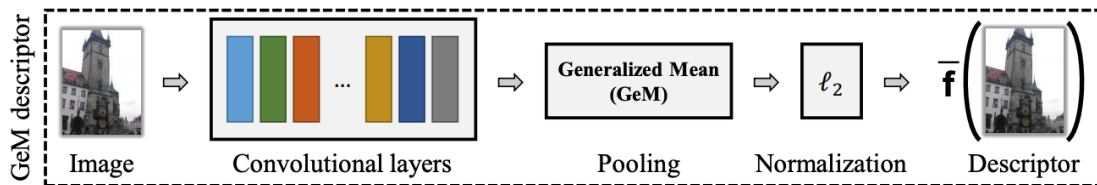
You can easily note that w_k , b_k and c_k are sets of trainable parameters. So, the NetVLAD layer is characterized by three independent sets of parameters, compared to the only one already present in the original VLAD (c_k). According to the authors, this helps to increase the flexibility of the architecture.

Although NetVLAD can help in increasing accuracy, it suffers from the same scalability issue of VLAD because the extracted features are still the same size.

2.2.4 GeM

Just a couple of years ago, a new idea was provided in [6] to generate a compact descriptor. They proposed a novel trainable Generalized-Mean (GeM) pooling layer that generalizes max and average pooling and showed that it can even boost retrieval performance.

Figure 2.5. CNN architecture with the NetVLAD layer. The layer can be implemented using standard CNN layers (convolutions, softmax, L2-normalization) and one easy-to-implement aggregation layer to perform aggregation. Parameters are shown in brackets.



Generalized Mean Pooling (GeM) computes the generalized mean of each channel in a tensor. Formally GeM can be defined as:

$$\mathbf{e} = \left[\left(\frac{1}{|\Omega|} \sum_{u \in \Omega} x_c^p \right)^{\frac{1}{p}} \right]_{c=1, \dots, C}$$

where p is a parameter. Setting this exponent as $p > 1$ increases the contrast of the pooled feature map and focuses on the salient features of the image. The idea behind GeM is the generalization of the average pooling commonly used in classification networks ($p = 1$) and of spatial max-pooling layer ($p = \infty$). The proposed pooling layer not only performs on pair with other SotA architectures but it reduces the features dimensionality of more than one order of magnitude.

2.3 Other approaches

The task can also be solved as a classification task [7]. In Figure 2.7 the region clustering proposed in the PlaNet paper.

Tackling geo-localization with a classification approach allows to have more sample for a given label and reduce the time complexity because no *mining* is required.

Figure 2.6. GeM compared with max and average pooling.

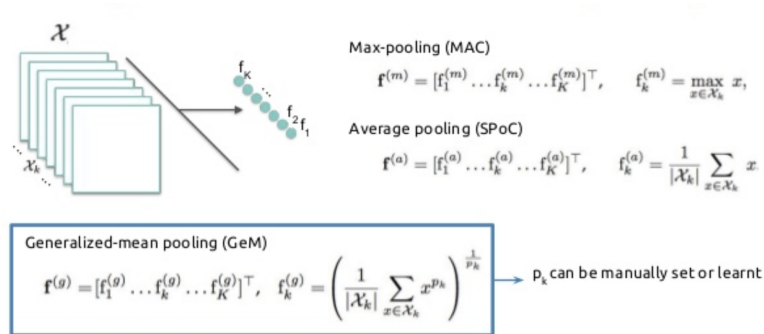
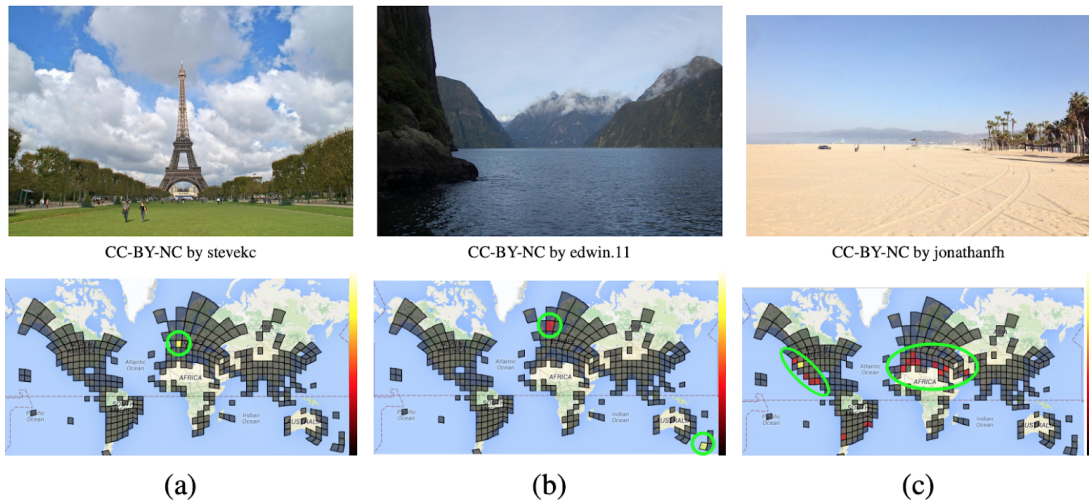


Figure 2.7. PlaNet: a classification approach.



Other proposed approaches are also based on:

- MAC/R-MAC [11]
- Fisher Vector [12]
- 3D Point cloud
- Exploit depth information
- Combine aerial and ground taken images
- Use satellite images

- Landmark classes [13]
- Use of sequence of image to exploit the temporal dimension
- Multimodal exploitation[14]

The list is endless and many other approaches are continuously being studied.

Chapter 3

Data and evaluation

3.1 Dataset

3.1.1 Introduction

A wide range of different datasets are available but they present differences according to the task they were produced for. We can distinguish them among various categories like robotic and non-robotic dataset, landmark dataset, streetview like dataset. Robotic datasets are usually made up of sequences of images, landmark datasets contain photographs of points of interest and their names. StreetView like datasets can be considered similar to the robotic ones in terms of viewpoints (usually a camera is put in the front/rear of the vehicle) but not organized as a sequence.

Almost all experiments listed in this work were performed with Pitts30k and SVOX dataset. Both datasets fall in the StreetView/Robotic category and each image is labeled with its relative GPS coordinates.

Furthermore a new gallery dataset, based on streetview, containing almost every street of all the regional capitals was adopted as gallery for the developed software.

3.1.2 Research dataset

The datasets Pittsburg30k and SVOX have been used to try out different architectures.

In regards to Pitts30k it was pruned from all the images that had no gallery within a 10 meters radius. The gallery of each set (*train*, *test*, *val*) is composed of 10 thousands samples and the database of about 7 thousands.

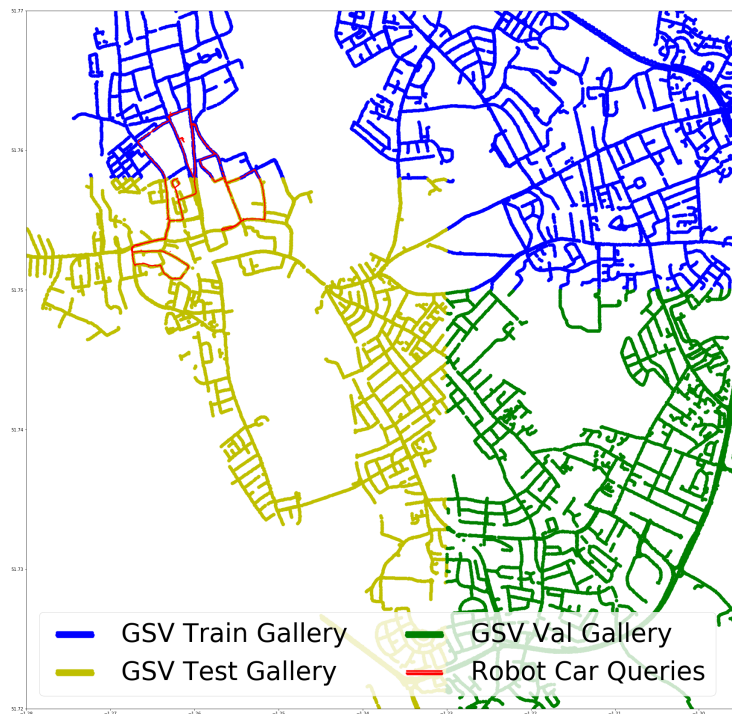
SVOX is a dataset composed of 384x512 pixels images. The dataset contains samples taken from both Oxford RobotCar and StreetView. Thus, the dataset contains a subset of pictures taken in different conditions:

- Sun
- Snow
- Rain
- Night
- Overcast

In details, for all the experiments but the domain adaptation ones, the shown results will be related to the model trained and tested only on sunny pictures. For the domain adaptation experiments on SVOX the train set was the same but testing was performed on both sunny and night domain. The Figure 3.1 shows the dataset composition.

All the images fall in the geographical patch identified by the following coordinates (51.72, -1.28) ; (51.77, -1.195).

Figure 3.1. SVOX composition showed on map.



For what concerns the streetview samples, the panos from 2012 are used in the gallery, while the panos from 2014 are used as queries. Furthermore, for each pano at a given location, just two crops (front and rear) have been taken to maintain consistency with the photos sampled from RobotCar. Each sample is labeled according to the utm coordinates where the picture was taken.

3.1.3 Development dataset

Although the research dataset can be useful to compare results with different architectures and perform benchmarks, this kind of dataset is limited for the development of a real geo-localization software.

The first and easiest reason is that the gallery set should contain samples within the area to be localized. For this reason a new gallery dataset was defined for a whole country: Italy.

The dataset contains samples for all the regional capitals in Italy. As a source for our data we leveraged Google Street View samples. Google Streetview images are taken by cars equipped with a 360° camera that takes a picture every few meters. Each Streetview Panorama is an image that provides a full 360 degree view from a single location (equirectangular projection). Each image contains the exact GPS location¹ of where it has been taken and some other information like the timestamp. All the images have been taken during daylight hours, and cover almost every street. Furthermore, images taken over a few years period are used to improve the robustness and generalization capabilities of the model.

The pipeline adopted to create a similar dataset was the following:

- Download each pano and its metadata for all the places inside a given zone of interest (i.e. Italy, Rome, Sicily)
- Cut the region of interest as horizontal band (512x3584) and define N_c number of crops² (in our case $N_c = 7$).
- Save each undistorted crop and associate its label (its geo-tag)
- Organize the samples in a structured format

¹Note that current GPS technologies usually provides an accuracy within a 3 meters radius.

²Our crop are defined so that they are not overlapping.

3.1.4 Domain Adaptation dataset

As discussed earlier in Section 1.3.5, the challenge of domain adaptation is to try to find a representation that fits both a source and a target set. The first issue in this regard is that we need to have (or a way to generate) samples from both domains.

In the following chapter we will describe three different approaches used to perform domain adaptation. We should anticipate that for these approaches to work we require a method that allows us to perform a domain shift from day lighting to night. So, *how can a daytime image shifted to night for a given sample?*

There are different approaches useful in that regard like an auto encoder, a GAN or other techniques. In our experiments we have chosen two recent methods for doing that:

- **FDA** (Fourier Domain Adaptation for Semantic Segmentation) [29]: Domain adaptation via style transfer made easy using Fourier Transform.
- **CUT** (Contrastive Learning for Unpaired Image-to-Image Translation)[30]: a GAN architecture which aims to generate images so that each patch in the output should reflect the content of the corresponding patch in the input, regardless of the domain. In our task this is extremely useful because the goal is to change the lighting condition while preserving the structural information.

FDA One of the biggest advantage of this architecture is that involves no adversarial training. This is a really interesting property in our large scale context. The idea is to perform domain adaptation via style transfer by using Fourier Transform to shift the spectrum of the source image to the target one. The method can be applied by following three steps:

- Apply FFT to both source and target samples.
- Replace the low frequency part of the source amplitude with that from the target.
- Apply inverse FFT to the modified source spectrum.

CUT Because of some artifacts in the images generated with FDA, we wanted to try an alternative method based on GAN. This approach even if it

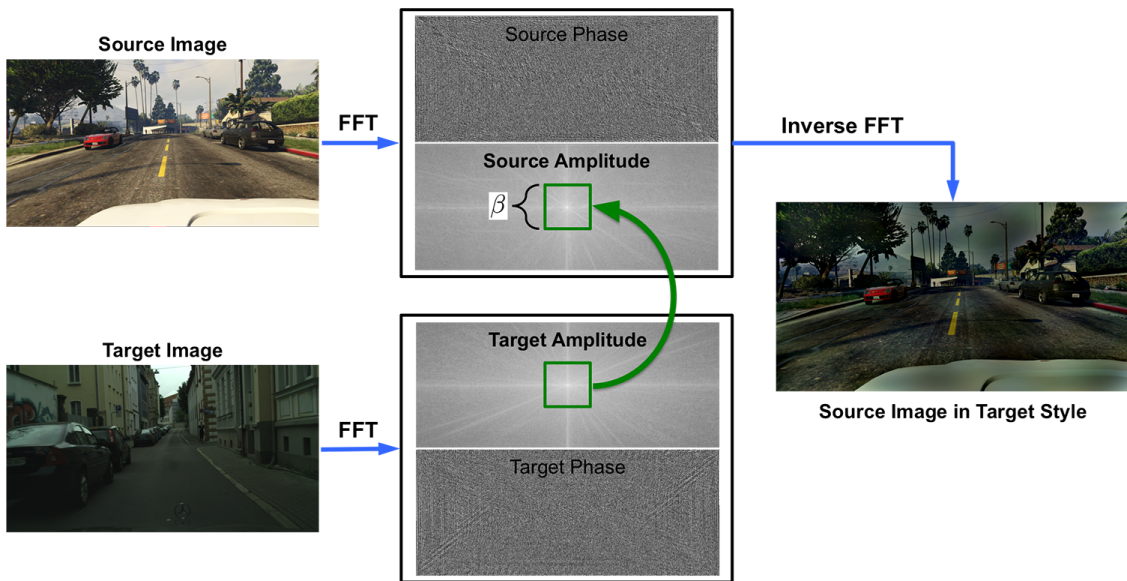


Figure 3.2. FDA diagram[29].

isn't as fast as FDA and need an adversarial training, it is still highly efficient with respect to other GANs. Moreover, we haven't used the whole CUT but its optimized and reduced variation called FastCUT. The reasoning behind is the low memory and computation footprint. The FastCUT network was trained in more stages on 2 different datasets:

- KAIST[31]
- Oxford RobotCar[32]

Moreover, apart from the good performance and high efficient of the network, CUT idea is really a good fit for our use case because of its working principle.

Quoting the authors:

In image-to-image translation, each patch in the output should reflect the content of the corresponding patch in the input, regardless of the domain. We propose a straightforward method for doing so – maximizing mutual information between the two, using a framework based on contrastive learning. The method encourages two elements (corresponding patches) to map to a similar point in a learned feature space, relative to other elements (other patches) in

the dataset, referred to as negatives [..]. Notably, we use a multi-layer, patch-based approach, rather than operate on entire images. Furthermore, we draw negatives from within the input image itself, rather than from the rest of the dataset. [30]

In other words, this approach try to enforce some constraints on the structural shapes of the images, thus avoiding any deformation of the photo. Other GANs don't show such behaviour and can be used to generate deformed samples (i.e. GANs for face generation).

To train the CUT model we used two datasets which already contain samples belonging to both domains: Oxford RobotCar[32] and KAIST[31]. In particular, for RobotCar we used only 2 subsets in the training set: images taken with night and daytime lighting.

For the KAIST dataset we used the split suggested by the authors named *train-day-02* and *train-night-02*. Both datasets contain *unaligned* samples from day and night domains, in other words we haven't a night match for each daytime sample but we have just a bunch of night and day images which the network can learn from.

3.2 Evaluation

The performance is measured on two publicly available image retrieval datasets: Pittsburg30k and SVOX. For both, a set of predefined queries with relative ground truth is used. Typically, both *mean average precision (mAP)* and *recall* are often used to perform retrieval benchmark.

We choose to perform evaluation by recall.

3.2.1 Recall

The recall is formally defined by:

$$R = \frac{T_p}{T_p + F_n}$$

where T_p indicates the *true positives* and F_n represents the *false negatives*.

In particular, we choose to measure the retrieval performance in terms of recalls at k : R@1, R@5, R@10, R@20.

The usage of recall at k is justified by the task we want to accomplish. The idea is that the final system to be developed should provide as output the top- k predictions and not just one. Note also that, in some deployment

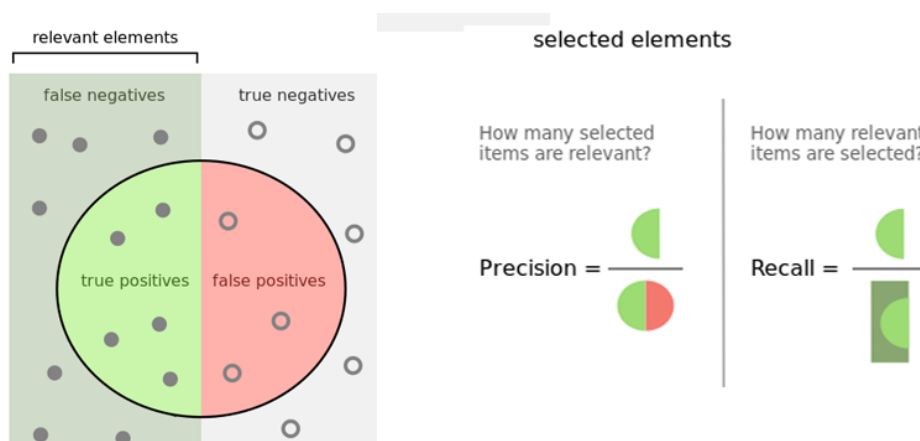


Figure 3.3. Comparison between precision and recall.

setup, multiple years/crop of the the same locations could be in the gallery and so also different outcomes should be considered as correct. In other words, the recall at k is the proportion of relevant items found in the top- k results.

The recall at N can be defined in an intuitive way as:

$$R@k = \frac{\text{Number of relevant items found at } k}{\text{Total number of relevant items}}$$

A perfect model has recall equal to 1.

For instance, suppose we want to compute the recall at 5 ($R@5$). Given that 4 out 5 results are considered as correct and that we have a total of 5 relevant photos in the gallery, then $R@5 = \frac{4}{5}$. The Figure 3.3 visually describes the recall.

3.2.2 Model selection

For each presented architecture the results that are shown should be intended as the ones optimizing the *recall at 5*. Of course, the recall is computed on the test set once the training process is totally completed. In other words, the model selection procedure used in the training process, after each epoch, works as follows:

- Compute recall at 5 ($R@5$) with the new model on validation set
- If the $R@5$ is better than the previous best model, then define this as the new best model and save its recall at 5.

In addition, for every model used as feature extractor, a comparison in terms of both size and time complexity is also provided.

Chapter 4

Architectures and benchmarks

4.1 Architectures

4.1.1 Overview

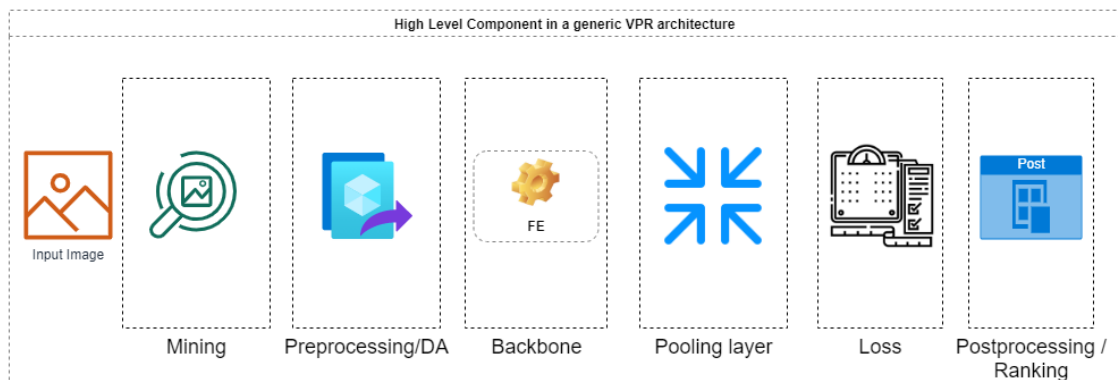


Figure 4.1. High Level Component for a generic VPR architecture.

Before diving into the networks details let's summarize the components that can be considered critical for the task, as described in Figure 4.1. In short, we can identify six key components:

- **Mining** Mining is needed every time a pair of positive and negative is required. Mining solves the issue of finding good negatives.

- **Preprocessing / DA** Usually images are not fed into the network directly (i.e. resize, crop, normalization, ..). During the training we can also apply some Data Augmentation techniques to further improve the generalization capabilities of the model.
- **Backbone** This represents the first real NN component. Usually a SotA ANN truncated at the last convolutional layer is chosen.
- **Aggregation or Pooling layer** The layer chosen to reduce the size of the descriptor given as output of the backbone.
- **Loss** The loss can be considered as part of the architecture.
- **Postprocessing / Ranking** As the last step, we can also adopt some post processing techniques to filter, rank and improve the results.

Not all the aforementioned components should be present or altered among different architectures. Before introducing all the configurations one by one, in this section, we want to provide a summary that describes the components used in all the experiments.

4.1.2 Mining

Mining is the process that aims to find, for each sample, *positives* (correct matches) and *negatives* (wrong matches). The mining process dramatically affects the training process. For example if the mining isn't able to find negatives images that are hard to identify, the model will learn a sub-optimal solution.

The chosen mining implementation was based on the approach suggested by the authors of NetVLAD [5].

To generate the positives for a given query, we select all the images within a radius of 10 meters from the location of the query.

The mining of negatives is slightly more complex. We start by defining a *negative* as any image further away more than 25 meters. Although it would be possible to train the model by choosing every negative for a given query, this should be considered a naive approach because it would be easy for the network to distinguish among totally different images. For this reason we define the *hard negatives* as all the negatives that are similar to the query.

In regards to the challenge of selecting hard negatives to improve the learning process, as we have just seen, we can't just pick random negatives to have a good set of hard negatives. Thus, the mining can be performed by:

- Random sampling 1000 *random negatives*.
- Stacking them with the *hard negatives* generated in the previous epoch. Now we have a set of 1010 negatives.
- Searching for the 10 closest negatives on the feature space in the whole negative set.

According to [5], remembering the previous hard negatives adds stability to the training process. Although the proposed approach would work, it is not feasible or fast enough. In particular, to get the most similar images on the feature space we would need to compute the descriptor for each of the 1010 negatives. In practice, more than 1010 forward passes into the net to process each training tuple would be required.

For this reason, a common approach is to compute, once in a while, the descriptors for the whole query and gallery set. This approach is usually known as *caching*. *Caching* allows us to perform the mining without computing the image representation on the fly, just by using the cached descriptor. As a consequence we need to perform both backward and forward pass to the network only for the 10 *hard negatives* that have just been found. Of course to choose how often the cache has to be refreshed, we need to find a trade-off between epoch duration, convergence and quality of the solution. In this regard, we confirm that refreshing the cache every 500/1000 queries allows us to reach a good trade-off.

4.1.3 Backbones

Another key component of the architecture is represented by the *backbone*. This corresponds to our *local feature extractor*. As already mentioned, the oldest approaches were characterized by using some handcrafted features, but, today, thanks to the progress in the CV field, it has been proved that CNNs as feature extractors are usually a better choice for a good variety of applications.

The proposed architectures leverage two state of the art networks, truncated at their last convolutional layer, as local feature extractor: ResNet and EfficientNet.

It's worth mentioning that, relatively to the VPR task and according to our experiments, we found out that a deeper backbone doesn't always improve the performance of the overall architecture. Moreover all the models used as backbones were preloaded with weights trained on ImageNet[24], concept that is usually known as *transfer learning*.

ResNet A residual neural network (ResNet) is an artificial neural network (ANN) inspired by pyramidal cells in the cerebral cortex. For this reason, residual neural networks introduce *skip connections*, or *shortcuts* to jump over some layers.

One of the biggest motivations behind is to tackle the problem of vanishing gradients affecting deep ANN that makes usage of gradient-based learning methods and backpropagation. It's worth mentioning that ResNet won the

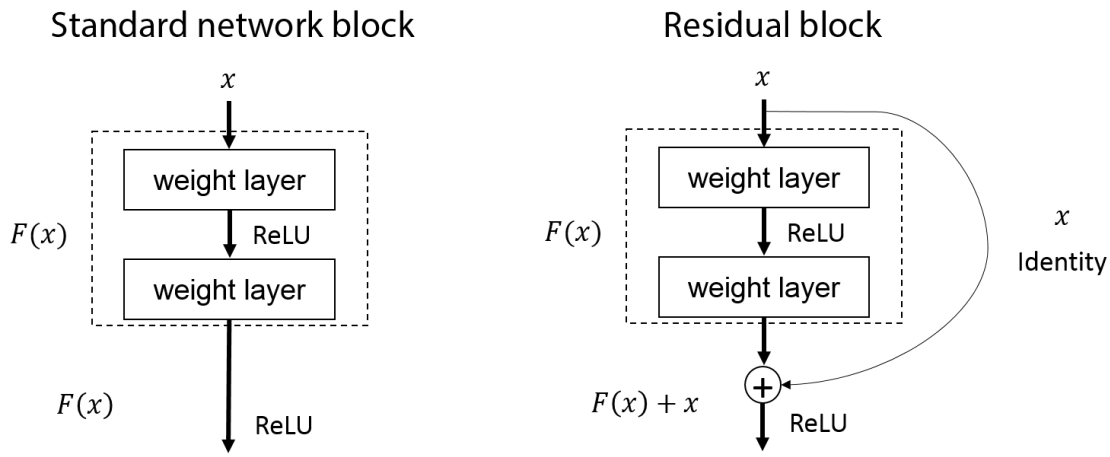


Figure 4.2. Residual and standard blocks compared.

1st place on the ILSVRC (Imagenet Large Scale Visual Recognition Challenge) 2015 taken by a good margin over the second. ResNet is composed of peculiar blocks, usually known as *residual blocks*. The difference between a residual block and a standard block is the presence of the skipping connection as shown in Figure 4.2. In short, the output of a layer is added to the output of the previous one before feeding it to the next layer. Of course, the authors need to find a solution when the size of the output of different layers mismatches, as usually happens when using convolutional layers.

To better understand the improvements that ResNet was able to introduce, we want to recall some basic concepts about how neural networks training works.

To train a Deep Neural Network we need a *training loss*, a *training algorithm* like the *backpropagation* and an *optimization method*.

Loss The loss is required to measure how wrong is the network at doing predictions.

Backpropagation The backpropagation computes the gradient of the loss function with respect to the weights of the network for a given input-output pair, in an efficient way, by applying the *chain rule*.

The gradients can be seen as a measure of how much each weight contributed to the loss, so that we can change the weight accordingly and reduce such error.

Optimizer We also need an *optimizer* that should update the weights of the model according to the previously computed gradients. In other words, the goal of the *optimizer* is to find the parameters (weights and bias) of the network that minimize the loss function.

Vanishing gradient When dealing with deep neural networks that use backpropagation and gradient-based learning algorithm, the problem known as *vanishing gradient* can be encountered. In particular, the weights of the network are updated proportionally to the partial derivative of the error function with respect to the current weight. Deeper the network higher the probability of having a vanishingly small gradient, in particular for the first layers.

In conclusion, ResNet allows training of networks that are substantially deeper than those used before. The authors provide empirical results to show that this kind of networks can be easily optimized, and can exploit depth to gain accuracy[21]. The authors present networks of different depth obtained as combination of residual blocks. We have both small networks like ResNet18 and ResNet34 and big ones like ResNet50, ResNet101, ResNet152.

EfficientNet In their ICML 2019 paper, the authors propose a novel architecture designed to scale in a structured manner. The older approaches scale network dimensions, such as width, depth and resolution, while the proposed method scales proportionally each dimension with a predefined set of *scaling coefficients*.

They designed and proposed a family of models, called EfficientNet, which improves SotA accuracy with up to 10x better efficiency (smaller and faster)[22].

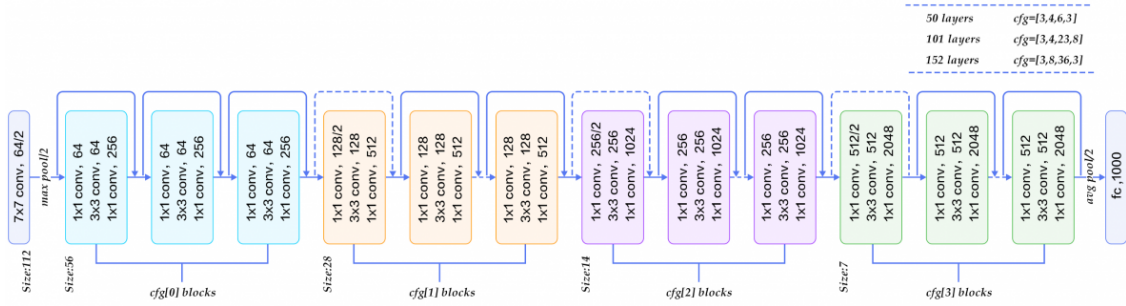


Figure 4.3. ResNet architecture.

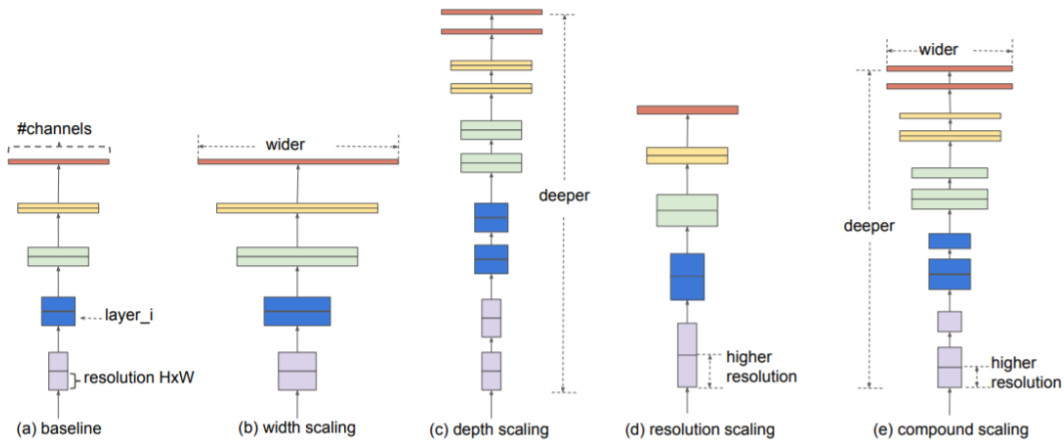


Figure 4.4. Scaling approaches. (a) baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is the proposed scaling method that uniformly scales all three dimensions with a fixed ratio. Figure from [22].

4.1.4 Pooling layer

We used three different global descriptors:

- NetVLAD
- GeM
- CMAP

The first pair of pooling layers were already described in 2.2, when we

talked about the SotA descriptors. But we also introduced a novel pooling layer.

CMAP Motivated by the same observation made by the GeM’s authors, we experimented and designed a new pooling layer called *CMAP* (Chained Max and Average Pooling). The intuition behind is really simple: *What about chaining max and average pooling, instead of generalizing them?*

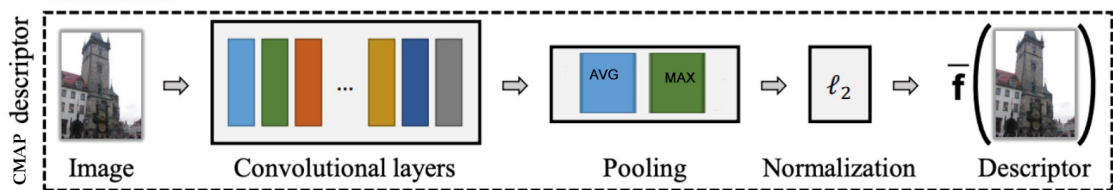


Figure 4.5. CMAP based architecture.

Anyway we want to clarify that the idea of mixing different pooling layers is not a novelty in the CV field[23].

The Figure 4.5 shows a whole feature extractor module with CMAP.

4.1.5 Loss

In all of our experiments we always used the same loss that is a slightly different variant of the Triplet Loss as described in [5]. They call it *Weakly supervised triplet ranking loss*. Two photos taken in the same location could depict different scenes and objects since the orientation of the camera could be different or occlusions could happen. For this reason, for each training query q , the localization information can only be used to define:

- potential positives $\{p_i^q\}$: images close to the query
- definitive negatives $\{n_j^q\}$: images geographically far from the query

In the training process we aim to learn a representation f_θ to improve the VPR performance.

Given a query q , we want to rank higher an image in the database that is geographically close to the query (I_i^*) with respect to all the other far away pictures in the gallery (I_i). Formally, for that to happen, we need to force the Euclidean distance $d_\theta(q, I_i^*)$ between the query q and the closer image I_i^*

to be less than the distance $d_\theta(q, I_i)$ to the images I_i in the database that are far away from the query.

Let’s consider a training dataset composed of triplets $(q, \{p_i^q\}, \{n_j^q\})$ that associate every query q to a set of *potential positives* and a set of *definite negatives*. At first, we wish to find, among all the potential positives, the best matching one. This can be done by looking at the most similar image for each training tuple and by defining the best match $p_{i^*}^q$ as:

$$p_{i^*}^q = \arg \min_{p_i^q} d_\theta(q, p_i^q)$$

To impose that the distance to the best match should be lesser than the distance to the negatives we can formally write the distance constraint as:

$$d_\theta(q, p_{i^*}^q) \leq d_\theta(q, n_j^q), \forall j$$

From here, we can finally define the *weakly supervised triplet ranking loss*:

$$L = \sum_j hl(\min_i d^2(q, p_i^q) + m - d^2(q, n_j^q))$$

where hl is the hingeloss

$$hl = \max(x, 0)$$

and m represents a margin. The last equation says that the overall loss can be computed as the sum of the hinge losses relative to each negative. Given a negative, the hinge loss returns zero if the distance between the query and the negative is greater than the distance to the best positive by a margin m . On the contrary, if the condition isn’t met, it will assume a value greater than zero by a quantity proportional to the violation.

4.1.6 Domain Adaptation

We have already highlighted the different domains to which an image can belong, different lighting condition and places are a serious challenge on large scale environment. One of the way to help the network to deal with such differences is to use *domain adaptation*. Domain adaptation can help both in increasing the overall generalizing capabilities of the network by learning domain-independent features and in increasing accuracy for samples belonging to the target domain.

To study the effect of domain adaptation in a VPR context we designed two architectures. Both are equal except for the way in which the image is shifted from day to night. We present several architectures that were explored and show the results.

First approach

The first idea was to try one of the easier approach such as the domain adaptation with Data Augmentation. We tried to feed the network with 2 triplets. One being the standard triplet that we have already described in the mining section, the other being the same triplet but with altered images for the *query* and the *gallery*. The images were altered by shifting them from daytime to night. We have also redefined the loss to reduce the contribution for the altered images with the introduction of a weight λ . The images were altered by using the FDA approach described in Subsection 3.1.4.

On a high level, this can be considered similar to a self-supervised domain adaptation in which a pretext task is in charge of guessing whether an altered pair of images chosen among all the positives, the queries and the negatives, had been taken in the same location or not. By defining a new loss we don't need anything more to accomplish the task, since we can reuse the triplet margin loss and the already described mining process to accomplish the task. We only need to perform inference with the altered images and compute the loss accordingly, as shown in Algorithm 1.

Algorithm 1 Domain Adaptation training algorithm 1

Triplet from dataset

$$T = \{(x_i^{tp}, x_i^{tq}, X_i^{tn})\}_{i=1}^{N_t}$$

procedure TRAINING(T, S)

foreach *iteration* **do**

 Load triplet T

 Forward to get descriptors of query, positive and negatives

 Compute main loss \mathcal{L}_m

 Shift with a probability p_q the query to night

 Shift with a probability p_p the positive to night

 Forward to get descriptors of the (eventually) altered query and positive

 Compute loss \mathcal{L}_p

 Compute total loss \mathcal{L} and update weights from $\nabla \mathcal{L}$

end procedure

Loss

For these experiments, we trained the network to minimize the loss function

$$\mathcal{L} = \lambda_m \mathcal{L}_m + \lambda_p \mathcal{L}_p$$

The m or p indicates whether the function is referred to the main task or to the chosen variation. Both \mathcal{L}_m and \mathcal{L}_p are based on the triplet loss function. More exactly:

$$\mathcal{L}_m = \sum_j hl(\min_i d^2(q, p_i^q) + m - d^2(q, n_j^q))$$

and:

$$\mathcal{L}_p = \sum_j hl(\min_i d^2(\bar{q}, \bar{p}_i^q) + m - d^2(\bar{q}, \bar{n}_j^q))$$

where the terms with upper bar in the \mathcal{L}_p loss equation denotes the joined features obtained by the combination of the descriptors of both untouched and shifted pairs. The λ weight was introduced to tune each contribution accordingly.

Second approach

In the second approach we wanted to investigate other domain shifting techniques and we decided to replace FDA with FastCUT, described in Subsection 3.1.4. This resulted in more detailed and more appealing images from a human perspective and you can look at a sample image in Figure 4.8.

Third approach

To further study the effects of domain adaptation on VPR we decided to try a third architecture based on the *reverse gradient*. Even if this is not a recent approach [33], the idea behind is as simple as it is clever. The figure 4.6 shows a RevGrad based architecture and how it works.

The proposed architecture is made up of a feature extractor (green) and a deep classifier (blue). Unsupervised domain adaptation is achieved by adding a domain classifier (red) connected to the feature extractor via a gradient reversal layer that multiplies the gradient by a certain negative constant during the back-propagation based training. Otherwise, the training proceeds in a standard way and

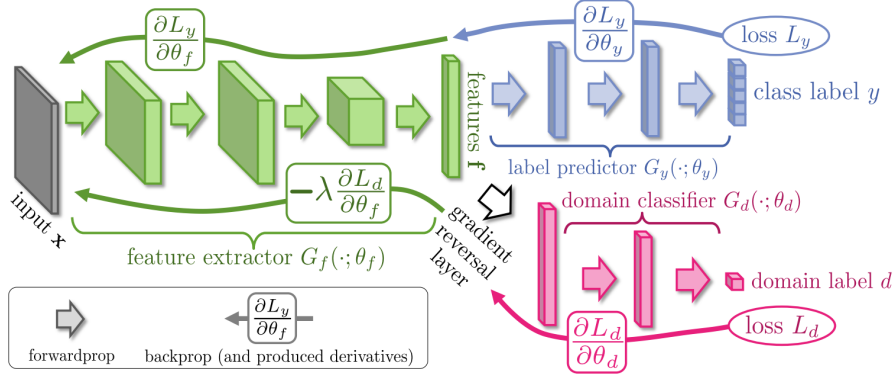


Figure 4.6. A generic RevGrad based architecture [33].

minimizes the label prediction loss (for source examples) and the domain classification loss (for all samples). Gradient reversal ensures that the feature distributions over the two domains are made similar (as indistinguishable as possible for the domain classifier), thus resulting in the domain-invariant features.[33]

To do that, we defined a new classifier which aims to guess if a given image has been taken at day or night. We defined the new head by adding a fully connected layer, just after the pooling, with 1 output class.

In regards to the loss, it can be formally described by the following equation:

$$\mathcal{L} = \lambda_m \mathcal{L}_m + \bar{\lambda}_p \bar{\mathcal{L}}_p$$

with $\bar{\lambda}_p$ weighting factor for the reverse gradient and $\bar{\mathcal{L}}_p$ a binary cross-entropy loss, formally:

$$\bar{\mathcal{L}}_p = \ell(x, y) = \{l_1, \dots, l_N\}^\top, \quad l_n = -w_n [y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n)]$$

The training algorithm for this approach is shown as pseudo-code in Algorithm 2.

Algorithm 2 Domain Adaptation training algorithm 2.

Triplet from dataset

$$T = \{(x_i^{tp}, x_i^{tq}, X_i^{tn})\}_{i=1}^{N_t}$$

procedure TRAINING(T, S)

foreach *iteration* **do**

 Load triplet T

 Forward to common backbone to get descriptors of query, positive and negatives

 Compute (Triplet) loss \mathcal{L}_m

 Shift the query to night q_n

 Forward the transformed query (q_n) to get its descriptor and then forward to the pre-text classification head.

 Compute (BCE) loss $\bar{\mathcal{L}}_{pp}$

 Shift the positive to night p_n

 Forward the transformed positive (p_n) to get its descriptor and then forward to the pre-text classification head.

 Compute (BCE) loss $\bar{\mathcal{L}}_{pq}$

 Compute loss $\bar{\mathcal{L}}_p = \bar{\mathcal{L}}_{pp} + \bar{\mathcal{L}}_{pq}$

 Compute total loss \mathcal{L} and update weights from $\nabla \mathcal{L}$

end procedure

4.1.7 Finetuning

When a good architecture and some good hyperparameters are found is always possible to apply some *finetuning* techniques to improve by a limited amount its performance. Some techniques we experimented with are:

- **Data Augmentation:** ColorHue, Saturation, Horizontal/Vertical Flip
- **Whitening:** Fully Connected layer
- **Normalization:** L1, L2
- **Regularization:** Batch Normalization, Drop Out

The obtained results are reported and discussed in the following section.

4.2 Results and discussion

4.2.1 Overview

Before starting to present the results for each architecture, we provide in Table 4.1 a summary describing the composition of each configuration.

#	Category	Backbone	Pooling	Loss
1	Retrieval	ResNet	NetVLAD	Triplet
2	Retrieval	ResNet	GeM	Triplet
3	Retrieval	ResNet	CMAF	Triplet
4	Retrieval	EfficientNet	GeM	Triplet
5	Retrieval	EfficientNet	CMAF	Triplet
6	Classification	EfficientNet	-	ArcFace
7	Retrieval + DA(FDA)	ResNet	GeM	Triplet
8	Retrieval + DA(FastCUT)	ResNet	GeM	Triplet
9	Retrieval + DA(RevGrad)	ResNet	GeM	Tr. + BCE

Table 4.1. Summary of the experimented architectures.

In the next section we introduce them and the obtained results.

4.2.2 ResNet based architectures

The first set of architectures is characterized by the use of ResNet as local feature extractor. As already mentioned, we want to remark that all the models used as backbone are truncated at the last convolutional layer.

ResNet-18

In the first experiments we used ResNet 18 as backbone and the Table 4.3 and Table 4.4 show the designed configurations and their relative results.

All the experiments were performed with both some fixed hyper-parameters and variable ones.

Hyper-parameters The fixed hyper-parameters used to compare the different pooling layers on ResNet-18 are:

- *Batch Size*: 4
- *Cache Batch Size*: 24
- *Queries/Epoch*: 5000
- *Negatives per query*: 10
- *Deterministic*: True
- *Freezed before*: 3rd set of layers ¹.

On the other hand, the tuned hyper-parameters are:

- *Learning Rate*: $[10^{-7}, 10^{-3}]$
- *Whitening*: Pre/Post/None/Both
- *Normalization*: Pre/Post/None/Both
- *Num Clusters*: $\{40, 64, 100\}$ ².

¹We used the ResNet model as provided by the PyTorch framework. They define the whole architecture as made up by sequential layers. The first three sequential layers were freezed by setting `requires_grad=False`

²This parameter applies only to NetVLAD.

- *Dropout*³: [10, 20]%

In the first set of experiments we wanted to (i) study whether a small network like ResNet18 can lead to good results or not; (ii) which parameters and techniques highly affect the results and (iii) perform a first comparison between the aggregation layers.

Dataset	Whitening	#Epochs	Margin	R@1	R@5	R@10	R@20
Pitts30k Test	Both	2	0.1	69,9	85,7	90,4	93,2
Pitts30k Test	Pre	1	0.1	65,7	83,4	89,2	93,1
Pitts30k Test	Post	1	0.1	70,4	86	90,9	94,1
Pitts30k Test	None	11	0.1	76	88,6	92,4	95,1
Pitts30k Test	None	22	0.01	76,5	90	92,9	95,3

Table 4.2. Results of ResNet18 architecture on Pitts30k (L2 norm + whitening).

Results The overall results, shown in Table 4.3 and Table 4.4, answer the first question and allow us to consider ResNet18 good enough to reach a fair accuracy. This is particularly true when a bigger dataset is used, as can be seen by looking at the different results between SVOX and Pitts30k.

Dataset	Pool	#Epochs	R@1	R@5	R@10	R@20
Pitts30k Test	NetVLAD	5	85,6	92,5	94,5	96
Pitts30k Val	NetVLAD	5	87,8	95,7	97,1	98,1
Pitts30k Test	GeM	22	76,5	90	92,9	95,3
Pitts30k Val	GeM	22	78,4	91,6	94,8	97
Pitts30k Test	CMAF	30	86,5	93,2	95	96,3
Pitts30k Val	CMAF	30	83	91,7	93,8	95,4

Table 4.3. Comparison of top results among aggregation layers with ResNet18 as backbone on Pitts30k.

Many researchers (i.e. [19]) propose whitening techniques to improve the accuracy. More specifically, in [6], the authors suggest the usage of a *fully connected layer* to apply whitening.

³The dropout layer was inserted just before the aggregation layer.

Dataset	Pool	#Epochs	R@1	R@5	R@10	R@20
Oxford60k Test	NetVLAD	45	94,4	97	97,6	98,2
Oxford60k Val	NetVLAD	45	93	96,3	97,1	97,7
Oxford60k Test	GeM	12	87,3	93,5	95,2	96,6
Oxford60k Val	GeM	12	83,5	91,4	93,5	95,2
Oxford60k Test	CMap	15	86,5	93,2	95	96,3
Oxford60k Val	CMap	15	83	91,7	93,8	95,4

Table 4.4. Top results comparison between aggregation layers with ResNet18 as backbone on SVOX.

Furthermore, whitening can be performed both before and after the aggregation layer and we will refer to them as *Pre Whitening* and *Post Whitening*. In regards to whitening, we have got discordant results. Wherever it is applied, it leads to a loss in the accuracy ($\pm 5\%$) and doesn't show any other benefit. It's worth mentioning that to perform whitening we used a fully connected layer with its default initialization⁴.

Dataset	Whitening	R@1	R@5	R@10	R@20
Pitts30k Test	Post	66,1	84,2	89,5	93,2
Pitts30k Test	Pre	63,3	80,2	86,1	90,7
Pitts30k Test	Both	57	78,2	84,5	89,8
Pitts30k Test	None	70,4	86,5	91,2	94,2

Table 4.5. Results with ResNet18 as backbone and gem as pooling layer, **without normalization** on Pitts30k.

Moreover, we wanted to see what the impact of normalization, if applied, before and after the aggregation layers is. The best results were recorded by using a L2 Norm applied at both positions. It's worth mentioning that GeM was the most affected in this regard and the observed difference in terms of recall was $\pm 4\%$ as can be seen by comparing the Table 4.2 with the Table 4.5.

In addition, according to our experimental results and as suggested by many, a value of $k = NumCluster = 64$ leads to the best performance.

⁴We used the PyTorch framework that, by default, initialize linears layers with *He initialization*, as proposed in [27]

In conclusion, the first experiments lead to different results such as:

- Normalization is highly recommended and has the greatest impact on some layers.
- Whitening⁵ doesn't show any benefit to us even if it is applied in different positions and combinations.
- Although ResNet18 is small and lightweight it can still lead to interesting results.
- In accordance to the literature, $NumCluster = 64$ recorded the best results.
- The NetVLAD aggregation layer seems to perform the best when using small backbones like ResNet-18.
- The CMAP aggregation layer was the slowest to converge.

ResNet50

After some experiments backed by ResNet18, we wanted to analyze the impact of other hyper-parameters without having to worry about the possible limitation introduced by a network with limited capacity like ResNet18. For this reason, we moved on and a second set of experiments was performed with ResNet50 as backbone.

Hyper-parameters Below is the list of the fixed parameters used to train the model:

- *Cach Refresh Rate*: 1000
- *Cache Batch Size*: 24
- *Queries/Epoch*: 5000
- *Negatives per query*: 10
- *Deterministic*: True
- *Normalization*: L2 Norm applied both pre and post aggregation.

⁵We applied it with a fully connected layer

As can be seen from the above list, we decided to always apply the L2 normalization because of the previously obtained results. Moreover, we decided to give whitening another chance and performed different experiments to see if it could lead to any improvement.

The hyper-parameters search-space for this set of experiments was:

- *Learning Rate*: $[10^{-6}, 10^{-5}]$
- *Frozen before*: $[0, 3]$
- *Batch Size*: $[1, 4]$
- *Whitening*: Pre/Post/None/Both
- *Loss margin*: $[0.001, 0.9]$
- *Dropout*: $[10, 20]\%$

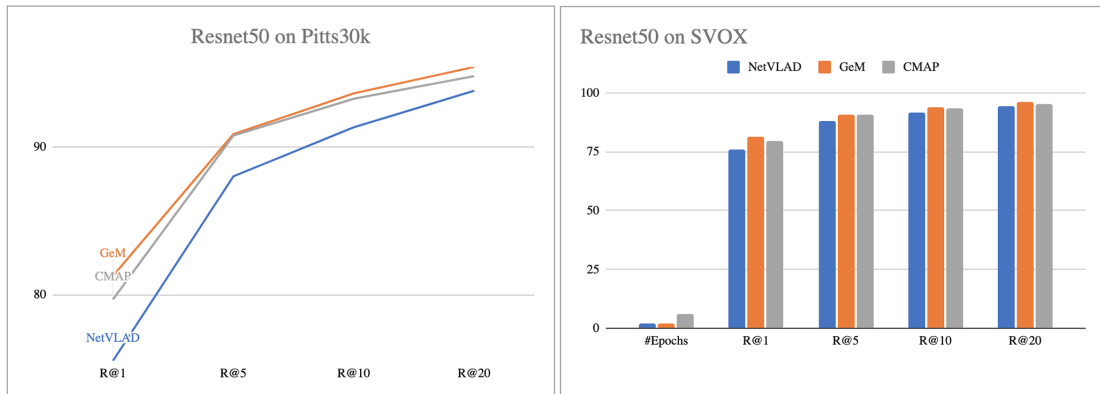


Figure 4.7. Comparison of different aggregation layers with a ResNet50 based architecture on Pitts30k (left) and SVOX (right).

Dataset	Pool	#Epochs	R@1	R@5	R@10	R@20
Pitts30k Test	NetVLAD	2	75,9	87,9	91,4	94,1
Pitts30k Val	NetVLAD	2	78,5	89,9	92,8	95,4
Pitts30k Test	GeM	2	81,2	90,9	93,9	95,9
Pitts30k Val	GeM	2	83,1	92,3	94,8	96,2
Pitts30k Test	CMap	6	79,7	90,8	93,5	95,2
Pitts30k Val	CMap	6	81,7	92	94,7	97,0

Table 4.6. Comparison of top results among aggregation layers with ResNet50 as backbone on Pitts30k.

Dataset	Pool	#Epochs	R@1	R@5	R@10	R@20
Oxford60k Test	NetVLAD	8	88,9	94,4	95,1	96,3
Oxford60k Val	NetVLAD	8	88,7	93,8	95,1	96,2
Oxford60k Test	GeM	12	93,6	96,8	97,6	98,1
Oxford60k Val	GeM	12	91,2	95,4	96,6	97,4
Oxford60k Test	CMap	25	94	96,8	97,5	98,1
Oxford60k Val	CMap	25	92,2	95,9	96,8	97,7

Table 4.7. Comparison of top results among aggregation layers with ResNet50 as backbone on SVOX.

Results By looking at the results, shown in the Tables 4.6 and 4.7, we can easily see that the changes in the backbone were useful to gain some points in the accuracy. But this comes with a cost: the network is slower to train and the extracted features are bigger in size.

Furthermore, if NetVLAD performed the best when paired with ResNet18, with a deeper backbone this is no longer the case and both GeM and CMAP outperform NetVLAD.

When using *transfer learning*, it is often useful to freeze some of the first layers so that their weights are not updated during the learning process. The idea comes from the fact that the first layers can be seen as *low level features* and we can assume them to be the same also among different tasks (i.e. first layers can be thought as *circular shape detectors*, *edges detectors* and similar). In other words, it’s legit to assume that the notion of circular shapes can be useful both to identify a car and a motorbike and so it makes sense to freeze this layer and preserve their parameters.

Under these assumptions we tried to train both the full network (*frozen*

before = 0) and to freeze just the first few layers. The best accuracy was observed with the *frozen before* parameter set to a value of 2. In addition, according to all our experiments, the best learning rate was $lr = 10^{-5}$, independently by the chosen configuration. It’s worth mentioning that a learning rate above 10^{-4} led to divergence for some models.

Moreover, we confirm that *whitening* doesn’t bring any advantage according to our results. That’s why we won’t further investigate it in the following experiments.

Besides the hyper-parameters of the architecture, we have got interesting results by tuning the margin m of the *Triplet Margin Loss*. In particular, the configurations that make use of CMAP and GeM, gained +2% accuracy for $m = 0.02$ compared to the results obtained with $m = 0.1$.

Dataset	NetVLAD	GeM	CMAP
Pitts30k	4,6GB	69MB	138MB
Oxford60k	9GB	134MB	268MB

Table 4.8. Comparison of cache sizes over different aggregation layers.

Moreover, we report in Table 4.8 the size of the cache⁶ on disk during training for the different aggregations layers. The displayed sizes have different magnitude of order and this highlights the importance of the descriptor size from a large scale perspective. This is particularly true for NetVLAD: because of its notable size it would be impossible to use the NetVLAD descriptors right away in the search process and a further step of dimensional reduction (like PCA) is required, thus increasing the time and the easiness of the computation of the offline features for the gallery and the query.

It’s worth mentioning that the addition of a dropout layer with $p = 10\%$ increased the accuracy up to +0.7%⁷. At the same time, the addition of a batch normalization layer between the local feature extractor and the pooling layer led to poorer performance. One likely reason for that, could be the batch size because in our experiments we have never exceed a value of 4, too small to allow any benefit that comes with batch normalization. We have also tried to use a variation of the triplet loss by applying the distance swap as described in [28] but just a slightly decrease in the accuracy was observed.

⁶The cache is saved in the HDF5 data format.

⁷CMAP configuration on Pitts30k

ResNet-101

To check whether ResNet50 was limiting the accuracy we ended up replacing it with the bigger ResNet101.

The results show negligible differences in the performance and highlight the limitations, in terms of memory complexity, of the NetVLAD descriptor. In fact, while the experiments with the other pooling layers gave the same top-results shown for ResNet50, we couldn't ever try out with NetVLAD (*BatchSize* = 1) because we ran *Out Of Memory*⁸.

In conclusion, a deeper model as ResNet101 didn't show any improvement with respect to ResNet50, suggesting that the capacity of the latter can be considered enough for our dataset and task.

4.2.3 EfficientNet based architectures

A second set of experiments was performed with architectures backed by EfficientNet, a novel architecture introduced in the previous chapter. The idea behind is to analyze how much the architecture can be improved in terms of efficiency, scalability and accuracy.

Hyper-parameters

- *Cach Refresh Rate*: 1000
- *Cache Batch Size*: 24
- *Queries/Epoch*: 5000
- *Negatives per query*: 10
- *Deterministic*: True
- *Normalization*: L2 Norm applied both pre and post aggregation.

The results shown in the following paragraph show the top results for each configuration, relatively to the following hyper-parameters.

- *Freed*: [0,25]%

⁸We tried to run the model on a Nvidia RTX 2080 Ti, resulting in 11 GB of VRAM available.

- *Batch Size*: [1,2]
- *Loss margin*: [0.001, 0.9]

Results The results show that replacing ResNet50 with EfficientNet0 led, after a proper tuning, to comparable results on SVOX. On the other side, when using Pitts30k, EfficientNet-B0 seems to improve the accuracy by a few points. The results in Table 4.9 and 4.10 suggest that both EfficientNet and ResNet are suitable backbone for the task. It’s hard to pick a clear winner, depending on the chosen capacity of the network, dataset and pooling layer, the architectures show similar performances.

Dataset	Pool	#Epochs	R@1	R@5	R@10	R@20
Pitts30k Test	GeM	7	88,1	93,9	95,3	96,6
Pitts30k Val	GeM	7	86,4	93	94,6	95,9
Pitts30k Test	CMAF	12	81,7	92,4	94,8	96,4
Pitts30k Val	CMAF	12	83,3	95	97,2	98,3

Table 4.9. Comparison of top results among aggregation layers with Eff-B0 as backbone on Pitts.

Dataset	Pool	#Epochs	R@1	R@5	R@10	R@20
Oxford60k Test	GeM	20	90,9	95,7	97	97,9
Oxford60k Val	GeM	20	89	94,6	95,9	97,1
Oxford60k Test	CMAF	25	92,6	96,4	97,4	98,1
Oxford60k Val	CMAF	25	90,9	95,6	96,7	97,6

Table 4.10. Comparison of top results among aggregation layers with Eff-B0 as backbone on SVOX.

4.2.4 Classification approach

A different experiment was performed by tackling VPR with a classification approach. The tested architecture was composed by EfficientNet as backbone and the ArcFace loss. The experiments performed with such configuration, whose results will be shown later, were run on a different dataset made up from StreetView panoramas sampled from Milan, Turin and Naples. Although the network was trained on a different dataset, the model was tested

against the Pitts30k and SVOX dataset. The reasoning behind the different dataset is mainly justified by the different format needed for the classification task and this should be taken into account when comparing the results. The architecture works by decomposing the map into a grid in which each cell has an area of $M * M$. To avoid potential issues at the margin of each cell, we chose to define different sub-dataset to train the network, defined in such a way that each of them had no cells confining with an other.

In particular, we defined another parameter D indicating how many cells are in between two cells of the same dataset.

Another parameter, *classes per bucket*, defines how many classes should be considered inside a cell.

Hyper-parameters Following is a list of the fixed parameters used to train the model:

- D : 4
- *Cache Batch Size*: 24
- *Min panos per bucket*: 10

The results show the top accuracy for each configuration, relatively to the following range of hyper-parameters:

- M : [10,20]%
- *Class per bucket*: [2,6]
- *Learning rate*: [0.01, 0.001]
- *Freezed*: [30, 50]
- *RYO*: [20, 400]
- *Batch size*: [64, 256]

We used Stochastic Gradient Descent as optimizer.

Results Even if the results can't be fairly compared with the other experiments because of the different overall architectures and training dataset, the results show that the classification approach is a suitable choice in VPR. The model has a really good generalization capabilities as can be seen from the

Dataset	Backbone	R@1	R@5	R@10	R@20
Pitts30k Test	EfficientNet0	86.7	95.4	97.1	98.4
Pitts30k Test	EfficientNet4	88.4	97.2	98.2	99.1
Pitts30k Test	EfficientNet7	89.1	97.0	98.2	99.0

Table 4.11. Top results with the classification approach on Pitts30k.

Table 4.11 and performs well with EfficientNet-B0. Furthermore, the size is comparable to the descriptor’s size produced by using GeM or CMAP.

Moreover, applying some Data Augmentation⁹ techniques, led to a slightly improvement in the accuracy as can be seen from the Table 4.12.

Dataset	Backbone	DA	R@1	R@5	R@10	R@20
Pitts30k Test	EfficientNet4	No	88.1	96.7	98.1	99.0
Pitts30k Test	EfficientNet4	Yes	88.4	97.2	98.2	99.1

Table 4.12. EfficientNet accuracy with and without Data Augmentation.

4.2.5 Domain Adaptation

As discussed earlier, domain adaptation can help to improve the generalization capabilities of a network. In this section we show the results obtained with different approaches.

Baseline

Here we report all the results of our Domain Adaptation experiments applied to the best ResNet50 architecture we have previously reported. The results are shown only for SVOX and the related RobotCar scenarios, since Pitts30k is entirely made up of images taken at daytime.

In Table 4.13 you can look at the recall of the baseline on the target set.

⁹The techniques that returned good results were: *Saturation*, *ColorJit*, *RandPerspective*, *RandRot*



Figure 4.8. On the left a picture from Oxford RobotCar taken at daytime. On the right the same picture shifted to night domain thanks to FastCUT.

Dataset	Arch	#Epochs	R@1	R@5	R@10	R@20
SourceTest	GeM	10	93,6	96,8	97,6	98,1
TargetTest	Gem	10	0	0,4	1,4	3,5

Table 4.13. ResNet50 Domain Adaptation baseline.

First approach

Hyper-parameters

- *Cach Refresh Rate*: 1000
- *Cache Batch Size*: 24
- *Freezed*: 2
- *Queries/Epoch*: 5000
- *Negatives per query*: 10
- *Deterministic*: True

- *Normalization*: L2 Norm applied both pre and post aggregation.

The results shown in Table 4.14 show the top results for each configuration, relatively to the following search space.

- *Batch Size*: [1,2]
- *P_{dropout}*: [0,20]%
- *Loss margin*: {0.001, 0.02}
- λ_p : [0.001, 1.5]
- λ_m : [0.8, 1]
- *P_q*: [0, 1]
- *P_p*: [0, 1]
- *FDA L*: [0, 1]

ResNet50 Results The table below shows the results of the FDA based approach applied to the best performing ResNet50 architecture.

Dataset	Arch	#Epochs	R@1	R@5	R@10	R@20
SourceTest	GeM	13	92,1	95,8	96,8	97,6
TargetTest	Gem	13	1,7	5,7	7,7	11,3

Table 4.14. FDA based DA.

Second approach

Hyper-parameters

- *Cach Refresh Rate*: 1000
- *Cache Batch Size*: 24
- *Freezed*: 2
- *Queries/Epoch*: 5000
- *Negatives per query*: 10

- *Deterministic*: True
- *Normalization*: L2 Norm applied both pre and post aggregation.

The results in Table 4.15 show the top recall for each configuration, relatively to the following search space.

- *Batch Size*: [1,2]
- *P_{dropout}*: [0,20]%
- *Loss margin*: {0.001, 0.02}
- λ_p : [0.001, 1.5]
- λ_m : [0.8, 1]
- P_q : [0, 1]
- P_p : [0, 1]

We used Stochastic Gradient Descent as optimizer.

To train FastCUT we used the following training procedure:

1. 50 epochs on KAIST with $lr = 10^{-5}$
2. 136 epochs on Oxford RobotCar with $beta1 = 0.55$ and $lr = 0.001$
3. 30 epochs on Oxford RobotCar with $beta1 = 0.55$ and $lr = 0.0005$

Dataset	Arch	#Epochs	R@1	R@5	R@10	R@20
SourceTest	GeM	6	84,7	91,7	93,7	95,4
TargetTest	Gem	6	2,9	8	12	15,8

Table 4.15. FastCUT based DA.

Results

Third approach

Hyper-parameters

- *Cach Refresh Rate*: 1000
- *Cache Batch Size*: 24
- *Queries/Epoch*: 5000
- *Negatives per query*: 10
- *Deterministic*: True
- *Freed*: 2
- *Normalization*: L2 Norm applied both pre and post aggregation.

The results in Table 4.16 show the top recalls for each configuration, relatively to the following search space.

- *Batch Size*: [1,2]
- *P_{dropout}*: [8,20]%
- *Loss margin*: {0.001, 0.02}
- $\bar{\lambda}_p$: [0.001, 1.5]
- $\bar{\lambda}_m$: [0.8, 1]
- *P_q*: [0, 1]
- *P_p*: [0, 1]

Dataset	Arch	#Epochs	R@1	R@5	R@10	R@20
SourceTest	GeM	6	90,8	95,2	96,4	97,3
TargetTest	Gem	6	0,7	2,3	8,9	12,6

Table 4.16. RevGrad approach.

Results

Comparison and conclusion

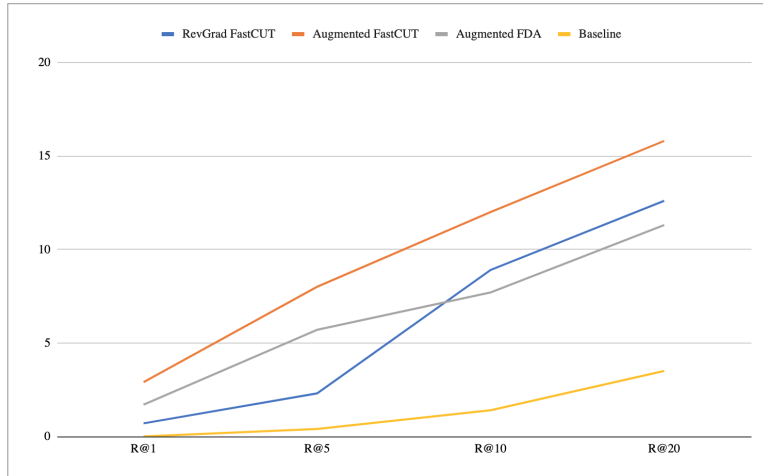


Figure 4.9. Comparison of results for domain adaptation on target set (night condition).

By looking at the Figure 4.9 we can see that all the techniques helped in increasing the accuracy on the target set by a tangible margin. Furthermore, the improvements on the target set only slightly decreased the accuracy on the source set. Moreover, we confirm that FDA is a really interesting method for domain adaptation that doesn't require any training and is really easy to tune and to use also if you need an online solution. We can also conclude that using GANs as data augmentation technique is useful in a VPR context. We also confirm the hardness of *domain generalization* in VPR, a network with a high precision on a specific domain can be totally wrong on a different one.

Chapter 5

Software

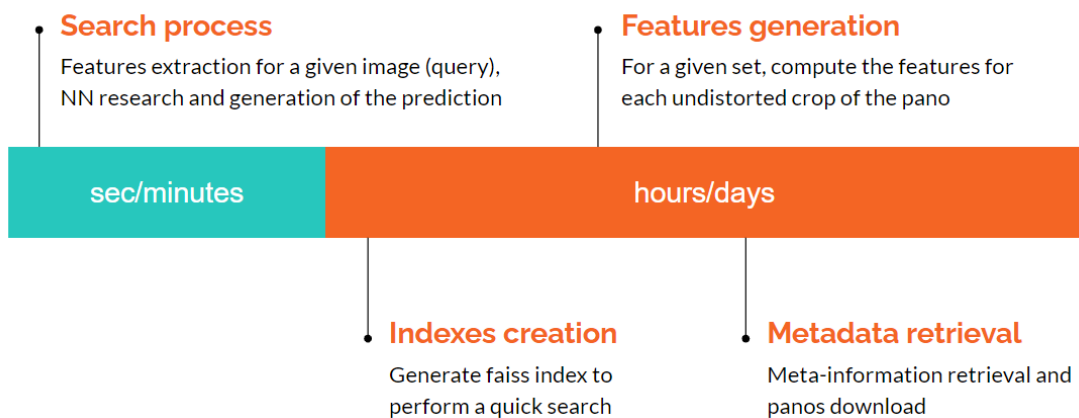


Figure 5.1. Time requirements for each step in our pipeline.

After researching and experimenting we can finally select a model. But *how can this model be deployed in order to obtain a useful and working software out of it?* Actually, an architecture can be seen as a sequence of layers and the trained model as the weights and bias that should be loaded. In this chapter we discuss the pipeline needed for the software to run, its architecture and features.

5.1 Overview

Before being able to perform a search for a query, several one-time steps are required, we have in order *metadata retrieval*, *features generation*, *index creation*. In particular, all these steps should be re-executed only in special cases:

- There is the need for a new gallery (or some samples should be added to it).
- We want to use a different model as *feature extractor*. In this scenario all the steps, except the first one, should be run again.

In regards to the first point it's worth mentioning that the current implementation allows a cumulative update of the gallery. Thus, if new images are added to include a previously uncovered location, it's possible to quickly compute the features only for the missing ones.

Now, let's describe each step in the pipeline and clarify how it works.

Metadata Retrieval At first, we start by collecting all the images and their relative metadata by calling the StreetView APIs offered by Google. This step can take up to several days for big cities (i.e. Rome) and it requires a great amount of space on disk. In particular, this step can be considered as made up of different sub-steps:

- Download all the metadata for a given territory.
- Download all the panoramas for a given location.
- Process the panoramas and generate 7 undistorted crop from each of them.
- Save the images and the metadata in a structured format to be easily accessed.

Moreover, the software module in charge of this task is highly efficient and fault-tolerant. In practice, it makes usage of multi-thread and it is able to restore its execution status if terminated early or if any error occurs (i.e. network issues).

For reference, performing this step on Turin with a *gigabit connection* and *40 concurrent thread* took *5 days* to end.

Features generation When all the images and metadata are in a suitable format, the next step is the extraction of features for all the preprocessed images in the gallery. This is done by performing inference for each image of the database and writing to disk the returned representation.

In case the descriptor is big in size like NetVLAD, a further step of *dimensionality reduction* is applied right after the extraction of features.

To further remark the importance of scalability, we should mention that applying a similar technique like PCA, requires dozens of days to be accomplished. For this reason, even if this step is performed only rarely, NetVLAD is highly penalized by this perspective. The *feature extraction* process **without PCA** took *100 hours* to be completed.

Index creation As we have already mentioned, the search process consists of finding the closer images on the features space. One way to reach such result is to perform a *k-NN* to look at the nearest *k* neighbors and predict accordingly to the results.

Unfortunately, using the *k-NN* algorithm is not efficient and would lead to unacceptable time performance and that's why we leveraged the *faiss library*. Faiss is designed around an index type that stores a set of vectors, and provides a function to search in them with L2 and/or dot product vector comparison. Some index types are simple baselines, such as exact search. Most of the available indexing structures correspond to various trade-offs with respect to search time, search quality, memory used per index vector, training time [25] [26].

We chose the *IndexFlatL2* index. It is an exact search index that encodes the vectors into fixed-size codes that compares the L2 (euclidean) distance between vectors and returns the *top-k* similar vectors. During the search, all the indexed vectors are decoded sequentially and compared to the vector whose nearest neighbours are being calculated. This vector is also called the query vector. For comparison, this is not like doing a common similarity search (i.e. with Scikit-learn), as we have to choose the measure of similarity and select the index accordingly. Furthermore, the library stores the indexes in memory or disk by using a tree data structure to hugely reduce the search time.

For reference, the generation of all the required indexes for the city of Turin took about *1 hour*.

Search process After having described all the *configuration steps*, we can finally have a look at the search process.

For each query given as input, the search process involves in:

- Pre-processing the image
- Computing the descriptor for the query
- Searching the closest neighbours in the feature domain
- Ranking the results

In regards to the first point, the image given as input is usually resized or cropped before feeding it into the net for computing its descriptor. When the descriptor is available, we can perform a fast and efficient search thanks to the previously generated *faiss index*. As result, a list of the closest gallery samples is returned. After that we can choose the ranking order. We tried out two different ideas:

- **Ranking by the closest distance** In other words, it means ranking by similarity in feature space.
- **Ranking by crop voting** A number N of crops is generated from the query. Different predictions are computed for each crop and the final predictions are obtained as the most voted gallery samples.

According to our dataset and experiments the first approach worked as well as the second, however the first should be preferred for the sake of simplicity. In regards to the time complexity, the time required to run a search for 5 different queries over the whole Italian capitals took less than *5 secs*.

5.2 Requirements

Dealing with a large scale context involves analysing the requirements needed for a software to run. In this section we want to discuss the time and memory complexity and what has been done.

First, we need to consider how much space is required to store the gallery dataset and their descriptors on disk. The size of the gallery and its stats are reported in the Appendix A 7.1.

We want to draw attention in regards to the size of the descriptors because this represents one of the most relevant bottlenecks. In particular, to perform the similarity search for a given query and to avoid the overhead introduced by reading from disk, we need to load them in RAM. That highlights the importance of a compact representation.

The size of the descriptors highly depends on the chosen architecture, as shown in the comparative Table 4.8.

On the other hand, both the CPU and the GPU don't represent a bottleneck for the application at test time, since the first is really needed only to run an efficient similarity search algorithm and the latter is used just to extract the descriptor from the query image.

5.3 Design and features

Running mode The software is characterized by 2 modalities, namely the deployment and the maintenance/update modality. The software must run in deployment in order to run searches for its users, it must be in the maintenance execution each time the *one-time steps* have to be run again. In deployment, the software can run in different modes:

- As a *python* module to be imported.
- As a script from the CLI.
- As a website for an easy remote access and a better UX.
- As an API endpoint.

To implement persistence, the software offers compatibility for both SQLite and MySQL database and the choice can be specified in the config.json file.

Technologies and dependencies The software was mostly written in python and has just a few dependencies like *faiss*, *PyTorch*, *PIL*, *SKLearn*, *h5py*. The GUI was build with *PHP/HTML/CSS + Bootstrap and LeafletJS* and was deployed on a *XAMP stack*.

Features The software has several capabilities and optimizations. For example all the faiss indexes are stored on disk but they are loaded in RAM only when they are really needed. This allows to avoid wasting too much RAM and improves the time efficiency. In regards to the authentication, the software distinguishes *admin* and *user* roles. The admin user is able to see registered users and create new ones.

Moreover, the software is capable to perform optimized searches for locations within a user-defined radius. In other words, it's possible to specify a center

point and a radius on a map and search for results only within that geographical region. That leads to two advantages: the first being the increased accuracy because of the lesser number of false-positives and the second being the increased performances because only some images in the gallery are considered as candidates (in practice, only some *faiss indexes* are used to search).

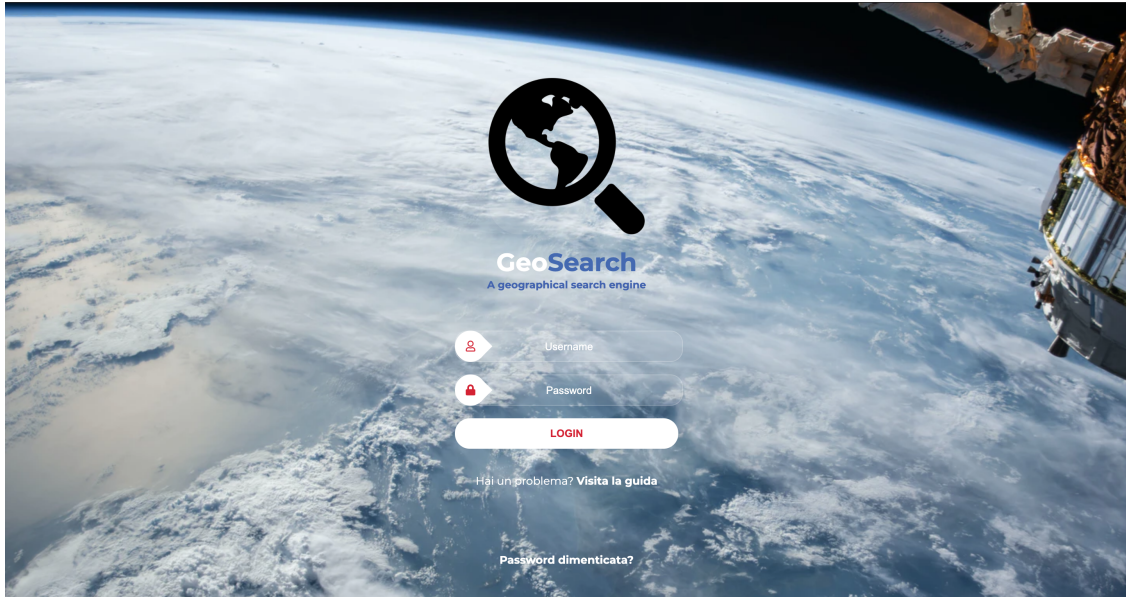


Figure 5.2. A screenshot of the GUI showing the authentication feature.

A summary with the main capabilities of the software is given as follow:

- Index lazy loading
- Memory and time efficiency
- Multi-OS support
- Scalability and modularity
- Easy maintenance
- Simple setup and configuration
- Authentication and user roles
- Logging

- Flexibility
- Support for multiple queries
- Parameters
- Geographical filter

It's worth mentioning that even if the software offers a full compatibility among all the common operative systems like *OS-X*, *Windows*, *Linux* it still needs *faiss* as dependence. Unfortunately, *faiss* authors don't offer *Windows* compatibility when using GPU acceleration and for this reason it's required to make some changes and compile it by hand.¹

5.3.1 Efficiency

Time and memory efficiency are key factors when dealing with large scale data. To improve the time efficiency we have chosen to use the *faiss* library. As defined by the creators "*faiss is a library for efficient similarity search and clustering of dense vectors. It contains algorithms that search in sets of vectors of any size, up to ones that possibly do not fit in RAM*"[25].

The library implements a fast approximation for the nearest neighbor search and is built around an index type that stores a set of vectors, and provides a function to search in them with L2 or dot product vector comparison[20].

Faiss helps a lot but still requires its index files to be stored in RAM! So, we used a lazy loading pre-initialization.

In other words, every time a new query is given as input, the software will search whether the indexes and other required structures can be read directly from RAM or if they should be read from disk.

5.4 Results and performance

The performance depends on different factors like the chosen architecture, the parameters, the number of queries, however performing a search on large scale in matters of seconds can be considered an accomplished goal.

In addition, a documentation covering the setup, the configuration and the usage of the software, both by API calls and the GUI, was produced.

¹We successfully compiled *faiss* under windows and reached to run the software with *faiss* with GPU acceleration enabled.

Furthermore, all the architectures that produce small sized features, will be faster because there is no need to compute PCA.

GUI Let's look at how the software GUI works from a user point of view. At first, the user needs to authenticate himself by using his username and password. After a successful authentication, the user is redirected to the homepage, as shown in Figure 5.3, and can navigate to the search page.



Figure 5.3. Home page of a logged in user.

The user is asked to select the images he wants to search for and is invited to set different parameters and filters as can be seen from Figure 5.4.

After having selected the search criteria, he can submit the form, the search process starts and a loading page appears. After few seconds, the results for each query are displayed in both tabular form and map, as can be seen in the page results attached in the Appendix C. When using the software via the GUI, for security reason, the user will be automatically logged off after a period of inactivity (5 min).

Furthermore, the UI development was led by the design of a good UX. Some examples are:

- The drag and drop to select the images and the ability to remove selected files if a wrong choice has been made.

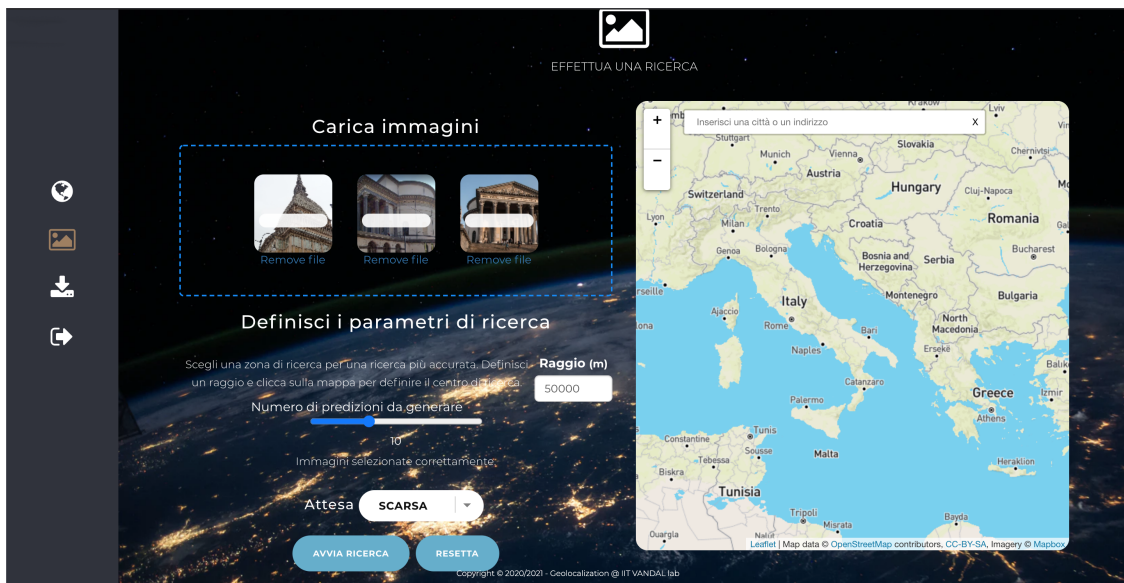


Figure 5.4. The search page. The filters and parameters available can be easily spotted.

- The search bar on the map for an easier and faster navigation.
- The loading page.

API The software also exposes an API method that can be called to perform research from remote for automation purpose. As of now, the API exposes only one public method named *uploadAndProcessSet*. In particular, it can be called for a given image and returns a *json* response with the predictions. The method has different parameters, some of which are optional. There are only four required parameters useful to specify the input images and perform authentication:

- **files**=[**binary**] Query image to be sent. The API allows this parameter to be repeated to execute a *batch search*.
- **num_preds_to_save**=[**integer**] Number of predictions to be saved on remote disk. If set to zero, the parameter *num_preds_per_query* should be specified.
- **uid**=[**integer**] A valid user ID.

- **APIKey**=[string] API Secret Key for a naive authentication.

Moreover, several other parameters can be specified to customize the search process. A list of the available optional parameters follows:

- **num_preds_per_query**=[integer] To be specified when *num_preds_to_save* is equal to zero. It represents the number of predictions that should be returned for each query.
- **precision**=[integer] This argument can be set in the range [0,5]. They are just preset parameters for the faiss search. A small number will prioritize the time at the cost of memory and, in rare cases, precision.
- **center_lat**=[float] Latitude of the central search point.
- **center_lng**=[float] Longitude of the central search point.
- **radius**=[integer] Geographical radius of search. This will be ignored if *center_lat* and *center_lng* aren't set.

If the request has been correctly formatted, a successful response will be returned. A successful response, as shown in the Appendix B 7.2, contains information about the input queries and an array containing the predictions for each query. A successful response contains, for each prediction of every query, the following fields:

- **city** The predicted city.
- **confidence** The confidence of the result in the range [0, 100].
- **cropped_index** The index of the crop in the range [0,6] related to the portion of panorama for which a match was found.
- **date** The date when the panorama was taken.
- **knn_d** The distance from the similar image in the gallery returned by faiss.
- **lat** The predicted latitude.
- **lng** The predicted longitude.
- **metro** The nearest metro stop.
- **pano_id** A unique ID identifying each image in the gallery.
- **pano_path** The URI pointing at the predicted image in the gallery.

5.5 Improvements

The software could be further improved on different fronts such as security and accuracy. In regards to security the static API Key or the password based authentication could be replaced with a JWT (JSON Web Token) based system. In addition, it could be interesting to propose an auto-update feature to autonomously update the gallery to include fresh panoramas. In regards to accuracy, there is always some margin of improvements and many different ideas could be considered:

- Try other recent backbones as ResNet-RS[34] or EfficientNet v2[35]
- Usage of Data Augmentation
- Further study on Domain Adaptation
- Attention based methods or other solution to handle bigger occlusions.
- Post processing techniques (i.e. *post-pruning* some unrelated results like a photo of a truck or taken in suburban areas)

In this regard, the system is more prone to bad predictions when a big occlusion occurs or under extreme lighting conditions (i.e. night). You can look at some predictions of the models in the Appendix D 7.4. Other interesting features could be adding some more administrative capabilities such as showing the last searches performed by user or editing the password of a given user or invalidating its *API key*.

The development dataset could also be improved by pruning all the indoors panoramas or suburbs zone.

Chapter 6

Conclusion

The goal of this thesis was to investigate efficient and scalable visual place recognition architectures and to develop a software to accomplish this task. At first, we described the main challenges related to high scalable VPR systems and we analyzed what techniques have been proposed over the years. Then, we demonstrated that many architectures can be employed in order to develop such a system.

We included many experiments useful to perform comparisons between the different solutions and described their strengths and weaknesses. The experiments highlighted the key role of the *aggregation layer*, particularly in regards to the scalability, and revealed what the main edge cases and limitations for similar networks are. Furthermore, we proved that other aggregation layers could be researched and employed and we also confirmed that tackling VPR as a classification task is a suitable approach and can lead to SotA performance. In the process, we tried to justify our choices and to suggest which improvements and limitations apply. We also showed that different domains are a real challenge for geo-localization networks and reported the results after experimenting with several domain adaptation techniques. In this regard, we noticed that, even if simple techniques can help in achieving better performance on the target set, they aren't enough to improve the overall generalization capabilities of the network.

Moreover, we discussed some implementation details about the developed software. We defined its architecture, its components and the available features. To the best of our knowledge, this is the first work to tackle VPR from such a large scale perspective and to describe its overall design, its key components and its pipeline.

In conclusion, this work has successfully achieved the previously defined

goals and wants to be an incentive for further research in this area.

Chapter 7

Appendix

7.1 Appendix A: Development Gallery

City	Approx. Lat,Lng	Size On Disk	#Panoramas
Roma	(41, 12), (42,12)	1,1T ; 36G	3624126
Napoli	(40, 14)	268G	955844
Torino	(45, 07)	383G	1205070
Milano	(45, 09)	515G	1719752
Bologna	(44, 11)	162G	529306
Firenze	(43, 11)	138G	462626
Palermo	(38, 13)	161G	526655
Venezia	(45, 12)	118G	416548
Bari	(41, 16), (41, 17)	111G	355015
Perugia	(43, 12)	105G	345092
Cagliari	(39, 9)	73G	287902
Ancona	(43, 13)	62G	207469
Trento	(45, 11), (46, 11)	61G	199374
Trieste	(45, 13)	68G	202116
L aquila	(42, 13)	59G	196141
Potenza	(40, 15)	48G	117601
Genova	(44, 8), (44, 09)	18G	102629
Catanzaro	(38, 16)	28G	109573
Campobasso	(41, 14)	23G	77330
Aosta	(45, 7)	11G	36160

7.2 Appendix B: API Example

Example of an API request with curl:

Listing 7.1. API call example

```
curl --location --request \
POST 'https://localhost/api/v1/searchengine/search/
uploadAndProcessSet?center_lat=&center_lng=&radius=50000
&precision=0&uid=3&tot_q=7&num_predictions=10?apiKey=' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--form 'file[]=@/path/to/file'
```


An example of a response:

Listing 7.2. API Response

```
[
  {
    "predictions": [
      {
        "pano_path": "p1.jpg",
        "cropped_index": 5,
        "url": "predictions/3_2021-02-02_11-04-17/p1@.jpg",
        "lat": "45.06864",
        "lng": "07.69252",
        "pano_id": "1234",
        "year": "2016-09",
        "date": "2016-09",
        "knn_d": "55.364",
        "confidence": 36,
        "metro": "Porta Nuova a 1.25 Km"
      },
      {
        "pano_path": "p2.jpg",
        "cropped_index": 5,
        "url": "p2.jpg",
        "lat": "45.06872",
        "lng": "07.69259",
        "pano_id": "1235",
        "year": "2016-09",
        "date": "2016-09",
        "knn_d": "56.551",
```


```
    "confidence": 35,
    "metro": "Porta Nuova a 1.26 Km"
  },
  ...
  ,
  {
    "pano_path": "p10@.jpg",
    "cropped_index": 5,
    "url": "predictions/3_2021-02-02_11-04-17/p10@.jpg",
    "lat": "45.06889",
    "lng": "07.69465",
    "pano_id": "5415",
    "year": "2018-09",
    "date": "2018-09",
    "knn_d": "62.989",
    "confidence": 31,
    "metro": "Porta Nuova a 1.41 Km"
  }
],
"query": {
  "url": "queries/3_2021-02-02_11-04-17/moleq.jpg",
  "id": 0
}
}
```


7.3 Appendix C: GUI results page













RISULTATI GEOLOCALIZZAZIONE

La tua immagine



Risultati

URL					
CONFIDENCE	51	49	49	46	46
MAPS	StreetView	StreetView	StreetView	StreetView	StreetView
COMUNE	Venezia	Venezia	Venezia	Venezia	Venezia
METRO	Sant'Eufemia - Buffalora a 206.01 Km	Sant'Eufemia - Buffalora a 205.97 Km	Sant'Eufemia - Buffalora a 205.99 Km	Sant'Eufemia - Buffalora a 206.03 Km	Sant'Eufemia - Buffalora a 206.02 Km
LAT	45.43426	45.43442	45.43425	45.43416	45.43436
LNG	12.33878	12.33848	12.33867	12.33897	12.33897
DATE	2013-07	2013-04	2013-04	2017-10	2013-07
URL					
CONFIDENCE	46	44	44	43	43
MAPS	StreetView	StreetView	StreetView	StreetView	StreetView
COMUNE	Venezia	Venezia	Venezia	Venezia	Venezia
METRO	Sant'Eufemia - Buffalora a 206.01 Km	Sant'Eufemia - Buffalora a 206.02 Km	Sant'Eufemia - Buffalora a 206.03 Km	Sant'Eufemia - Buffalora a 205.97 Km	Sant'Eufemia - Buffalora a 206.02 Km
LAT	45.43431	45.43437	45.43401	45.43442	45.43436
LNG	12.33882	12.33897	12.33899	12.33848	12.33896
DATE	2013-07	2013-05	2013-05	2013-04	2013-04

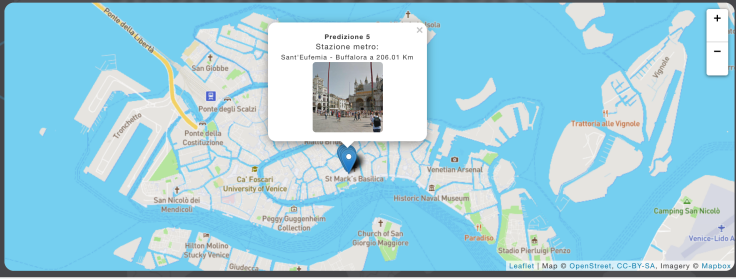


Figure 7.1. Results page relative to a search with a query and *NumPredictions=10*.

7.4 Appendix D: Results example

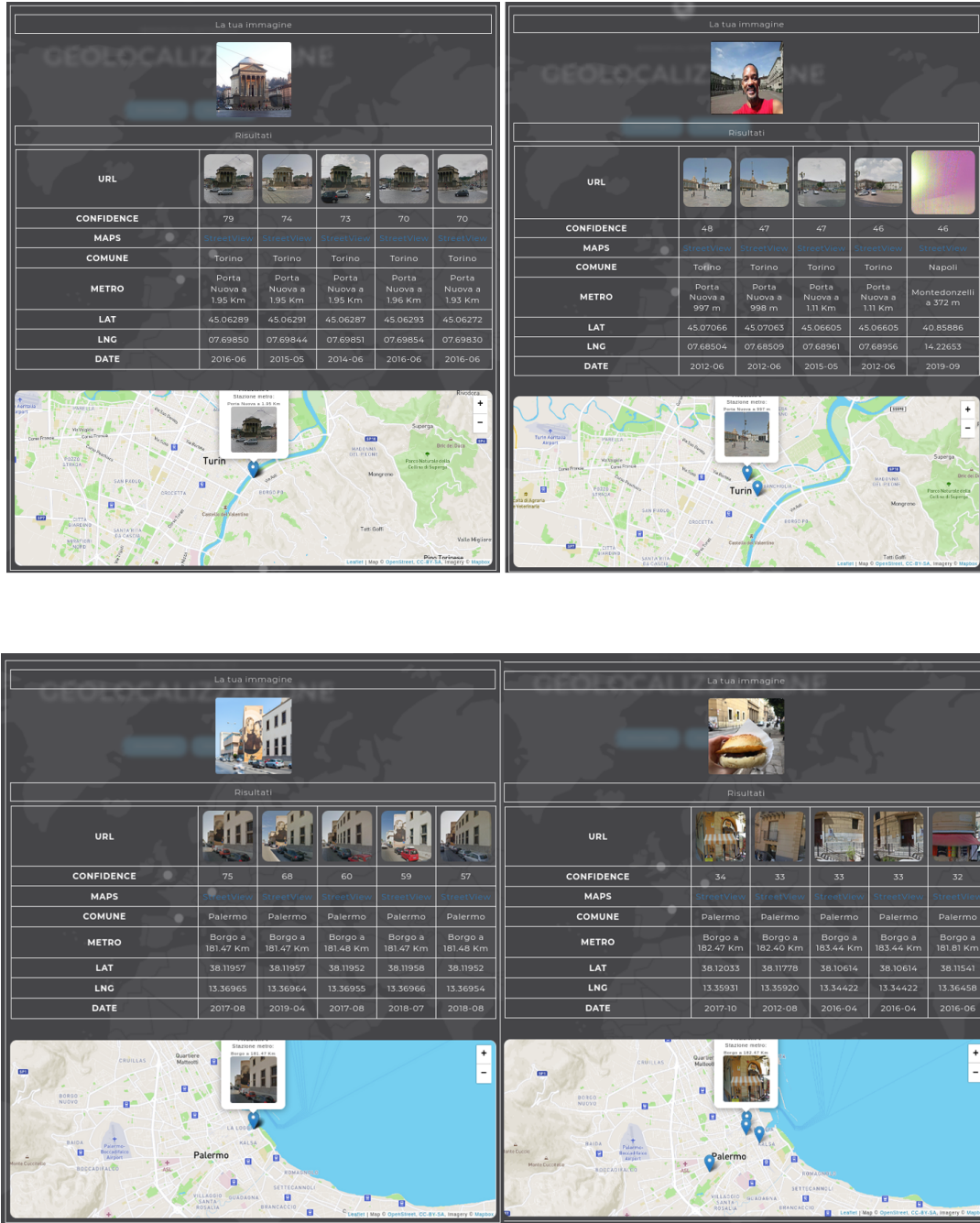


Figure 7.2. On the top two search results in Turin. On the bottom the results relative to a search in Palermo; At the bottom right corner a wrong prediction due to a big occlusion.

Bibliography

- [1] Sivic, J. and Zisserman, A., *A Text Retrieval Approach to Object Matching in Videos*, in Proceedings of the International Conference on Computer Vision, 2003
- [2] S. Lowry, N. Sunderhauf, P. Newman, J. J. Leonard, D. Cox, P. Corke, and M. J. Milford, *Visual place recognition: A survey*, IEEE Transactions on Robotics, vol. 32, no. 1, pp. 1–19, 2015.
- [3] H. Wang, C. Wang, and L. Xie, *Intensity scan context: Coding intensity and geometry relations for loop closure detection*, in IEEE international conference on robotics and automation (ICRA), 2020.
- [4] H. Jégou, F. Perronnin, M. Douze, J. Sánchez, P. Pérez and C. Schmid, *Aggregating Local Image Descriptors into Compact Codes*, in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 34, no. 9, pp. 1704-1716, Sept. 2012. doi: 10.1109/TPAMI.2011.235
- [5] Relja Arandjelović, Petr Gronat, Akihiko Torii, Tomas Pajdla and Josef Sivic. NetVLAD: CNN architecture for weakly supervised place recognition, 2015; arXiv:1511.07247.
- [6] Filip Radenović, Giorgos Tolias and Ondřej Chum. *Fine-tuning CNN Image Retrieval with No Human Annotation*, 2017; arXiv:1711.02512.
- [7] Weyand, Tobias and Kostrikov, Ilya and Philbin, James, *PlaNet - Photo Geolocation with Convolutional Neural Networks*, 2016; in Vol 9912. pp 37-55. 10.1007/978-3-319-46484-8_3. .
- [8] Gabriele Moreno Berton, Valerio Paolicelli, Carlo Masone and Barbara Caputo. *Adaptive-Attentive Geolocalization from few queries: a hybrid approach*, 2020, Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), 2021, pp. 2918-2927; arXiv:2010.06897. DOI: 10.1109/WACV48630.2021.00296.
- [9] James Philbin and Ondrej Chum and Michael Isard and Josef Sivic and Andrew Zisserman. *Lost in Quantization: Improving Particular Object Retrieval in Large Scale Image Databases*, 2008 IEEE Conference on

Computer Vision and Pattern Recognition

- [10] R. Arandjelovic and A. Zisserman. *All about VLAD*, In Proc. CVPR, 2013
- [11] Giorgos Tolias, Ronan Sifre and Hervé Jégou. *Particular object retrieval with integral max-pooling of CNN activations*, 2015; arXiv:1511.05879.
- [12] Praveen Kulkarni, Joaquin Zepeda, Frederic Jurie, Patrick Perez and Louis Chevallier. *Hybrid multi-layer Deep CNN/Aggregator feature for image classification*, 2015; arXiv:1503.04065.
- [13] Babenko et al. ECCV, 2014;
- [14] Hanjiang Hu, Zhijian Qiao, Ming Cheng, Zhe Liu and Hesheng Wang. *DASGIL: Domain Adaptation for Semantic and Geometric-aware Image-based Localization*, 2020; arXiv:2010.00573. DOI: 10.1109/TIP.2020.3043875.
- [15] A. Torii, J. Sivic, T. Pajdla and M. Okutomi, *Visual Place Recognition with Repetitive Structures*, 2013; IEEE Conference on Computer Vision and Pattern Recognition, Portland, OR, USA, 2013, pp. 883-890, DOI: 10.1109/CVPR.2013.119.
- [16] P. Gronát, G. Obozinski, J. Sivic and T. Pajdla, *Learning and Calibrating Per-Location Classifiers for Visual Place Recognition* IEEE Conference on Computer Vision and Pattern Recognition, Portland, OR, USA, 2013, pp. 907-914, DOI: 10.1109/CVPR.2013.122.
- [17] HeeJae Jun, Byungsoo Ko, Youngjoon Kim, Insik Kim and Jongtaek Kim. *Combination of Multiple Global Descriptors for Image Retrieval*, 2019; arXiv:1903.10663.
- [18] Bingyi Cao, Andre Araujo and Jack Sim. *Unifying Deep Local and Global Features for Image Search*, 2020; IEEE Access, Vol.9, pp 19516-19547 DOI: 10.1109/ACCESS.2021.3054937.
- [19] C. Masone and B. Caputo *A Survey on Deep Visual Place Recognition*, 2021 arXiv:2001.05027.
- [20] Jeff Johnson, Matthijs Douze and Hervé Jégou. *Billion-scale similarity search with GPUs*, 2017; arXiv:1702.08734.
- [21] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. *Deep Residual Learning for Image Recognition*, 2015; arXiv:1512.03385.
- [22] Mingxing Tan and Quoc V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*, 2019, International Conference on Machine Learning, 2019; arXiv:1905.11946.
- [23] Chen-Yu Lee, Patrick W. Gallagher and Zhuowen Tu. *Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree*, 2015; arXiv:1509.08985.

- [24] Deng, J. and Dong, W. and Socher, R. and Li, L.-J. and Li, K. and Fei-Fei, L., *ImageNet: A Large-Scale Hierarchical Image Database*, CVPR09, 2009
- [25] <https://github.com/facebookresearch/faiss>
- [26] Jeff Johnson, Matthijs Douze and Hervé Jégou. *Billion-scale similarity search with GPUs*, 2017; arXiv:1702.08734.
- [27] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*, 2015; arXiv:1502.01852.
- [28] Vassileios Balntas, Edgar Riba, Daniel Ponsa and Krystian Mikolajczyk. *Learning local feature descriptors with triplets and shallow convolutional neural networks*, BMVC, 2016; isbn:1-901725-59-6.
- [29] Yanchao Yang and Stefano Soatto. FDA: Fourier Domain Adaptation for Semantic Segmentation, 2020; arXiv:2004.05498.
- [30] Taesung Park, Alexei A. Efros, Richard Zhang and Jun-Yan Zhu. Contrastive Learning for Unpaired Image-to-Image Translation, 2020; arXiv:2007.15651.
- [31] Y. Choi and N. Kim and S. Hwang and K. Park and J. S. Yoon and K. An and I. S. Kweon, *KAIST Multi-Spectral Day/Night Data Set for Autonomous and Assisted Driving* IEEE Transactions on Intelligent Transportation Systems, 2018, Vol 19, 934-948, DOI: 10.1109/TITS.2018.2791533.
- [32] Will Maddern and Geoff Pascoe and Chris Linegar and Paul Newman, *1 Year, 1000km: The Oxford RobotCar Dataset* The International Journal of Robotics Research (IJRR), Vol 36 Num 1, pp. 3-15, 2017 DOI: 10.1177/0278364916679498.
- [33] Yaroslav Ganin and Victor Lempitsky. Unsupervised Domain Adaptation by Backpropagation, 2014; arXiv:1409.7495.
- [34] Irwan Bello, William Fedus, Xianzhi Du, Ekin D. Cubuk, Aravind Srinivas, Tsung-Yi Lin, Jonathon Shlens and Barret Zoph. Revisiting ResNets: Improved Training and Scaling Strategies, 2021; arXiv:2103.07579.
- [35] Mingxing Tan and Quoc V. Le. EfficientNetV2: Smaller Models and Faster Training, 2021; arXiv:2104.00298.