POLITECNICO DI TORINO

Master degree in Computer engineering - Data science

Master Thesis

Deep learning for visual place recognition

Large scale software and self-supervised approach to geo-localize a given photo

Supervisors
prof. Barbara Caputo

Valerio Paolicelli
matricola: 253054

Academic year 2019 – 2020

Deep learning for visual place recognition
Master thesis. Politecnico di Torino, Turin.

April 2020.

Acknowledgements

Come succede per la conclusione di ogni percorso, anche per quanto riguarda la mia tesi, la fine dei miei anni di studio, ho il piacere, ma anche il dovere, di ringraziare tutti coloro che mi sono stati vicini, ognuno prezioso per la mia crescita come uomo, che hanno dedicato parte del loro tempo al mio fianco, lasciandomi ogni volta un grande bagaglio di riflessione per maturare e migliorare.

Alla professoressa Barbara Caputo, che ha creduto nelle mie capacità e nella mia persona, concedendomi l'opportunità di lavorare con lei e il suo gruppo di ricerca. Per gli innumerevoli consigli tecnici e non, dei quali farò sempre tesoro.
A Gabriele, compagno di viaggio nello sviluppo di questo progetto, e a tutto il gruppo "VANDAL". Nuovi colleghi con i quali ho affrontato questa esperienza; grato per ogni singolo confronto, aiuto e riflessione fatta in questi mesi.

A mia madre e mio padre il grazie più grande e la dedica di questo lavoro, per aver sempre voluto il meglio per me, disposti a subire anche la distanza che ci ha separato in questi anni, per il solo generoso obiettivo di vedermi realizzato. I loro insegnamenti e incoraggiamenti, sono stati il motore della mia crescita e fanno la persona che sono oggi. A loro un grande ringraziamento, speranzoso di averli resi fieri di me durante tutto il percorso.

A mio fratello, punto di riferimento e ammirazione durante questi anni di università, sempre pronto ad ascoltarmi e ad aiutarmi. Sicuramente questi mesi di lavoro senza di lui, non sarebbero stati gli stessi.

A tutti i parenti e amici di famiglia, che ad ogni mio ritorno tra le frasi "Quando sei arrivato? ...e quando te ne rivai?" hanno dimostrato di essere felici di riabbracciarmi e tenaci nell'incoraggiare le mie scelte di studio. A

mio cugino Giuseppe, punto di riferimento per questo settore dai miei primi anni di studio. A chi non c'è più, che non ha potuto vedere la mia crescita professionale, ma che sicuramente mi avrebbero guardato con grande orgoglio e soddisfazione.

A Rossella, nata coinquilina e trasformata in sorella in questi anni, presenza quotidiana di ogni esperienza di vita in questi lunghi anni torinesi. Compagna di mille trascorsi, tra risate e qualche momento difficile, capace di comprendermi sempre, supportarmi in ogni mia scelta e consigliarmi di fronte ai dubbi che mi mettevano in difficoltà.

A tutti gli amici di questi anni universitari, a quelli partiti con me da Matera, quelli conosciuti tra le aule e i corridoi del Politecnico e tutti coloro che per una qualsiasi coincidenza ho incontrato nel mio percorso lontano da casa e sono rimasti, nonostante tutto, al mio fianco. Per tutti i pomeriggi passati a studiare, a scambiarsi conoscenze per la sola voglia di aiutare l'altro, per tutte le serate passate insieme, le risate, i brindisi, le carbonare, le esultanze di fronte ad una conquista, un esame andato bene o semplicemente un goal.

A tutti gli amici de "Motel crew", che nonostante la distanza, mi hanno sempre accolto a braccia aperte al mio ritorno, mantenendo solido il nostro rapporto fraterno, capace di superare ogni vicissitudine della vita, per quanto questa può aver allontanato le nostre strade.

Se oggi sono qui, è anche merito vostro, di tutto quello che ognuno di voi mi ha fatto provare e imparare. Grazie.

"La fine di un percorso, non è che l'inizio di una nuova avventura."

Valerio Paolicelli - 3 Aprile 2020

Abstract

An open problem in the artificial intelligence community is building an algorithm able to geo-localize a given photo, overcoming the multiple problems related to the domain shift between the images used during the training and the ones passed at test time.
Our thesis is a combination of research and development contribution. To approach the visual place recognition (VPR) problems with a deep learning method, we have used the current state-of-the-art convolutional neural network (CNN) called NetVLAD [1]. We have properly modified it in order to speed up the process and to study the results on our dataset. In particular, we have changed the backbone and trained the entire network on our dataset with a self-supervised approach, to make the network more confident with the different domains belonging to the training and testing phases. Moreover, we have also partially investigated the effect of some artificial occlusions applied on the image that needs to be geo-localized, seeing if they can be useful to focus the network attention on the relevant object inside a photo, instead of the dynamic ones. Finally, two kinds of software are developed, the first one is composed by a set of steps, to download and create all the necessary things related to the dataset that a user wants to create, while the second one is a graphical user interface, used to upload a photo and visualize the network results.

Contents

Chapter 1

Introduction

The landscape: In recent years, the research in the artificial intelligence environment has reached huge improvements, making systems more intelligent and capable of performing actions always more similar to what a human can do.
The best results, in terms of accuracy, have been obtained through the implementations of Deep Learning algorithms, which are able to receive in input a set of images and directly analyze the pixels contained in them thanks to the application of convolutional neural networks (CNN). So, CNNs extract more useful information and they are able to perform multiple tasks than compared to the more traditional machine learning algorithms.
This is the reason why currently in the Computer Vision community the CNNs find applications in a variety of fields, such as robotics, medical decisions and autonomous driving. The networks, in turn, have changed much since their first developments, they are richer in layers, thus reaching a higher level of abstraction.
At the beginning, their role was only the ability to correctly solve a classification task, such as understanding whether an image contains a dog or a cat, or predict if an extract of histological cancer tissue is probable or not.
After further developments, the community also deployed networks that can localize the objects in a photo, as for the object detection task, or systems that are used only to extract the important and more abstract features, from the original pixels set. For this reason, CNNs have easily found space in the image retrieval field.
The image retrieval has the goal to recognize the content inside an image, called query, and to find its best match in a large-scale database that is labelled and filled of various metadata. This process can be summarized as

follow:

1. Receive a big database of images and for each of theme:
    - Extract the most useful features, related to the development field (e.g. for example for face recognition these features can be the eyes, mouth and nose position, face shapes and so on);
    - Save an aggregated representation of these features;
2. Receive a query image:
    - Extract the most useful features and build the aggregate representation also for each of them;
    - Through some optimized research algorithm, as the KNN, finds the most relevant and similar match between the query features representation and the ones inside the database.

Furthermore, it becomes really important the sets of features extracted by the input images. Nowadays the CNNs are used in order to perform the best feature extraction as possible. They assume the role of encoder instead of a standard classifier, extracting the descriptors from a certain convolutional layer. Another key point in the whole retrieval process, especially for the real-time systems, is the possibility of applying filters on the search area based on parameters known a priori, such as somethings suggested by a user. In this way, it makes an effort to minimize the possible solutions space. Thus, the time needed to obtain the results is greatly reduced.
Overall, the instance retrieval process can be applied in a lot of different fields, such as recognition of faces, car models, a fashion brand e-commerce and also the outdoor geo-localization.

Visual place recognition: The visual place recognition problem (VPR) is really challenging and currently it is an open problem in the robotics and computer vision community. The goal of VPR is to build a system able to recognize the exact location of a given outdoor photo. This system should respect different constraints, related to the amount of time needed and the efficiency in retrieving the GPS coordinates of a given street-corner compared with large-scale dataset, that ideally can contain the entire city/country. Although, the huge improvements of the recent years, the main challenge remains how to represent images of places containing buildings, landmarks etc. in order to distinguish them when they seem to be very similar.

In the most recent years, the visual place recognition field is approached as an image retrieval problem, that appears to be easier and more accurate rather than trying to solve the issue with the image classification approach. Therefore, the main effort is to associate the query image location to one of the most visual similar places contained in the database used for the retrieval task. Moreover, the image retrieval process is mainly implemented with a convolutional neural network, which is used for the extraction feature.
Furthermore, the problems of VPR are also related to changes in appearance between the query and database images, such as different lighting conditions. These changes produce a domain shift, that should be overcomed to develop a solid and efficient system. We can summarize these problems as:

- Different point of view: the images inside the database are captured from a certain point of view, which is more or less the same one for the entire database domain. On the other hand, the queries set, i.e. all the images that we want to geo-localize from the database, can be composed by photos that may have been taken from another point of view, maybe with some environment details that are not in the foreground.
- Lighting conditions: in the same way as the point of view, the database set may contain always the same lighting condition as daytime, while the queries set can be composed of photos taken at nighttime.
- Dynamic objects: this is one of the most relevant issues because the algorithm should focus on the most important features of the images, so it should ignore dynamic objects like cars and people. So during the extraction of the descriptors from the queries, the algorithm should not really take into account the ones of the dynamic object to perform the image retrieval.
- Long-term conditions: the scenarios continuously evolve during the year, the environment suffers from many changes due to climate and seasonal changes. For example, we can easily picture how much a road with many trees changes during the year. Therefore, the query set taken during the winter should be recognized as the same compared with a database set collected during the summer.

In addition, this research area is becoming increasingly interesting, thanks to the progress made in fields such as augmented reality or autonomous driving. In the last case, the technology should use the VPR results to geo-localize vehicle position, when the GPS system is offline. Another interesting field is given by applications that want to geo-localize on outdoor archival imagery.

1.1 Thesis contribution

Focusing on image geo-localization, the thesis objectives are related to a software system, that should be used by inexpert users and its main goal is to be robust for the environment changes, to be fast in producing results and more accurate as possible on large-scale situations. Our work concerns different implementations. We want to approach the VPR task through some deep learning methods. Before starting with the algorithms, we have developed two different softwares to collect data and run the algorithms. Then, the state-of-the-art (SOTA) network for VPR is downloaded and tested on our large-scale dataset. From this point on we have tried to improve the accuracy of that CNN, modifying the architecture and testing some new approaches (Section 4).
The first software developed is composed by a set of steps that allow the download of all the needed data in order to create a dataset of photos taken by Google Street View. These steps mainly consist in downloading the metadata about the images in a certain region, downloading the images and storing them in a MySQL database. Through these codes, all the images of Google Street View in Turin have been downloaded and stored creating the Turin1M dataset. Starting from this huge dataset, we have created three subsets related to three geographically-disjoint zones. It is called Turin30k and it is used to train and test our deep learning algorithms. Furthermore, a new set of images is created, containing photos of Turin taken with our smartphone, so belonging to a third-domain w.r.t. Turin1M and Turin30k, it is called Turin81. More details about this software and datasets are reported in Section 3, where we have also introduced some other subsets that are built to perform some analysis (Turin81_occluded) and to solve some problems with Turin1M (Turin_undistorted). The last steps of this pipeline also allow to execute the test phase of our CNN, performing some operations offline, in order to reduce the big amount of time needed for large dataset such as Turin1M. The implementation and results with the deep learning methods are shown in the section related to the experiments 4.3.
The second software instead is an application with a web-like graphical user interface, that allows the user to upload an image, set some filters to speed up the process and improve the final accuracy and visualize the results produced by our CNN, also providing the user with a map representation (Section 3.7).
Finally, the thesis is also research oriented and for this reason we have started the SOTA for VPR, called NetVLAD [1]. First, we have downloaded a PyTorch version [2] and tested it on our dataset (Section 4.3) to have our

baseline. Then, we have changed the backbone from the VGG16 [3] to the ResNet18 [4] to speed up the prediction time, re-training on the same dataset of the previous version to allow a comparison. We have noticed a huge amount of test time saved thanks to the new backbone (Section 4.5). Moreover, this new architecture is trained on our dataset to build a model robust for the images and the domain downloaded related to Turin1M. We have demonstrated an improvements in performance when training with the Turin source domain and, as consequence, we executed the user-friendly software with higher accuracy and faster geo-localization time (Section 4.6).
In addition, to improve the accuracy of the third domain dataset (Turin81), since it is composed by few images that cannot be used to train the VPR branch, we have used a domain adaptation approach by adding a self-supervised task to the architecture (Section 4.7.2), in order to reduce the gap across the different domains. The results improve of few points, suggesting us that the mobile phone domain is not so different than Google Street View one, considering also that all these images contain pretty the same environment conditions.

Chapter 2

Related Works

2.1 The landscape

Research on visual place recognition has been steadily growing in the last decade. This increase in interest is due to the creation of large geo-localized images datasets, the easiness in acquiring new data (e.g. through camera on smart phone) and the limitations of localization and orientation systems (e.g. GPS signal worsens in urban cluttered environments), which intrinsically have a high degree of approximation. Moreover, an increasing number of present-day practical applications would rely on a ideally perfect visual based localization system, such as 3D reconstruction, consumer photography -"Where did I take these photos?"-, augmented reality, and outdoor or indoor navigation systems, which are essential for self-driving cars and robotics. There is no standardized designation for visual place recognition, methods name vary from one paper to another. The most common ones are visual place recognition, visual based localization, structure-based localization, visual geo-localization, image-based pose estimation, and a number of rearrangements of these terms. Most of current approaches for VPR rely on image retrieval, while others try to exploit 3D methods, and some even view VPR as a classification task.

2.2 Image retrieval

An image retrieval system is a system that, given a dataset of images, called database or gallery, and a single other image, called query, is able to retrieve the images in the database that are most similar to the query. Image retrieval

can have lots of practical applications, one of the most common being face retrieval. In this task, the database is made of images of faces, usually annotated with the person's full name. Then, when a new image of a face is given to the system, this has to be able to understand which person the new image belongs to, by searching the most similar face in the database. This task is similar to the VPR task, where the database is made of images of places, and the queries can be new images uploaded by a user. In the vast majority of cases, an image retrieval system is based on 4 steps:

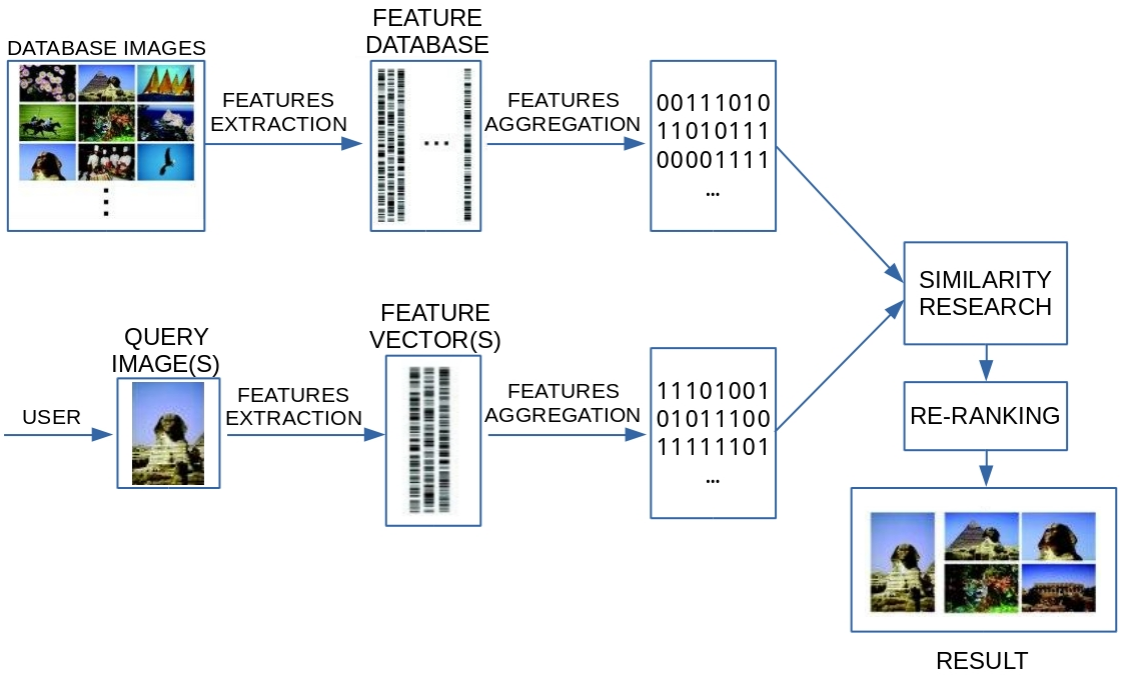


Figure 2.1: Image retrieval

- features extraction, which is the extraction of the descriptors that give meaningful information about the image;
- features aggregation, which is the rearrangement of features, preparing them for the next step;
- similarity research, which is the algorithm that takes the features in input and outputs the likelihood of their images representing the same place;

- candidates re-ranking, after having found a restricted number of potential candidates, re-ranking them from the most likely positive.

2.2.1 Features extraction

The task of extracting the best descriptors to represent an image has been approached in a huge variety of methods. The goal is to have features that incorporate the greatest amount of discriminant information, possibly without requiring too much memory, and in a fast and light way. The types of features used belong to one of these 3 groups: local, global, hybrid.

2.2.1.1 Local features

In the pre-deep-learning era, local features were without a doubt the most used in visual place recognition and computer vision in general. Their description occur at a small level, usually just a pixel and its local neighborhood. Its extraction is based on two steps: finding a salient area, and extracting its descriptors. The most famous features-extractor of this kind is the scale-invariant feature transform (SIFT) [5], published and patented in 1999. Various alternatives have been proposed during the years, mostly to speed up the computation, such as SURF [6], which are required when the computation needs to be real-time. Another more recent evolution of SIFT is RootSIFT [7], which creates better descriptors than SIFT while reducing the computational requirements

2.2.1.2 Global features

Global features consider the image as a whole, and compute a vector as its representation. Perhaps the most naive example of global features would be to just use the raw image itself, possibly after resizing. Global features are usually less robust to changes in point-of-view and local changes (such as occlusions), but are usually much less computationally hungry than local features. In this group there are both hand-crafted methods, such as GIST [8], and more recent learned methods. Among the learned methods, the most widely used for images are convolutional neural networks [1, 9], which can be used as a features extractor for a given image. Usually the features are the output from one or more convolutional layer. The CNN can either be trained on a different task, like classification, or directly trained for the image retrieval task.

2.2.1.3 Hybrid features

By hybrid features we refer to those types of features that don't belong to the previous two groups, and that either consider just a part of the image, or that combine multiple types of features. One example is represented by patch features, which, during the extraction, consider only a patch of the image at a time. The patch can be chosen in various ways, either based on the content (e.g. on the image saliency [10]) or in a standardized fashion, like through a fix grid or a sliding window [11]. On the other hand, combined features use a combination of local and global features to build the final descriptors of the image. One example is presented by [12], which uses global features to restrict the number of potential positives, and local features to compute the final output

2.2.2 Features aggregation

Features aggregation is the task of aggregating features in a more convenient way. These is because the features, especially if local, can have huge dimension. Moreover, in visual place recognition, aggregation can be performed in a way to benefit the retrieval process, for example by enhancing features that are known to be useful for the task.

2.2.2.1 Feature to visual word assignment

This technique comes from the more famous bag of words (BoW) used in natural language processing, where a vector is built counting the occurrences of each word. However, in computer vision there are no such things as words, so it was common to group the features in clusters, or Voronoi cells, and consider only the center of the clusters as features [13]. In this way, a vector with the same length as the number of clusters can be built, with the counts of features belonging to each cluster. This creation of a visual vocabulary is known as bag of visual words or bag of features. However, this hard assignment of each feature to its cluster can worsen the representation of the features, and numerous ways to replace this with a "softer" assignment have been thought. An example is hamming embedding, by Jegou et al. [14], which further divides each cluster for a more precise assignment of every feature. Another subsequent work by Jegou et al. is Vector of Locally Aggregated Descriptors (VLAD) [15], in which also the distance between the feature and the center of its cluster is saved, giving a more precise representation.

2.2.2.2 Weighting scheme

Weighting consists in assigning a weight to each features, usually by giving higher weights to more discriminative features. The weights can be computed by taking into account the frequency of a certain feature in the dataset, as in [13]. Other works [16] propose to assign the weight according to their intra and inter-burstiness, which is the likelihood that a feature is repeated more than once in an image and in the dataset. Other techniques [17] simply propose to remove the least discriminative features, saving memory and reducing noise.

2.2.2.3 Multiple features aggregation

Features obtained with neural networks can be used together with local or patch features. This way its possible to gather multiple types of features into a single vector, like a bag of features. Mixing local and neural features has been done in [18], and [19] has replace local with patch features.

2.2.2.4 Pooling of deep feature maps

With the recent advances of deep learning, it has become more common to extract features with CNNs. The features can be extracted by the last convolutional layers, and can be concatenated together. To reduce the high dimensionality several types of pooling have been applied. Maximum Activatin of Convolutions [20] proposes to reduce the dimensionality by aggregating each maximum of the activation maps into unidimensional vectors. Sum-Pooled Convolutional features [21] gives better results, simply summing the responses of each maps instead of finding their maximum. More recently Arandjelovic et al. [1] propose NetVLAD, a fully differentiable layer that can be plugged into a CNN and be treated like another layer of the network. This layer has trainable weights, which make it the state-of-the-art option at the moment. This later propose to mimic the architecture of NetVLAD in a deep-learning fashion, therefore calculating clusters and distances from them for each feature.

2.2.3 Similarity Research

The most common way to find the most similar vector of features to a given one (e.g. computed from a query) is by finding the closest to it by euclidean distance (usually with L2 norm) or cosine similarity. This is simple to do

as long as the dimensions don't grow too much, both in terms of number of descriptors (one for each image) and of size of each descriptors. In the latter case, a dimensionality reduction is often needed, and PCA is a common technique [1, 9]. To improve results, whitening can also be applied, as in [1].

2.2.3.1 K-nearest neighbors

Brute force KNN can be used in those cases where the datasets are not too large, and computation can be done in a reasonable time. However, if the size of the dataset grows, so does the number of descriptors, and having an exact response can take too long to be computed. In some cases, approximate k-nearest neighbors can be a better solution, in which a little loss in accuracy can give a huge reduction of computation time [22]. An example of a library which implements approximate nearest neighbors search is Faiss [23], developed by the Facebook AI Research team.

2.2.3.2 Machine Learning methods

Other methods have been employed which focus on better understanding the distribution of the features, and exploit this knowledge to improve the accuracy of the retrieval task. An example is given by [24], where the similarity search is viewed as a classification task, and an SVM is used. In [25] the authors avoid the heaviness of SVM by exploiting the advantages of Linear Discriminant Analysis (LDA).

2.2.4 Candidates re-ranking

The retrieved candidates can optionally undergo a post-processing phase of re-ranking, attempting in this way to sort them in a more accurate way. This is often done when the similarity search is done in an approximate way, and the re-ranking of the selected candidates can be performed in an exact manner. Another example in which re-ranking can be used, is when the features have been reduced by PCA, and the candidates can be sorted by using their distance with the query considering their full dimension, skipping therefore the PCA reduction. Re-ranking is also performed in cases where other data is known about the dataset, such as in [26], where Torii et al. exploit the location of their dataset to further improve the accuracy of the final response.

2.3 Other approaches

A lot of research has tried to approach the VPR problem with various image retrieval techniques that take into account the various problem of VPR, such as domain shift, while others tried to find solutions very different from the standard image retrieval, such as using 3D methods or transforming the localization task into classification.

2.3.1 Cross-appearance localization

Given the huge amount of interest in standard visual place recognition, some researchers also focused their work on solving the problem in different ways. This is the case of cross-domain VPR, where the database and query images belong to slightly different domains, and cross-view VPR, where the domain shift is much bigger.

2.3.1.1 Cross-domain

Cross-domain visual place recognition takes explicitly into account the differences in distribution between the database domain and the query domain. Some examples can be of database images taken as normal daylight photos, while the queries can belong to many different domains, such as photos taken at night, grey-scale images, or even painting (Figure 2.2). An example in this field is [25], in which the query is an old sketch or painting.

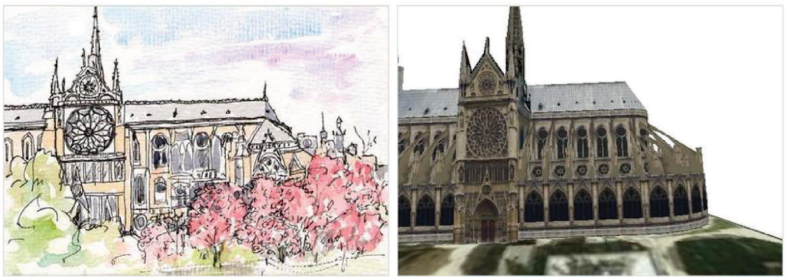


Figure 2.2: Cross-domain

2.3.1.2 Cross-view

Cross-view VPR focuses on the much more challenging task of having a database made of aerial views images (Figure 2.3). These images have the

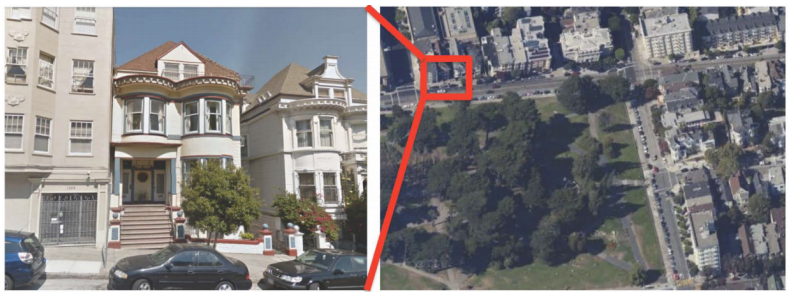


Figure 2.3: Cross-view

advantage to be available for any corner of the globe. However the query images are still taken on from the ground, thus the difficulty of the task. Several works have been done in the field, both with classical [27] and deep methods [28]. Interesting work from Bansal et al. [29] relies on image rectification for ground-level query images, in a way to make them more similar to aerial-view images.

2.3.2 Localization problem as a classification task

Researchers have also approached the VPR task as a classification task, which is, instead of finding the exact position of the query image, to find the region in which it was taken. This allows VPR to take a more global approach, for example by designing a system that predicts in which continent or country the image was taken. To this end, researchers [30] propose a variety of world-wide grids, as in Figure 2.4, and design systems that classify images according to their position in the grid.

2.3.3 3D-based methods

3D based methods use a database of geo-localized 3D models, which are used to find the location of the 2D query image. The databases can be built using a variety of sensors, such as RGB-D cameras, LIDARs, RADARs, etc., and are usually expensive to acquire, in terms of money cost (3D sensors are usually more expensive than cameras), time, computation and storage needs. These methods are often much slower than 2D-based methods, which makes them less scalable, but they give much higher 6-DOF accuracy [31]. This methods can be used together with 2D-based methods, as in [32], where the authors use the speed of 2D based methods to filter a limited amount of candidates from the database, and then 3D methods to have a precise estimate of the

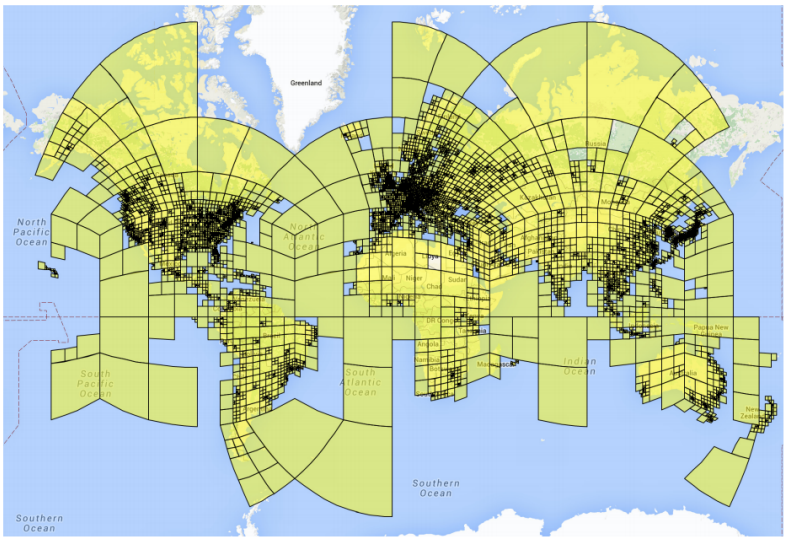


Figure 2.4: World-wide grid from [30]

query position.

Figure 2.5: 3D based method from [33]. These methods recover the exact pose of the query. The central image is the query and the surroundings represent the 3D database

2.4 Deep Learning approach

Recently, a lot of works have used a deep learning approach to extract the image descriptors. They are more robust w.r.t. the handcrafted ones, in the sense that, are robust in recognizing particular details of a place also with drastic changes in viewpoint and lighting conditions. This work is done by means of a convolutional neural network, truncated in a certain layer, where the features are then concatenated or pooled. In this way, CNN is used as a feature extractor instead of a classifier.
A lot of CNN-based methods has been developed, like the work of Weyand et al. [34] that divides the earth into cells and performs a classification task to solve VPR problem, but it is not too accurate. There are also works that use 3D point cloud scans, for example, retrieved by LIDAR sensors, to estimate the coordinates of a certain place, like PointNetVLAD [35].
Overall, the state of the art is represented by NetVLAD [1] (next paragraph 2.4.1).

2.4.1 State-of-the-art for VPR

Today, the state-of-the-art (SOTA) is achieved by NetVLAD [1] network, that is properly developed for the VPR task solved via image retrieval and is inspired by the VLAD representation [36]. The network is composed by a backbone, which is the VGG16 [3] for the best result, or AlexNet [37]. It is truncated at the last convolutional layer (conv5), where the features related to the entire image are extracted. At this point, a novel trainable VLAD layer is used to compact the features in a fixed length vector representation. The architecture is shown in the Figure 2.6. In [1] the Triplet Loss (Figure

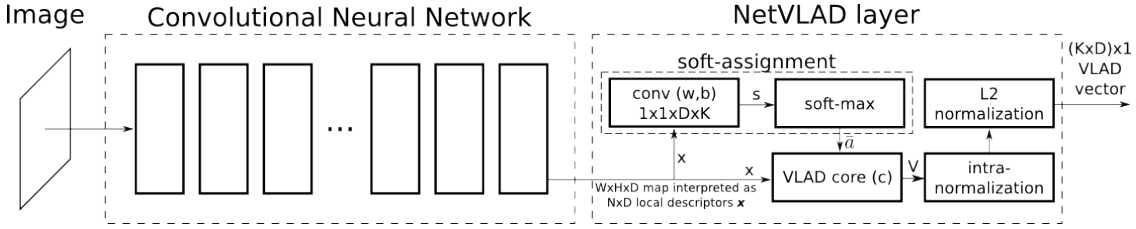


Figure 2.6: NetVLAD architecture [1]

2.7) is used, in a revisited form properly for the VPR task. They [1] call their loss 'Weakly supervised triplet ranking loss', that we can summarize in this way:

- Take a test query q;
- Between all its potential positives p_i^q, take the best matching potential positive $p_{i*}^q = \underset{p_i^q}{\operatorname{argmin}}\, d_\theta(q, p_i^q)$;
- Take its negative samples $\{n_j^q\}$;
- Compute the distance between the query and positive $d(q, \{p_{i*}^q\})$ and the ones between the query and all the negatives $d(q, \{n_j^q\})$;
- The goal becomes: $d(q, \{p_{i*}^q\}) < d(q, \{n_j^q\})\ \forall j$, so it's a ranking loss between each training triplet $(q, \{p_i^q\}, \{n_j^q\})$;
- So, their [1] loss is defined in this way:

$$L_\theta = \sum_j l\left(\min_i d_\theta^2(q, p_i^q) + m - d_\theta^2(q, n_j^q)\right)$$

  where l is the hinge loss and m is a constant figuring the margin. In this way all the negatives that have distance greater by a margin than the distance between the best matching potential positive and the query, will have loss 0. While when the margin is violated, the loss will be proportional to the amount of violation.

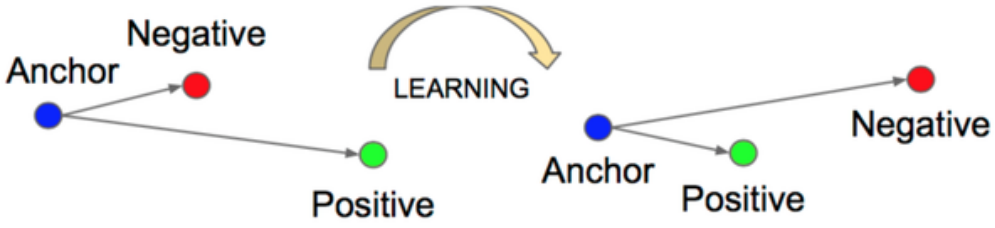


Figure 2.7: Triplet Loss: minimize the distance between the anchor and the positive sample, while maximize the distance between the same anchor and the negative sample.

Overall, the best advantage is related to the fact that their [1] network is end-to-end trainable and is developed for the specific VPR task, so it is very robust in order to find a certain street corner.

Part I

Chapter 3

Data collection

3.1 Building a dataset

After analysing the alternatives, we decided that 2D image retrieval would be the best option for our task. This because 3D datasets are too few and cover just small portion of the territory, and approaching the problem as a classification task, as in [30], would be too imprecise for our needs. Moreover, we decided not to use aerial views, because they usually rely on a stream of images, and their performance with a single image is still very low.
We then briefly explored the various options available in order to obtain a dataset with ground-level street images, that could be scalable and easy to use. The top options were Google Street View, Mapillary, Bing Streetside, Apple Maps, Open Street Cam. All these are websites or apps built on top of massive datasets, usually made of images taken by cars with a camera that drive around cities (Figure 3.1). However, the only ones covering all European cities are Google Street View and Mapillary. Of these two, the latter only has frontal and backward images, while the former has 360° spherical panoramas, which would be perfect for our case of study. Using Street View, we could build datasets covering almost any city of the western world (Figure 3.2).

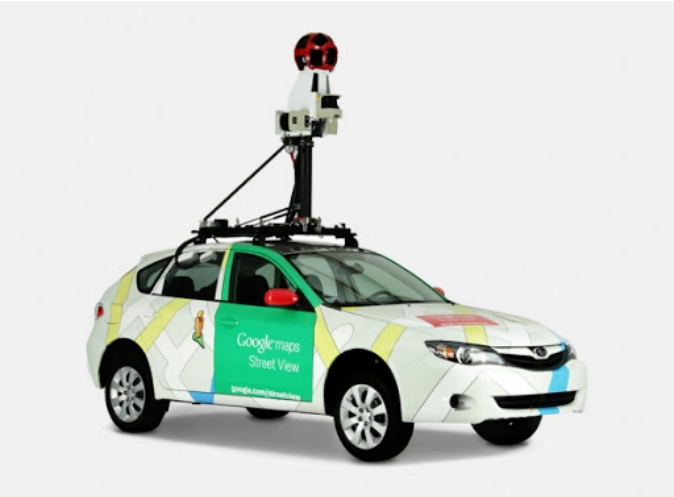


Figure 3.1: Car used by Google Street View to take panorama images

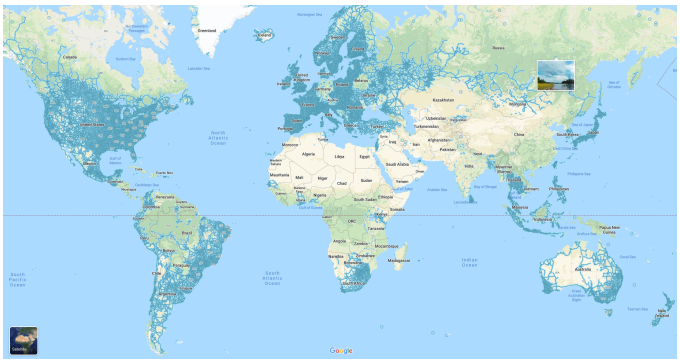


Figure 3.2: The streets in blue have been covered by Google Street View

3.2 Metadata and Google street view panoramas

The Google Street View dataset covers almost all Italian roads, and in most places the photos were taken several times throughout the years. We focused our research in the area of the city of Turin, to limit the amount of images needed for the task. The images provided by Street View are 360° equirectangular panoramas (Figure 3.3), that can be downloaded with various resolution, the highest being 6656x13312 pixels. Using Street View APIs we are able to download some metadata for each panorama, including its ID, latitude, longitude and the date when the image was taken. In this way we are able to build a large database with information about whole cities,

and its related dataset of images. We made scripts in order to simplify the download of the data, which take as parameters the coordinates of the area, and the number of processes that we want to launch, in order to speed up the download by multi-processing. In this way we are able to obtain city-wide metadata in a couple of hours, and their related panoramas in a couple of days, all without requiring high-end computers or incredibly fast internet connection.

Figure 3.3: Example of a Google Street View panorama

3.3 Google Street View Time Machine

A good algorithm for visual place recognition must be invariant to changes in viewpoint and lighting and to moderate occlusions. It should also learn to suppress confusing visual information such as clouds, vehicles and people, and to chose to either ignore vegetation or to learn a season-invariant vegetation representation. The Google Street View Time Machine can help to achieve this, as it can show the same location in different dates. The Time Machine is based on the fact that Google cars pass through the same roads multiple times over the years, and the dataset keeps accumulating images. So every passage of a Google car can be seen as a new layer of panoramas over the previous ones. On the Street View website, one can easily see the same location throughout the years, and using their APIs it is possible to download those panoramas. Figure 3.4 represents various images of the same location between 2008 and 2019. It can be noticed that the images have huge

differences, and by using them to train a neural network it can learn to ignore dynamic objects (such as vehicles), illumination changes, small point of view changes, changes in the vegetation and shadows.

(a) September 2008 (b) November 2011

(c) May 2012 (d) May 2014

(e) May 2015 (f) July 2016

(g) October 2017 (h) March 2019

Figure 3.4: Various images of the same place from 2008 to 2019

3.4 Removing the distortion

As shown in Figure 3.3, panoramas represent a 360° equirectangular projection of the surroundings of the camera. Although equirectangular projections

are an excellent way to map the surface of a sphere to a flat image, the resulting image appears very distorted, and thus very far from any undistorted image that we would later on use as query. This can represent a challenge for a retrieval neural network, as most visual place recognition networks are designed to handle database images and query images coming from the same domain. Undistorting an equirectangular panorama can be done only for small sections of the image (the field of view has to be less than 180°) by using the gnomonic projection, also known as rectilinear projection. The gnomonic projection is a nonconforming map projection obtained by projecting points P_1 (or P_2) on the surface of sphere from a sphere's center O to point P in a plane that is tangent to a point S. In Figure 3.5, S is the south pole, but can in general be any point on the sphere. Since this projection obviously sends antipodal points P_1 and P_2 to the same point P in the plane, it can only be used to project one hemisphere at a time. In a gnomonic projection, great circles are mapped to straight lines, and it represents the image formed by a spherical lens. In the projection of Figure 3.5, the point

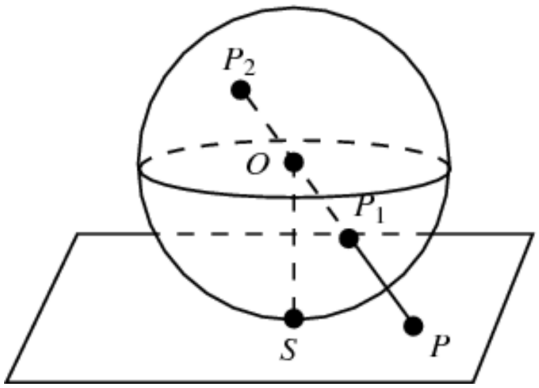


Figure 3.5: Gnomonic Projection

S is taken to have latitude and longitude $(\lambda, \phi) = (0,0)$ and hence lies on the equator. The transformation equations for the plane tangent at the point S having latitude ϕ and longitude λ for a projection with central longitude λ_0 and central latitude ϕ_1 are given by

$$x = \frac{\cos\phi \sin(\lambda - \lambda_0)}{\cos c}$$

$$y = \frac{\cos\phi_1 \sin\phi - \sin\phi_1 \cos\phi \cos(\lambda - \lambda_0)}{\cos c}$$

and c is the angular distance of the point (x, y) from the center of the projection, given by

$$\cos c = \sin \phi_1 \sin \phi + \cos \phi_1 \cos \phi \cos (\lambda - \lambda_0)$$

The inverse transformation equation are

$$\phi = \sin^{-1} \left(\cos c \sin \phi_1 + \frac{y \sin c \cos \phi_1}{\rho} \right)$$

$$\lambda = \lambda_0 + tan^{-1} \left(\frac{x \sin c}{\rho \cos \phi_1 \cos c - y \sin \phi_1 \sin c} \right)$$

where

$$\rho = \sqrt{x^2 + y^2}$$

$$c = tan^{-1} \rho$$

By using the inverse transformation equation we're able to undistort small tiles of the panorama (Figure 3.6), which we can then use for the image retrieval task.

(a) Distorted image (b) Undistorted image

Figure 3.6: Example of a tile of a panorama, distorted (a) and undistorted (b).

3.5 Cleaning the dataset

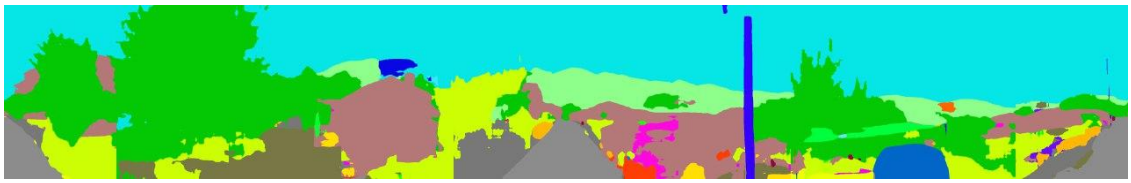
Even by keeping low resolutions for panoramas, a dataset collected with Google Street View can take huge amount of memory, and its processing will therefore need a long time and computation power. To reduce the size of the dataset, we remove parts of the panoramas that contain the least useful information: the top 4/13, and the bottom 5/13 (Figure 3.7), reducing the size to 4/13 of the original image. This helps to achieve huge memory sav-

Figure 3.7: An example of a panorama, and in the center the part that we save in the dataset

ings, but the dataset can still be of very large dimension. Just to give an example, a dataset of the Turin area, with around 1.000.000 panorama with resolution 512x3584 (removing the top and bottom parts), requires about 300GB of memory. Processing this amount of data can easily take days, and it is therefore necessary to make sure that the our dataset doesn't contain unneeded images. The focus of our research was to retrieve the position of given images which were taken in urban environments. Therefore, all the images containing non-urban scenes can be deleted from the dataset, and this can greatly reduce the amount of memory needed to store the dataset, depending on the area from which the images are downloaded. Obviously, due to the size of the dataset, this process of deleting rural panorama has to be done in an automated way. In order to do this, we decided to use semantic segmentation neural networks to estimate the number of pixels that belong to buildings for each panorama: if this number is below a threshold (we

chose 2.5% of the total amount of pixels as the threshold), it means that the photo was taken in a rural area and can therefore be deleted. The network that we used was a PSPNet [38] with a ResNet50 [4] backbone trained on the ADE20k dataset [39, 40], which showed good qualitative results on our Street View images. This method turned out to work very well for the task, and, when the geographical area selected to download images is big enough (more than 10.000km2), it helps to reduce the size of the dataset of up to 75%. Although the accuracy of PSPNet [38] is very high, its huge computational requirement made it unfeasible to be used of large scale datasets. Our solution was therefore to train a light-weight ResNet18 [4] to give an estimate of the number of pixels belonging to buildings for each panorama, changing therefore a semantic segmentation task to an inference task. To train the ResNet18 [4] we built a dataset of 288.000 panoramas using the PSPNet [38]. After a qualitative analysis we found that the ResNet18 [4] could identify the rural-area images equally well as the PSPNet [38], and was able to do this in a fraction of the time, giving us the possibility to use this method to clean even large scale datasets.

(a) Panorama with buildings



(b) Semantic segmentation of the panorama

(c) Semantic segmentation with buildings highlighted

Figure 3.8: Example of a panorama with many pixel belonging to buildings, and therefore it should be kept in the dataset

(a) Panorama without buildings

(b) Semantic segmentation of the panorama

(c) Semantic segmentation with buildings highlighted

Figure 3.9: Example of a panorama with no pixel belonging to buildings, and therefore it should be deleted

3.6 Our datasets

In order to train and test our algorithms we built three datasets, all of which contain images in the Turin area. All our experiments will be conducted on these three datasets, plus the Pitts30k dataset, used in the state-of-the-art paper of NetVLAD [1] and made publicly available by its authors.

3.6.1 Turin1M

Turin1M is a dataset of 949542 Street View panoramas covering the area between latitude 45.0 and 45.1 and longitude 7.6 and 7.7. This area covers most of the city of Turin, its whole center, and parts of nearby towns (Figure 3.10). The dataset is made of panoramas with resolution 512x3584 where the top and bottom part have been removed, as in Figure 3.7. The whole dataset requires about 300 GB of memory.

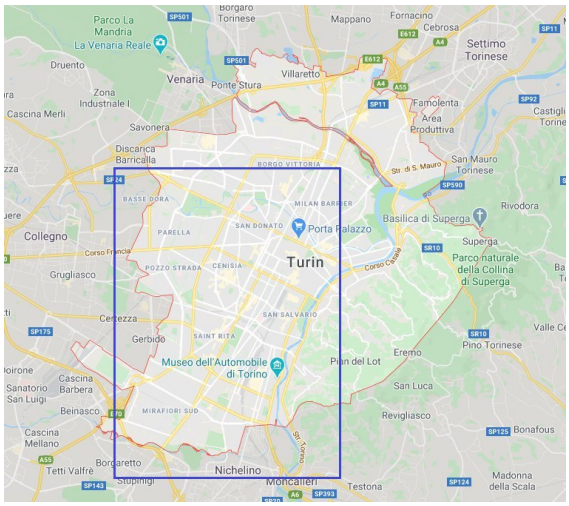


Figure 3.10: The blue rectangle represents the area of Turin1M, while the red boundary is the border of the city of Turin

3.6.2 Turin30k

When Turin1M dataset is completely downloaded, we have extracted three geographically disjoint subsets (train set, val set and test set) as in Figure 3.11, in order to fine-tune the network giving a reasonable amount of data and reducing also the retrieval time. First of all, during the training part of the CNN, we have followed what is done by the authors of [1], that consists in use images of the last year (2018/2019) for the query set, and the ones of the same places but in different years for the database set. This is a good approach, because in this way the network receives the image of the same places in a different instant, from different view-points and environment conditions. So, it generalizes and learns which features are useful and which are not. Further, the subsets are built providing the number of images inside the database equal to the one of the query set. This means that for each query there is exactly one positive corresponding database image (e.g. for 10 images of places of the database there are respectively the same 10 places of last year like queries). Since, the network will receive distorted images for the bag-of-visual-words like set and not-distorted images as queries, at test time, we have trained it to try to prepare the algorithm to this kind of situation. Then, we have trained using the cropped panos that are downloaded by means of unofficial API, so the distorted images, as a database for searching the similar images corresponding to the queries. While, the queries are retrieved using the official API, that returns images with straight lines.

Anyway, we have built three different subsets, following the previous rules, that are composed as described in Table 3.1.

Dataset	Database	Query set
Turin30k_train	10,000	10,000
Turin30k_val	10,000	10,000
Turin30k_test	10,000	10,000

Table 3.1: Number of images in subsets Turin30k.

In order to consider the whole Turin1M dataset, passing to a large-scale situation, a possible solution is to immediately filter the images that contain somethings similar to the queries, discarding all the rests. In this sense, the retrieval area should be reduced together with the needed amount of time. An example could be using such detection/semantic segmentation algorithms to produce annotations applied to filter over the entire dataset. Through these methods, we can retrieve what objects are represented in a photo, discarding all the dynamic ones, detecting texts, vegetation, car models, urban cleanings and so on. In this way, all the images that don't contain the annotation that is extracted by the query should be ignored at least in a first moment. Overall, we have postponed this kind of approaches to future works.

3.6.3 Turin81 - Third-domain dataset

The main idea of this thesis is to develop a software able to receive images from a third domain, like a mobile phone, and visualize the results produced by the algorithm.
Then, it is not trivial to collect a set of geo-localized images from an external device w.r.t. the one of the database. Overall, just to make some indicative tests, we have created a set of 81 images, called Turin81 (Figure 3.13), around the city-center of Turin captured with our iPhone 7 which has the GPS turned on. In this way, the information about the coordinates is stored in the EXIF metadata attached to the photo. Retrieve a consistent number of this kind of images remains a big problem. Some images of Turin81 are shown in Figure 3.12.
Moreover a set of 10000 images (1000 panos) is extracted by Turin1M to be used as database set during the retrieval task.
This images are also used to perform tests about occlusions.

Figure 3.11: The three rectangles represents the area selected to build the sets of training, validation and test

3.6.4 Extra

We have also transformed the previous dataset (Turin30k and Turin81) in two sets used for few tests.

3.6.4.1 Turin30k_undistorted

Turin30k_undistorted contains the same sets of Turin30k, but with the database images that are processed in our algorithm to remove the distortion (Section 3.4). This is used to perform the same tests and evaluate the domain shift given by the distortion in our task.

3.6.4.2 Turin81_occluded

Turin81_occluded contains different sets of images, all the ones collected in Turin81, where an artificial occlusion is applied (Section 4.4). The idea is to see if an occlusion can help the retrieving, occupying, for example, some possible dynamic object that distracts the algorithm. The occlusion is produced offline putting a black square/rectangle on the image. For the sake of

Figure 3.12: Example of Turin81 images

completeness we have applied the occlusion, in different percentages (12.5%, 50%, 75%) proportionally to the image size, in a different part of the image: on the left/right top/bottom corners, on the center, on the full bottom and randomly. Some examples are shown in Figure 3.14.
However, the occlusion analysis should be investigated better, a bigger collection of data should be used and some methods could be applied. For example given an image, apply online random occlusions, produce different copies of that query and perform voting on the results obtained. To complete our experiments and prepare the models for a third-domain, we have postponed the occlusion analysis to the future works.

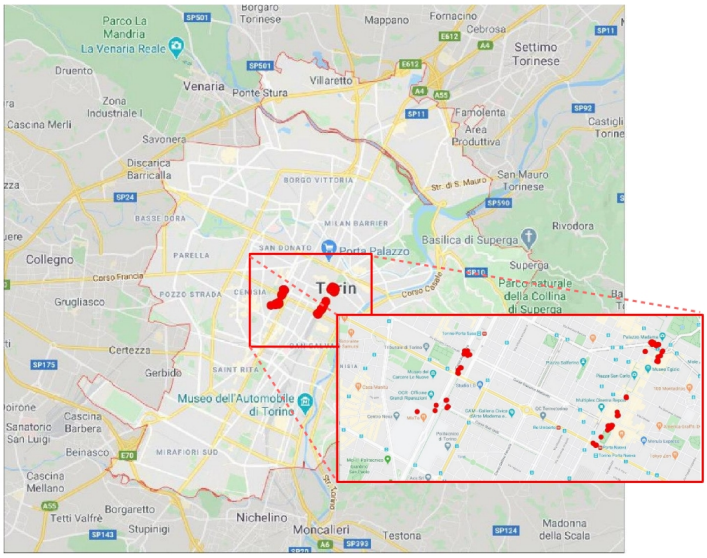
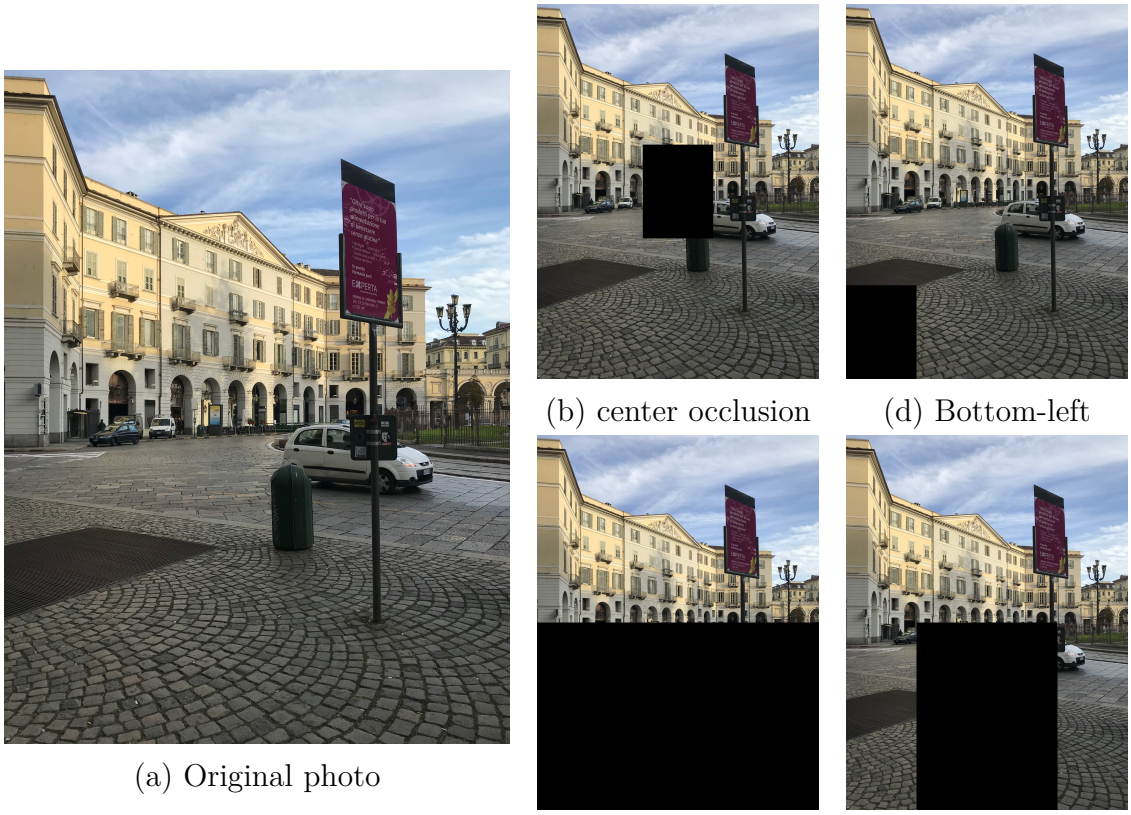


Figure 3.13: The red dots represent the location of the photos in Turin81

3.7 Interface to access data and test algorithm

Once, the dataset is built, we want to make easy for a novice user execute two kinds of operation: access to the database (not implemented for the thesis goal) and visualize the algorithm results, for an uploaded photo. In order to allow this, we have developed a web-like interface, the main page is shown in Figure 3.15. The software allows to choose a city, from the available (actually Torino), set some filter, for example, it's possible to select only a specific area of the selected city, through the map of the right, and finally explore the database visualizing all the image matching the filter, or upload a user photo and run the deep learning algorithm. At this point, the software remands the user to a new page, where the results are shown. Further, for what concerns the DL software-feature, the user can navigate on the map, that will show markers in the places predicted (Figure 3.16).
The website is developed in HTML, CSS, BOOTSTRAP and JavaScript for the client-side, while in PHP for the server one. The maps shown are implemented using the official Google API for JS, so they include all the features provided by Google Maps, like navigate through the streets with Google Street View.
Actually, the filter that works is the one on the geographical area. Once Torino is selected between the cities, the map is updated and a drawable



(a) Original photo

(b) center occlusion

(d) Bottom-left

(c) Bottom occlusion

(e) Bottom-center

Figure 3.14: Example of photo contained in Turin81.
a) Original photo made by iPhone7
b) Center occlusion with (h,w) = 12.5% of photo size
c) Bottom occlusion with (h,w) = (50% photo height, 100% photo width)
d) Bottom-left occlusion with (h,w) = 12.5% of photo size
e) Bottom-center occlusion with (h,w) = 25% of photo size

rectangle allows the user to select a specified zone. The corresponding coordinates are passed to the algorithm and are applied only at the end, so are shown only the predicted images that match the user-specified coordinates. The number of images returned by the algorithm is at most 20.
The improvement for the future is to apply the filter before the retrieval is performed.

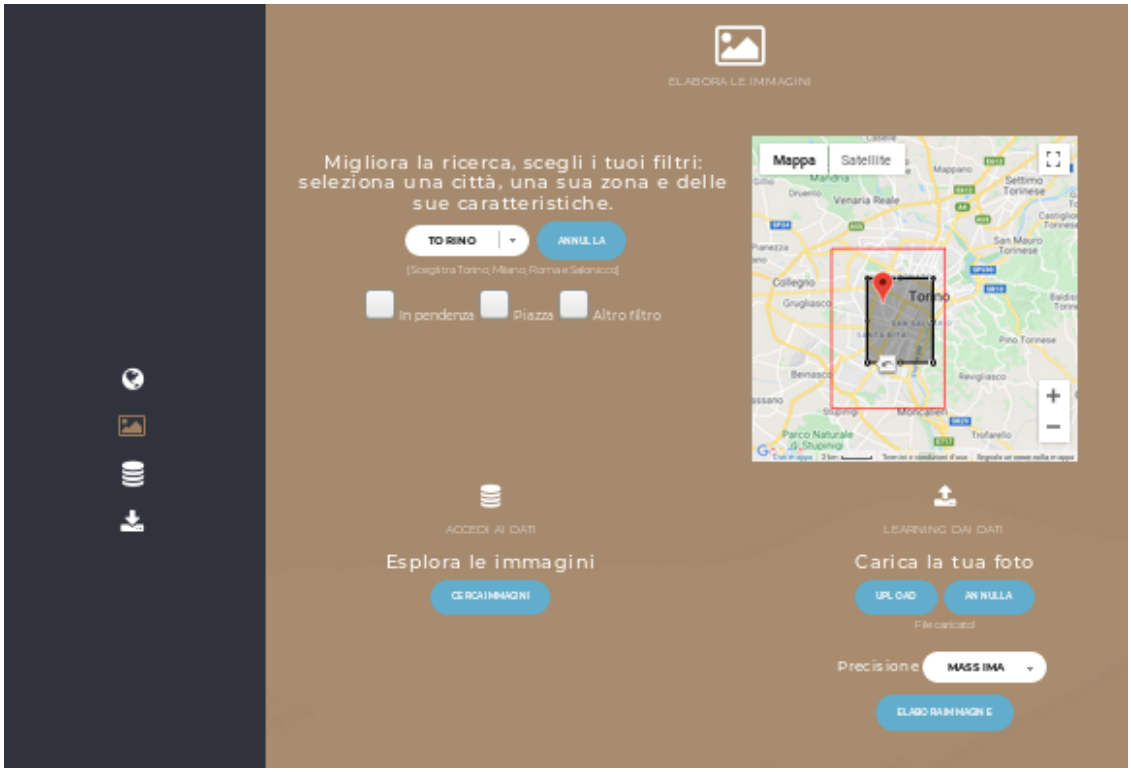


Figure 3.15: Main page of web site

Figure 3.16: Result-page of web site: on top there is the user-uploaded image, while in the red circles all the correct predictions. On the bottom the map shows the markers corresponding to the predictions done.

Part II

Chapter 4

Experiments

4.1 Hardware setup

The algorithms are trained and tested on computer with:

- GPU: NVIDIA GeForce GTX Titan X 12GB
- RAM: 64GB
- CPU(s): 12 x Intel(R) Core(TM) i7-5930K CPU @ 3.50GHz

4.2 NetVLAD on Pitts30k

4.2.1 Algorithm

In this section, we want to briefly present the algorithm used to train/test the network. It is downloaded from a GitHub repository [2], that contains a PyTorch version of NetVLAD [1] that was originally developed in Matlab. This code receives a set of arguments in order to run the algorithm for training, testing or clustering. Furthermore, from the command line, it's possible also decide the pre-trained network to use, the number of epochs to train, which dataset to test, the path to resume a checkpoint and many others settings.
We want to remember to the readers that each dataset used is composed of two subsets, the first one is the database i.e. the field of search of retrieval, while the second one is composed by the queries that will be searched. To train completely the network, two steps are needed:

- Run the algorithm in clustering mode:

  - Build model: it is compose by the pre-trained backbone (VGG16 [3] or AlexNet [37]) and a L2 normalization layer;
  - Take the dataset used to train: in this phase only the database set is used;
  - Create a dataloader sampling randomly from the database set and extract the descriptors;
  - Perform a K-means algorithm to cluster the extracted descriptors for a given K (from NetVLAD [1] paper K=64);
  - Store the cluster centroids.

- Run the algorithm in train mode:
  - Build model: the network is composed by the encoder (same backbone as before) and the pooling layer NetVLAD [1], that is initialized with the centroids previously extracted;
  - Take the dataset used to train: there are two classes for the dataset, the first one is for the database images, it contains the path for the images, how many database and query images there are, the coordinates and some others information. While the one for the queries contains all the stuff so the code to analyze each query and find its positive and negatives. This research is performed in two steps, the first one is a KNN with radius, used to find the positives images inside the 25 meters from the query coordinate, and the second step is a KNN based on the features, to keep the most positive between the ones previously found. This last KNN is executed inside the inherit ___getitem___ function of the dataset class, where the features of the database images are already passed. For the negative samples instead, the code considers all the samples out of 25 meters and keep randomly 1000 samples. Then, a KNN based on the query and negatives features is computed, extracting only 10 negative samples that are within margin w.r.t. the positive.
  - Start the training: for a certain number of epochs, it splits the queries set in subsets, for each of them the algorithm extracts the descriptors of the database saving that results in a cache variable. The dimension of the cache is given by the argument cacheBatchSize (default value is 24). The use and advantages of the cache are explained in [1]. At this point, the data loader of the considered subset of queries is created. The data loader of queries set returns tensors

corresponding to the query, its positive and its negatives. Then, the Triplet Loss is calculated and the backpropagation algorithm is performed. All this stuff is repeated for all the subsets of queries set.

- Each time an epoch ends, an evaluation is performed on a specified set (val set by default). A checkpoint is saved for each epoch and whether the accuracy is improved a further checkpoint called 'model_best' is stored.
- The metrics used to evaluate the algorithm is the recall@N. It consists in to give us how many predictions are correct after N ones. The value used to compare an epoch and evaluate if is the best one is the recall@5.

The training can be also restored from a checkpoint. In this case the option 'resume=path_of_checkpoint' should be defined when the program is launched. Then, all the flags, the optimizer parameters and so on, are resumed from the checkpoint.

- Run the algorithm in test mode:
  - Specify the checkpoint path in the resume option;
  - The model is built and the weights of the resumed checkpoint are loaded;
  - The specified dataset is loaded, both database and queries set. There is only one dataloader, because the samples are all collected in an sorted list, then the counter of database and queries images, give us the split of that two sets.
  - The batches pass through the network extracting the descriptors and search between the descriptor vector is executed. To improve the efficiency of this step, the faiss [23] python library is used.
  - The recall@N with N = [1, 5, 10, 20] is displayed.

4.2.2 Setup

- Dataset: Pitts30k (see Table 4.1);
- Backbone: VGG16 [3] (pre-trained on ImageNet) cropped at the last convolutional layer (conv5);

Dataset	Database	Query set
Pitts30k_train	10,000	7,416
Pitts30k_val	10,000	7,608
Pitts30k_test	10,000	6,816

Table 4.1: Number of images in subsets Pitts30k.

- Pooling: NetVLAD [1] initialized with the clusters extracted from Pitts30k 4.1;
- $Loss_{NetVLAD}$: Triplet ranking loss [1];
- Trainable layers: whole backbone + pooling;
- Number of clusters K: 64;
- Learning rate: 0.0001;
- Learning rate step: 5 epochs;
- Learning rate decay: 0.1;
- Momentum: 0.9;
- Weight decay: 0.001;
- Optimizer: SGD;
- Number of epochs: 30 (effective 25);
- Batch size: 4 tuples (each tuple contains the query, the positive and the negatives);
- CacheRefreshRate: 1000.

4.2.3 Results

Here, the results obtained using the pytorch implementation of NetVLAD [2], resuming the checkpoint after the training of Pitts30k:

	PITTS30K_VAL				
Backbone	R@1	R@5	R@10	R@20	Test Time
VGG	0,8527	0,9468	0,97	0,9838	0 : 21 : 44

Table 4.2: Test on Pitts30k_val (see Table 4.1): VGG = VGG16 [3].

	PITTS30K_TEST				
Backbone	R@1	R@5	R@10	R@20	Time
VGG	0,819	0,9123	0,9323	0,9575	0 : 19 : 59

Table 4.3: Test on Pitts30k_test (see Table 4.1): VGG = VGG16 [3].

4.3 NetVLAD on Turin81/Turin1M

4.3.1 Algorithm

Finally, in order to achieve our thesis' goal, we had to find a way to make our algorithms scalable to whole cities, and prove that they could work even on such huge datasets. Moreover, the final algorithm would have to be processed fast enough to be used through the GUI. Therefore, the classical pattern of online processing would be way too slow for our purpose (it might take days for a city).
One of the biggest challenges was due to the fact that even if each panorama was about 400 KB, its related vector of features, calculated with NetVLAD, was 1284 KB. This is because the vector of features is a vector of 32768 (2^{15}) floats of 4 bytes, and we take 10 crops from each panorama, and we therefore have 10 vectors of features. In order to calculate the euclidean distance between a vector of features of a query and all the vector of features of the database images, it is necessary to load the features of the whole database each time that we have a new query. This process could take days in a city-wide dataset. As an example, in our Turin1M dataset the images need about 300 GB of storage, while its vector of features need around 1200 GB. Just loading sequentially in memory such a huge dataset, without doing any computation on it, requires almost a day.
To overcome this memory obstacle, we decided to apply PCA on all the vectors of features, to reduce the dimension from 32768 to a smaller number, to make sure that these vectors could fit in RAM without losing too much accuracy. After trying different values for the final dimension of the vectors,

we decided 256 to be the best one: using PCA 256 the recall5 on the Pittsburgh dataset drops from 94.8% to 91.8%, but the dimension of the vectors of features could be reduced by 128 times. The memory required to store the features for the Turin1M dataset could therefore be reduced from 1200 GB to about 10 GB, which made them easy and fast to load to RAM. But even with the whole dataset loaded in RAM, finding the nearest neighbours with traditional methods would require too much time, as its complexity varies between O(nd+kn) and O(ndk), depending on the algorithmic choices, where n is the size of the database set, d the size of the query set, and k is KNN's hyperparameter. To speed up the KNN search, we found a great library developed by Facebook AI Research team named Faiss [23].
Faiss is a library for efficient similarity search and clustering of dense vectors. It is written in C++ with complete wrappers for Python/numpy. Faiss [23] has various implementation of KNN search, many of which give approximated results, and which can be performed on GPU. Moreover, Faiss [23] has a great tool for clustering, which can also be performed on GPU. The clustering can be of great help because given a vector of features from a query, we can perform the KNN only on the features of the database images which are closest to the features of the query. To find the closest ones, we just take those that belong to the same cluster to the query features, or to the neighboring ones. This helps to avoid computation on the farthest vectors of features, which obviously would not improve the recall, and would greatly slow down the algorithm.

Step	Duration	Offline/Online	Disk usage
Download metadata	3 hours	Offline	300 MB
Download panoramas	1 day	Offline	300 GB
Features computation	2 days	Offline	1200 GB
PCA-256 computation	2 days	Offline	10 GB
Faiss [23] clustering	2 hours	Offline	10 GB
Retrieval	0.1 sec - 2 min	Online	NA

Table 4.4: We show the duration of each step for the retrieval on Turin1M

When the faiss [23] index are built on the Turin1M dataset, the model can be tested through the GUI or by our dedicated step in the pipeline. This pipeline step is useful to test a set of images in a supervised environment, while with the GUI we can test one image at a time, without knowing the recall but watching the database images predicted. In order to process and

test big images like the ones produced by a smartphone, we have decided to perform a 5-crop during the pre-processing. Each crop has square shape equal to the 90% of the smallest side of its original image. Then each crop is resized to a smaller dimension (224x224 pixels) and finally tested. The final result is produced by a voting performed on the produced predictions of each single crop.
Summarizing, the test on each query is performed in this way:

- produce 5 crops;
- extract a certain number of predictions for each crop called *preds_per_crop*;
- the predictions that will be pooled for the entire query is *preds_per_query*, where *preds_per_query* < *preds_per_crop*);
- assign to all the *preds_per_crop* different weights: the first *preds_per_query* have weight A and to all the rest (*preds_per_crop* - *preds_per_query*) a weight B, where $A > B$. In this way the first preds_per_query predictions of each crop will have higher importance w.r.t. the others, so the most common predictions between the 5 crops will be extracted;
- only the *top_per_query* (at most *preds_per_query*) predictions are used to show the results and calculate the recalls.

4.3.2 Setup

- Model: NetVLAD [1] with VGG16 [3] pre-trained on Pitts30k (Tab.4.1);
- Database: Turin1M
- Query set: Turin81
- PCA: 256
- Precision: 100
- Predictions per crop: 1000
- Predictions per query: 100
- Top per query (max recall): 100
- Weights for voting: $3 \cdot preds\_per\_query + 2 \cdot (preds\_per\_crop - preds\_per\_query)$

4.3.3 Results

Here we want to show the results obtained using our pipelines, where the original NetVLAD [1] is used (explained in the Section 4.2), passing our third domain query set (Turin81), cropped in 5 tiles, each of them resized to 224x224 pixels and elaborated by means of voting for the final result. This is one of the first experiments, so the goal was to test our pipeline and see somethings about the occlusion presence. Best results could be achieved by bigger crops (e.g. 500x500 pixels) and with one of the models obtained in the previous experiments (Sections 4.5 4.6 4.7.2).

Set	R@1	R@5	R@20	R@50	R@100
$Turin81$	0,407	0,617	0,864	0,877	0,914

Table 4.5: Test of third domain set Turin81 over Turin1M dataset

4.4 NetVLAD on Turin81_occluded/Turin1M

4.4.1 Algorithm

In order to superficially test the influence of occlusions, we have developed a brief code to produce occlusions in a different part of the queries (Turin81). This is done offline, so when the new set of images containing occlusions is produced, they can be tested as described in Section 4.3. The algorithm to produce offline images with occlusion is structured as follow: first of all the occlusion has a different dimension, it is computed w.r.t. the size of the original image. Then a black mask is created, it has the same dimension of the original image, but with a certain area composed of black pixels. Then OpenCV library is used to perform a bitwise_and operation and finally perform an and operation between the original image and the mask.
The occlusions are produced on the corners (bottom-left, bottom-right etc.), on the centre of the image, on the side centre, over the entire bottom part and randomly. The percentage of the original size that is used to create the occlusions is 12.5% for the occlusions on the corners, on the centre and on the centre sides, while it is 25% for the occlusion that lies entirely on the bottom. We have seen that with a higher percentage it didn't make sense because the results were never better.

4.4.2 Setup

The algorithm and setup are the same ones explained in the Section 4.3.

4.4.3 Results

We can see from the Table 4.6 that the images with bottom occlusions performs quite better than the ones without occlusions (Table 4.5. This is comprehensible due to the fact that the occlusions on the bottom can hide some dynamic objects like the cars, that can distract the network during the test. Also the occlusions on the top don't perform bad, since they can obscure the sky and other characteristics not useful. For this reasons we have postponed this kind of analysis to future works, since it can be useful or however it can give us some idea.

4.5 ResNetVLAD on Pitts30k

4.5.1 Algorithm

We have tried to change the backbone from a VGG16 [3] to a ResNet18 [4], to speed up the test process and later try to add a self-supervised task. Here we are focusing on the algorithm used to change the backbone and the results obtained. The goal is achieve the baseline given by NetVLAD [1] with the VGG16 [3] as backbone.
We have downloaded the torchvision ResNet18 [4] architecture, loading the weights of the ResNet18 pre-trained on ImageNet. Then, we have tried different configurations for the backbone and the best results are achieved extracting the descriptors from the 4th block, training all the layers.
The rest of the algorithm works as we have already described in Section 4.2.1.

4.5.2 Setup

- Train dataset: Pitts30k_train (see Table 4.1);
- Backbone: ResNet18 [4] (pre-trained on ImageNet) cropped at the 4th convolutional block;
- Pooling layer: NetVLAD [1] initialized with the clusters extracted from Pitts30k (Tab. 4.1);
- $Loss_{NetVLAD}$: Triplet ranking loss [1];

- Trainable layers: last conv block of backbone + pooling layer;
- Number of clusters K: 64;
- Learning rate: 0.00001;
- Optimizer: Adam;
- Number of epochs: 15 (effective 4);
- Batch size: 4 tuples (each tuple contains the query, the positive and the negatives);
- CacheRefreshRate: 1000;
- Image size: original one (640x480).

4.5.3 Results

In the tables below, we show the results obtained. In the Table 4.7 there are the recalls on Pitts30k_val 4.1 and the difference w.r.t. the original model [2]. While in the Table 4.8 there are the recalls, but testing on Pitts30k_test.

	PITTS30K_VAL				
Backbone	R@1	R@5	R@10	R@20	Test Time
RN	0,875	0,953	0,969	0,981	0:04:09
RN-VGG	+0,022	+0,006	−0,001	−0,003	+0:17:35

Table 4.7: Test on Pitts30k_val (see Table 4.1): RN = ResNet18 [4] | VGG = VGG16 [3]. The first row is about the model with the new backbone, while the second one is the difference w.r.t. the original model.

	PITTS30K_TEST				
Backbone	R@1	R@5	R@10	R@20	Time
RN	0,855	0,924	0,943	0,959	0:03:49
RN-VGG	+0,036	+0,012	+0,011	+0,001	+0:16:10

Table 4.8: Test on Pitts30k_test (see Table 4.1): RN = ResNet18 [4] | VGG = VGG16 [3]. The first row is about the model with the new backbone, while the second one is the difference w.r.t. the original model.

As we can see from the Tables 4.7 and 4.8, and as it's known by literature, the model with the ResNet18 [4] is a lot faster than the one with

the VGG16 [3]. In fact, we can save more than 15 minutes at test time. Overall, the results obtained are pretty equal to the ones with the VGG16 [3], also considering that we have trained for less epochs and updating less parameters.

4.6 ResNetVLAD on Turin30k

4.6.1 Algorithm

In this new set of experiments, we want to use the network containing the ResNet18 [4] pre-trained on ImageNet to learn the features of our images 3.1. So, we have re-built the model, again cut at the 4th convolutional block, loading the weights learnt on ImageNet. This one is used as a normal pre-trained model, so the training is started with the default parameters as in Section 4.5. The NetVLAD layer is initialized with the centroids extracted from Turin30k dataset (as explained in 4.2.1). During the training, the learnable parameters are the ones contained in the NetVLAD pooling layer and in the 4th convolutional block of the ResNet18 [4].
We have kept the image dimension similar to the ones of Pittsburgh, it is 500x500 since we have also tried with smaller images (224x224) and the results were lower. All the rest about the algorithm works as described in Section 4.2.1.

4.6.2 Setup

- Train dataset: Turin30k_train (see Table 3.1);
- Backbone: ResNet18 [4] (pre-trained on Pitts30k 4.1) cropped at the 4th convolutional block;
- Pooling: NetVLAD [1] (pre-trained on Pitts30k 4.1);
- $Loss_{NetVLAD}$: Triplet ranking loss [1];
- Trainable layers: last conv-block of backbone + pooling;
- Number of clusters K: 64;
- Learning rate: 0.00001;
- Optimizer: Adam;
- Number of epochs: 15 (4 effective);
- Batch size: 4 tuples (each tuple contains the query, the positive and the negatives);

4.6.3 Results

The results here (Tables 4.9 4.10 4.11) shows the comparisons between the three networks, on Turin30k 3.1, where only the last one is trained on this dataset.
As expected the accuracy is higher when the network is trained on Turin30k dataset [3.1]. In the Table 4.11 we have reported the recalls obtained on the third domain dataset Turin81. The produced results by the ResNetVLAD trained on Turin30k is higher than the ones of the first ResNetVLAD trained on Pitts30k. In the next section a self-supervised task is attached to the ResNetVLAD to try to reduce the domain shift between Turin30k database set, Turin30k query set and Turin81.

4.6.3.1 Extra results

We have applied the technique explained in Section 3.4 to remove the distortion. The result should be investigated better, but training with the same setup, we obtain a very little improvement on the recalls. This tells us that there is not a lot of domain shift between the distorted images and the ones without. So Turin30k database and Turin30k query collections seem to be almost similar also for the network. In Table 4.12 there are the results obtained after the test on Turin30k_val and Turin30k_test.

	TURIN30K_VAL			
Backbone	R@1	R@5	R@10	R@20
VGG_{Pitts}	0,523	0,754	0,827	0,885
RN_{Pitts}	0,545	0,761	0,834	0,889
RN_{Turin}	**0,760**	**0,907**	**0,942**	**0,965**

Table 4.9: Test on Turin30k_val (see Table 4.1):
VGG_{Pitts} = VGG16 [3] pre-trained on ImageNet and then trained on Pitts30k (Table 4.1);
RN_{Pitts} = ResNet18 [4] pre-trained on ImageNet and then trained on Pitts30k (Table 4.1);
RN_{Turin} = ResNet18 [4] pre-trained on ImageNet and then trained on Turin30k (Table 3.1).

	TURIN30K_TEST			
Backbone	R@1	R@5	R@10	R@20
VGG_{Pitts}	0,5194	0,7159	0,7842	0,8451
RN_{Pitts}	0,547	0,736	0,800	0,858
RN_{Turin}	**0,767**	**0,913**	**0,948**	**0,970**

Table 4.10: Test on Turin30k_test (see Table 4.1):
VGG_{Pitts} = VGG16 [3] pre-trained on ImageNet and then trained on Pitts30k (Table 4.1);
RN_{Pitts} = ResNet18 [4] pre-trained on ImageNet and then trained on Pitts30k (Table 4.1);
RN_{Turin} = ResNet18 [4] pre-trained on ImageNet and then trained on Turin30k (Table 3.1).

	TURIN81			
Backbone	R@1	R@5	R@10	R@20
RN_{Pitts}	0,535	0,691	0,757	0,798
RN_{Turin}	0,523	**0,733**	**0,782**	**0,835**

Table 4.11: Test on third domain set Turin81:
RN_{Pitts} = ResNet18 [4] ImageNet and then trained on Pitts30k (Table 4.1);
RN_{Turin} = ResNet18 [4] pre-trained on ImageNet and then trained on Turin30k (Table 3.1).

Dataset	R@1	R@5	R@10	R@20
$Turin30k\_val$	0,773	0,914	0,945	0,965
$Turin30k\_test$	0,774	0,916	0,949	0,971

Table 4.12: Test on Turin30k_val and Turin30k_test with ResNetVLAD trained on Turin30k_train. All these sets contain database undistorted.

4.7 ResNetVLAD with self-supervision

In this section of experiments, we will explain a new approach based on a self-supervised task. It consists of using the source dataset for the main network task and the target dataset only for the second task. In this way, two losses are calculated and derived for the gradient. We hope to see an improvement of the accuracy when there is a domain shift if the network learns something about the target features just solving the secondary task.
So, we have added a new branch that solves the rotation task. It consists in giving to the network the input from the source and another one from the target that is randomly rotated of 90°, 180°, 270° or 0°. So the backbone receives both inputs and extracts the features. Then, the source feature maps are passed to the main task, NetVLAD in our case, while the target feature representations are given only to the rotation task. The Figure 4.1 shows the whole architecture.
This process is known as domain adaptation.

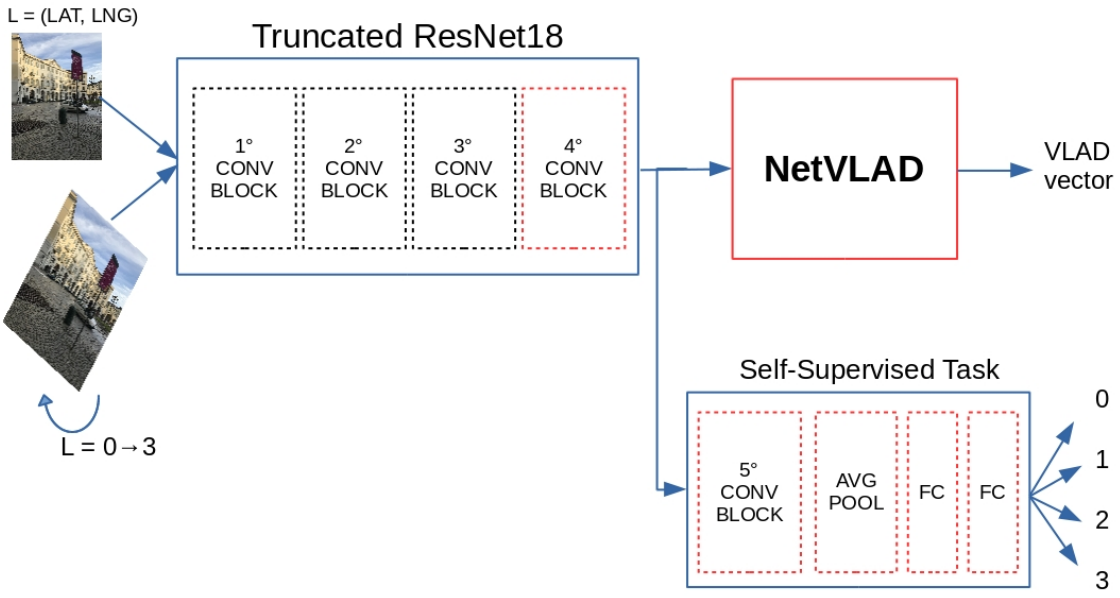


Figure 4.1: Architecture NetVLAD + Self-Sup branch. The layers trainable are in red.

4.7.1 Domain adaptation on Pitts30k/Turin81

Here, we have tried a domain adaptation when we train on Pitts30k. So, we want to improve the accuracy of ResNetVLAD on Turin81 when the main task learns to solve the VPR task on Pitts30k, overcoming the domain shift between Pittsburgh dataset and Turin dataset.

4.7.1.1 Algorithm

The main part of the algorithm is the same one explained in Section 4.2.1. To train the new network the clusters are immediately extracted from Pitts30k, to initialize the NetVLAD layer.
The source dataset is Pitts30k while for the target dataset we have passed Pitts30k query set, Turin81 database set and Turin81 query set. In this way, the rotation task receives both datasets and is solved across the domain. The network receives batches of source images with their original label (the coordinates) and batches of rotated images where the label is the rotation randomly applied (integer value from 0 to 3). The first ones pass inside the backbone and the NetVLAD layer, while the second ones are passed through the backbone and the self-supervised branch. Then, the rotation loss is computed and summed to the one of NetVLAD, both weighted of a certain value. The backpropagation is computed. Also, in this case, we have only trained the last convolutional block of the backbone.
To compare the effectiveness of DA, we have developed two kinds of experiments: in the first one (DA experiment 1) the self-supervised branch receives only the Pitts30k query set, while the second one (DA experiment 2) receives also the whole Turin81.

4.7.1.2 Setup

- Train dataset main task: Pitts30k_train (see Table 4.1);
- Dataset secondary task: Pitts30k_train query set, Turin81 database set, Turin81 query set;
- Backbone: ResNet18 [4] (pre-trained on ImageNet) cropped at the 4th convolutional block;
- Pooling branch: NetVLAD [1] (initialized with the centroids previously extracted);
- Self-sup branch: 5th conv-block of ResNet18 [4] + AvgPooling + Fully connected layers to outputs 4 classes;

- Trainable layers: last conv-block of backbone + pooling + self-sup;
- $Loss_{NetVLAD}$: Triplet ranking loss [1];
- $Loss_{SelfSup}$: Cross-Entropy loss;
- Final Loss: $Loss_{NetVLAD} + 0.5 \cdot Loss_{SelfSup}$
- Learning rate: 0.00001;
- Optimizer: Adam;
- Number of epochs: 11 (6 effective);
- Batch size: 4 tuples (each tuple contains the query, the positive and the negatives);
- Batch size self-sup: 256;
- Image size in NetVLAD branch: original one;
- Image size in Self-sup branch: Resized to 500x500 pixels and then randomly cropped to 100x100 pixels;
- Data augmentation in NetVLAD branch: Normalization;
- Data augmentation in Self-sup branch: Random crop at 100x100 pixels.

4.7.1.3 Results

Here the results (Tables 4.13 4.14 4.15 4.16 4.17) compare the training of this architecture when the self-supervised task receives only the Pitts30k query set (DA experiment 1) or also the whole Turin81 (DA experiment 2). The tests are made on the val and test sets of Pitts30k and Turin30k and on Turin81.
The accuracy is quite the same on the source domain (Pitts30k), while is improved on Turin81 of 6 points at recall@5.

	PITTS30K_VAL				
Experiment	R@1	R@5	R@10	R@20	Rotation
NO_DA	**0,875**	0,953	0,969	0,981	–
DA_1	0,868	0,952	0,967	0,980	**72%**
DA_2	0,873	**0,955**	**0,969**	**0,981**	70%

Table 4.13: Test on Pitts30k_val (see Table 4.1):
NO_DA is the model without the self-supervised branch (experiment in Section 4.5; DA_1 is the model with the self-sup branch trained with only the queries of Pitts30k_train; DA_2 is the model with the self-sup branch trained with the queries of Pitts30k_train and the Turin81 database and query images.

	PITTS30K_TEST				
Experiment	R@1	R@5	R@10	R@20	Rotation
NO_DA	**0,855**	**0,924**	**0,943**	**0,959**	–
DA_1	0,846	0,918	0,941	0,955	**71,7%**
DA_2	0,850	0,917	0,937	0,955	68%

Table 4.14: Test on Pitts30k_test (see Table 4.1):
NO_DA is the model without the self-supervised branch (experiment in Section 4.5; DA_1 is the model with the self-sup branch trained with only the queries of Pitts30k_train; DA_2 is the model with the self-sup branch trained with the queries of Pitts30k_train and the Turin81 database and query images.

	TURIN30K_VAL				
Experiment	R@1	R@5	R@10	R@20	Rotation
NO_DA	**0,545**	**0,761**	**0,834**	**0,889**	–
DA_1	0,520	0,733	0,807	0,868	69%
DA_2	0,523	0,742	0,814	0,871	87,3%

Table 4.15: Test on Turin30k_val (see Table 4.1):
NO_DA is the model without the self-supervised branch (experiment in Section 4.5; DA_1 is the model with the self-sup branch trained with only the queries of Pitts30k_train; DA_2 is the model with the self-sup branch trained with the queries of Pitts30k_train and the Turin81 database and query images.

	TURIN30K_TEST				
Experiment	R@1	R@5	R@10	R@20	Rotation
NO_DA	**0,547**	**0,736**	**0,800**	**0,858**	–
DA_1	0,536	0,727	0,790	0,848	66%
DA_2	0,531	0,724	0,786	0,846	**85,5%**

Table 4.16: Test on Turin30k_test (see Table 4.1):
NO_DA is the model without the self-supervised branch (experiment in Section 4.5; DA_1 is the model with the self-sup branch trained with only the queries of Pitts30k_train; DA_2 is the model with the self-sup branch trained with the queries of Pitts30k_train and the Turin81 database and query images.

	Turin81				
Experiment	R@1	R@5	R@10	R@20	Rotation
NO_DA	**0,535**	0,691	0,757	0,798	–
DA_1	0,469	0,741	0,765	0,790	64%
DA_2	0,506	**0,753**	**0,790**	**0,827**	**77,8%**

Table 4.17: Test on third domain set Turin81:
NO_DA is the model without the self-supervised branch (experiment in Section 4.5; DA_1 is the model with the self-sup branch trained with only the queries of Pitts30k_train; DA_2 is the model with the self-sup branch trained with the queries of Pitts30k_train and the Turin81 database and query images.

4.7.2 Domain adaptation on Turin30k/Turin81

4.7.2.1 Algorithm

The main part of the algorithm is the same one explained in Section 4.2.1. To train the new network the clusters are immediately extracted from Turin30k, to initialize the NetVLAD layer.
The main problem in our project is about the unsupervised test samples that belong to different domains. Moreover, we have few images of the third domain, that we cannot use to train the model and we have used distorted images in the database to compare the results against the ones obtained with Turin1M. So, the idea is to inject more information as possible to the network about our datasets. Summarizing, the focus on the very important features (e.g. the ones of the buildings) is already partially done in the SOTA [1], using the Google Time Machine, then we use the distorted images as database set also during the training. Finally, to learn more about these features and to adapt the network to the different domains of Turin datasets a self-supervised task is used.
The set of images used here is composed by only the queries that are seen also by the NetVLAD branch (self-sup experiment 1) or by the queries and the images of the third domain Turin81 (self-sup experiment 2).
The idea concerning the two self-supervised experiments is related to the fact that we want to see if NetVLAD [1] performs better on the third domain photos, seeing these only in the self-supervised branch, without the real label (coordinates). Since the third domain images are bigger than the ones of Turin30k Tab. 3.1, we have performed at training time a resized to 500 pixels

for the smallest edge, while the other one is respectively rescaled. Overall, the first layer in the self-supervised task receives only a random crop (100x100 pixels) from the original images.

4.7.2.2 Setup

The setup of the self-supervised branch is the best one belong our set of experiments which different configurations.

- Train dataset: Turin30k_train (see Table 3.1);
- Backbone: ResNet18 [4] (pre-trained on ImageNet) cropped at the 4th convolutional block;
- Pooling: NetVLAD [1] (initialized with the centroids previously extracted);
- Self-sup: 5th conv-block of ResNet18 [4] + AvgPooling + Fully connected layers to outputs 4 classes;
- Trainable layers: last conv-block of backbone + pooling + Self-sup;
- $Loss_{NetVLAD}$: Triplet ranking loss [1];
- $Loss_{SelfSup}$: Cross-Entropy loss;
- Final Loss: $Loss_{NetVLAD} + 0.5 \cdot Loss_{SelfSup}$
- Learning rate: 0.00001;
- Optimizer: Adam;
- Number of epochs: 11 (6 effective);
- Batch size: 4 tuples (each tuple contains the query, the positive and the negatives);
- Batch size self-sup: 256;
- CacheRefreshRate: 1000;
- Image size in NetVLAD branch: original one (500x500 pixels);
- Image size in Self-sup branch: original one for the images of Turin30k Tab. 3.1 (500x500 pixels), while it is resized to 500 pixels for the Turin81 (Nx500 or 500xN pixels, where N is the re-scaled side w.r.t. 500). In both cases the images are finally randomly cropped to 100x100 pixels.
- Data augmentation in Self-sup branch: Random crop at 100x100 pixels.

4.7.2.3 Results

Here we show the results obtained with the two self-supervised experiments, providing the recalls on the Turin30k val (Tab.4.18) and test (Tab.4.19) sets and also on the third domain set (Tab.4.20).

	TURIN30K_VAL				
Experiment	R@1	R@5	R@10	R@20	Rotation
NO_SS	0,760	0,907	0,942	0,965	–
$SelfSup_1$	0,748	0,901	0,937	0,961	93,43%
$SelfSup_2$	**0,749**	**0,904**	**0,939**	**0,962**	**93,54%**

Table 4.18: Test on Turin30k_val (see Table 4.1):
NO_SS is the model trained on Turin30k without the self-supervised branch (experiment in Section 4.6);
$SelfSup_1$ is the model with the self-sup branch trained with only the queries of Turin30k_train;
$SelfSup_2$ is the model with the self-sup branch trained with the queries of Turin30k_train and the third-domain images.

	TURIN30K_TEST				
Experiment	R@1	R@5	R@10	R@20	Rotation
NO_SS	0,767	0,913	0,948	0,970	–
$SelfSup_1$	0,758	0,906	0,946	0,969	**96,87%**
$SelfSup_2$	0,758	**0,908**	0,946	0,969	92,09%

Table 4.19: Test on Turin30k_test (see Table 4.1):
NO_SS is the model trained on Turin30k without the self-supervised branch (experiment in Section 4.6);
$SelfSup_1$ is the model with the self-sup branch trained with only the queries of Turin30k_train;
$SelfSup_2$ is the model with the self-sup branch trained with the queries of Turin30k_train and the third-domain images.

	Turin81				
Experiment	R@1	R@5	R@10	R@20	Rotation
NO_SS	0,523	0,733	0,782	0,835	–
$SelfSup_1$	0,519	0,728	0,790	0,852	92,59%
$SelfSup_2$	0,519	0,728	0,790	**0,877**	**96,30%**

Table 4.20: Test on third domain set Turin81:
NO_SS is the model trained on Turin30k without the self-supervised branch (experiment in Section 4.6);
$SelfSup_1$ is the model with the self-sup branch trained with only the queries of Turin30k_train;
$SelfSup_2$ is the model with the self-sup branch trained with the queries of Turin30k_train and the third-domain images.

	Turin81 WITH OCCLUSION				
Occlusion	AVG-R@1	AVG-R@5	AVG-R@20	AVG-R@50	AVG-R@100
$Bottom_{12.5\%}$	0,403	**0,634**	0,811	**0,881**	**0,916**
$Center_{12.5\%}$	0,350	0,556	0,695	0,798	0,852
$Top_{12.5\%}$	0,387	0,568	0,769	0,835	0,909
$FullBottom_{25\%}$	0,321	0,543	0,704	0,728	0,827
Random	0,260	0,442	0,627	0,700	0,781

Table 4.6: Test of third domain set Turin81 with artificial occlusion over Turin1M dataset:
In bold the results higher than the ones without occlusions (Table 4.5).
$AVG - R@N$: it's the Recall@N average calculated between the recalls obtained between the images with occlusion on the corners and on the centre side (e.g. $\frac{Recall\_bottom\_left+Recall\_bottom\_center+Recall\_bottom\_right}{3}$).
$FullBottom_{25\%}$: occlusion on the bottom with dimension $(h, w) = (25\%imageheight, 100\%imagewidth)$.
Random: random occlusions, with random dimension, spread on the image.

Chapter 5

Conclusions and future works

In this thesis we have focused on the visual place recognition (VPR) task, implementing deep learning algorithms and developing software to make easier the dataset creation related to Google Street View images and the geolocalization of user-updated photo.
As explained in this paper, our pipeline allows to download metadata and images for all the places covered by Google Street View and perform offline many expensive operations, reducing the amount of time during the deep learning algorithm test.
We have seen that the datasets used (Pitts30k and Turin30k) are not so different. The approach based on self-supervision helps the training, but the result concerning the domain adaptation suggest us that there is not a lot of gap between these domains. We will continue to investigate on this topic, trying other self-supervision tasks and comparing more different domains like day/night that are more challenging.
Moreover, we have already started removing of distortion inside Turin1M, and from the first tests, it seems to not affect a lot the results.
Also, the analysis of the occlusion should be continued with other techniques based on what we have already seen during this thesis.
The most interesting works that should be immediately tried from our point of views are:

- Attention module in the network, to focus the prediction on the features related to the static objects;
- Reduce the number of features used during the VPR task, with some

machine learning methods (e.g. PLDA-like), to reduce the amount of time needed during the retrieval;

- Annotation process: semantic segmentation or object detection networks to annotate and then filter the database images used during the retrieval;
- Domain adaptation / Domain generalization: other methods and selfsupervised tasks to improve the accuracy of extremely distinct domains as day/night or with/without snow/rain and so on.

Bibliography

[1] Relja Arandjelovic, Petr Gronát, Akihiko Torii, Tomás Pajdla, and Josef Sivic. «NetVLAD: CNN Architecture for Weakly Supervised Place Recognition.» In: *CVPR*. IEEE Computer Society, 2016, pp. 5297–5307. ISBN: 978-1-4673-8851-1. URL: http://dblp.uni-trier.de/db/conf/cvpr/cvpr2016.html#ArandjelovicGTP16.

[2] Nanne. *pytorch-NetVlad*. URL: https://github.com/Nanne/pytorch-NetVlad.

[3] Karen Simonyan and Andrew Zisserman. «Very Deep Convolutional Networks for Large-Scale Image Recognition». In: *CoRR* abs/1409.1556 (2014). URL: http://arxiv.org/abs/1409.1556.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. 2015. eprint: arXiv:1512.03385.

[5] David Lowe. «Distinctive Image Features from Scale-Invariant Keypoints». In: *International Journal of Computer Vision* 60 (Nov. 2004), pp. 91–. DOI: 10.1023/B:VISI.0000029664.99615.94.

[6] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. «Speeded-up robust features (SURF)». In: *Computer Vision and Image Understanding* 110 (June 2008), pp. 346–359. DOI: 10.1016/j.cviu.2007.09.014.

[7] Relja Arandjelovic and Andrew Zisserman. «All about VLAD». In: June 2013, pp. 1578–1585. DOI: 10.1109/CVPR.2013.207.

[8] Aude Oliva and Antonio Torralba. «Modeling the Shape of the Scene: A Holistic Representation of the Spatial Envelope.» In: *International Journal of Computer Vision* 42.3 (2001), pp. 145–175. URL: http://dblp.uni-trier.de/db/journals/ijcv/ijcv42.html#OlivaT01.

[9] Albert Gordo, Jon Almazan, Jerome Revaud, and Diane Larlus. *End-to-end Learning of Deep Visual Representations for Image Retrieval*. 2016. eprint: arXiv:1610.07940.

[10] Jiri Matas, Ondrej Chum, Martin Urban, and Tomas Pajdla. «Robust Wide Baseline Stereo from Maximally Stable Extremal Regions». In: *Image and Vision Computing* 22 (Sept. 2004), pp. 761–767. DOI: 10.1016/j.imavis.2004.02.006.

[11] Navneet Dalal and Bill Triggs. «Histograms of Oriented Gradients for Human Detection». In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2005)* 2 (June 2005).

[12] Charbel Azzi, Daniel Asmar, Adel Fakih, and John Zelek. «Filtering 3D Keypoints Using GIST For Accurate Image-Based Localization». In: Jan. 2016, pp. 127.1–127.12. DOI: 10.5244/C.30.127.

[13] J. Sivic and A. Zisserman. «Video Google: A Text Retrieval Approach to Object Matching in Videos». In: vol. 2. Nov. 2003, 1470–1477 vol.2. DOI: 10.1109/ICCV.2003.1238663.

[14] Hervé Jégou, Matthijs Douze, Cordelia Schmid, Theme COG, and Equipe-Projet Lear. «Hamming Embedding and Weak Geometry Consistency for Large Scale Image Search - Extended version». In: (Oct. 2008).

[15] Hervé Jégou, Matthijs Douze, Jorge Sánchez, Patrick Perez, and Cordelia Schmid. «Aggregating Local Image Descriptors into Compact Codes». In: *IEEE transactions on pattern analysis and machine intelligence* 34 (Dec. 2011). DOI: 10.1109/TPAMI.2011.235.

[16] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. «On the burstiness of visual elements». In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR '09)* (June 2009). DOI: 10.1109/CVPRW.2009.5206609.

[17] Relja Arandjelović and Andrew Zisserman. «DisLocation: Scalable Descriptor Distinctiveness for Location Recognition». In: Nov. 2014, pp. 188–204. ISBN: 978-3-319-16816-6. DOI: 10.1007/978-3-319-16817-3_13.

[18] Ke Yan, Yaowei Wang, Dawei Liang, Tiejun Huang, and Yonghong Tian. «CNN vs. SIFT for Image Retrieval: Alternative or Complementary?» In: Oct. 2016, pp. 407–411. DOI: 10.1145/2964284.2967252.

[19] Pilailuck Panphattarasap and Andrew Calway. *Visual place recognition using landmark distribution descriptors*. 2016. eprint: arXiv:1608.04274.

[20] Ali Sharif Razavian, Josephine Sullivan, Stefan Carlsson, and Atsuto Maki. *Visual Instance Retrieval with Deep Convolutional Networks*. 2014. eprint: arXiv:1412.6574.

[21] Artem Babenko and Victor Lempitsky. *Aggregating Deep Convolutional Features for Image Retrieval*. 2015. eprint: arXiv:1510.07493.

[22] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. «Object retrieval with large vocabularies and fast spatial matching». In: June 2007. DOI: 10.1109/CVPR.2007.383172.

[23] Jeff Johnson, Matthijs Douze, and Hervé Jégou. «Billion-scale similarity search with GPUs». In: *arXiv preprint arXiv:1702.08734* (2017).

[24] Colin Mcmanus, Ben Upcroft, and Paul Newmann. «Scene Signatures: Localised and Point-less Features for Localisation». In: July 2014. DOI: 10.15607/RSS.2014.X.023.

[25] M. Aubry, B. Russell, and J. Sivic. «Painting-to-3D Model Alignment Via Discriminative Visual Elements». In: *ACM Transactions on Graphics* (2013). Pre-print, accepted for publication.

[26] Akihiko Torii, Josef Sivic, and Tomas Pajdla. «Visual localization by linear combination of image descriptors». In: Nov. 2011, pp. 102–109. DOI: 10.1109/ICCVW.2011.6130230.

[27] Tsung-Yi Lin, Serge Belongie, and James Hays. «Cross-View Image Geolocalization». In: June 2013, pp. 891–898. DOI: 10.1109/CVPR.2013.120.

[28] Nam Vo and James Hays. *Localizing and Orienting Street Views Using Overhead Imagery*. 2016. eprint: arXiv:1608.00161.

[29] Mayank Bansal, Harpreet Sawhney, Hui Cheng, and Kostas Daniilidis. «Geo-localization of street views with aerial image databases». In: Nov. 2011, pp. 1125–1128. DOI: 10.1145/2072298.2071954.

[30] Tobias Weyand, Ilya Kostrikov, and James Philbin. «PlaNet - Photo Geolocation with Convolutional Neural Networks». In: (2016). DOI: 10.1007/978-3-319-46484-8_3. eprint: arXiv:1602.05314.

[31] Torsten Sattler, Will Maddern, Carl Toft, Akihiko Torii, Lars Hammarstrand, Erik Stenborg, Daniel Safari, Masatoshi Okutomi, Marc Pollefeys, Josef Sivic, Fredrik Kahl, and Tomas Pajdla. *Benchmarking 6DOF Outdoor Visual Localization in Changing Conditions*. 2017. eprint: arXiv:1707.09092.

[32] Paul-Edouard Sarlin, Cesar Cadena, Roland Siegwart, and Marcin Dymczyk. *From Coarse to Fine: Robust Hierarchical Localization at Large Scale*. 2018. eprint: arXiv:1812.03506.

[33] Youji Feng, Lixin Fan, and Yihong Wu. «Fast Localization in Large Scale Environments Using Supervised Indexing of Binary Features». In: *IEEE Transactions on Image Processing* 25 (Nov. 2015), pp. 1–1. DOI: 10.1109/TIP.2015.2500030.

[34] Tobias Weyand, Ilya Kostrikov, and James Philbin. «PlaNet - Photo Geolocation with Convolutional Neural Networks». In: (2016). DOI: 10.1007/978-3-319-46484-8_3. eprint: arXiv:1602.05314.

[35] Mikaela Angelina Uy and Gim Hee Lee. *PointNetVLAD: Deep Point Cloud Based Retrieval for Large-Scale Place Recognition*. 2018. eprint: arXiv:1804.03492.

[36] Cordelia Schmid Hervé Jégou Matthijs Douze and Patrick Pérez. «Aggregating local descriptors into acompact image representation». In: *CVPR 2010 - 23rd IEEE Conference on Computer Vision and PatternRecognition, Jun 2010, San Francisco, United States* (2010). URL: https://hal.inria.fr/inria-00548637.

[37] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. «ImageNet Classification with Deep Convolutional Neural Networks». In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[38] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. *Pyramid Scene Parsing Network*. 2016. eprint: arXiv:1612.01105.

[39] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. «Semantic understanding of scenes through the ade20k dataset». In: *arXiv preprint arXiv:1608.05442* (2016).

[40] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. «Scene Parsing through ADE20K Dataset». In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017.